Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ερωτήματα προτύπων σε βάσεις σημασιολογικών τροχιών με χωρο-χρονικά και λεκτικά κριτήρια σε υβριδικές R-Tree δομές στο Graph ΣΔΒΔ Neo4j** **(Spatio-Temporal-Keyword Pattern queries with hybrid R-Tree indexes in Neo4j Graph DBMS)**
Ονοματεπώνυμο Φοιτητή	**Φραγκίσκος Γρυλλάκης του Νικολάου**
Αριθμός Μητρώου	**ΜΠΣΠ12014**
Κατεύθυνση	**Συστήματα Υποστήριξης Αποφάσεων**
Επιβλέπων	**Πελέκης Νικόλαος, Λέκτορας**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Νικόλαος Πελέκης Λέκτορας	Ιωάννης Θεοδωρίδης Καθηγητής	Χρήστος Δουλκερίδης Λέκτορας

Περίληψη

Η αυξανόμενη χρήση συσκευών με GPS δυνατότητες έχει οδηγήσει στην ανάγκη αποθήκευσης και διαχείρισης μεγάλου όγκου χωροχρονικών δεδομένων που μπορούν να χρησιμοποιηθούν από κατάλληλες υπηρεσίες και εφαρμογές για την εξαγωγή από αυτά πληροφοριών και συμπερασμάτων. Η άντληση πληροφοριών απαιτεί τον ορισμό κατάλληλων αφαιρετικών δομών προκειμένου να είναι δυνατή η αποτελεσματική διαχείριση τους και η πραγματοποίηση κατάλληλων επερωτήσεων σε αυτά για την εξαγωγή γνώσης από αυτά. Με τον όρο σημασιολογικές τροχιές ορίζεται μία τέτοια αφαιρετική δομή που περιέχει διαδοχικές ακολουθίες από τμήματα κινήσεων ενός χρήστη που μπορούν να χαρακτηρισθούν ως υποτροχιές κινήσεως ή μη κινήσεως, αναλόγως αν το αντικείμενο τη συγκεκριμένη χρονική στιγμή πραγματοποιούσε κίνηση ή όχι. Η δημιουργία κατάλληλων επερωτήσεων σε βάσεις δεδομένων σημασιολογικών τροχιών απαιτεί την εύρεση κατάλληλων αλγοριθμικών τρόπων αποδοτικής εξαγωγής των απαιτούμενων πληροφοριών λαμβάνοντας υπόψη απαραίτητα τον μεγάλο όγκο διακινούμενων πληροφοριών καθώς και την ιδιαίτερη μορφή των δεδομένων αυτών. Μία σημαντική κατηγορία επερωτήσεων είναι τα spatiotemporal keyword pattern queries. Με την συγκεκριμένη κατηγορία ορίζονται ερωτήματα που αφορούν την αναζήτηση σημασιολογικών τροχιών με συγκεκριμένα χωροχρονικά και λεκτικά κριτήρια στις επιμέρους σημασιολογικές υποτροχιές τους. Η αναπαράσταση της ακολουθίας των υποτροχιών έχει τη μορφή regular expression. Για την αποτελεσματική υλοποίηση των spatiotemporal keyword queries απαιτείται η δημιουργία κατάλληλων ευρετηριακών δομών που να λαμβάνουν υπόψη τη τριπλή φύση (λεκτική και χωροχρονική) των ανωτέρω ερωτημάτων. Σε αυτή τη μεταπτυχιακή διατριβή με τη χρησιμοποίηση της graph βάσης δεδομένων Neo4j δημιουργήθηκε ένα κατάλληλα προσαρμοσμένο ευρετήριο για την αντιμετώπιση των συγκεκριμένων queries. Ειδικότερα, πραγματοποιήθηκε επέκταση του τρόπου ευρετηρίασης των χωρικών αντικειμένων της library Neo4j Spatial (R-Tree ευρετήριο). Για την αποτελεσματική αντιμετώπιση των ανωτέρω ερωτημάτων είναι απαραίτητο το ευρετήριο να έχει υβριδική μορφή, να περιλαμβάνει δηλαδή το R-Tree που θα χρησιμοποιηθεί εκτός από χωροχρονική πληροφορία και λεκτική. Η δημιουργία των λεκτικών ευρετηρίων γίνεται μέσω της library Lucene. Η Lucene είναι μία open-source βιβλιοθήκη υλοποιημένη στη γλώσσα προγραμματισμού Java που χρησιμοποιεί ανεστραμμένα ευρετήρια για την αναζήτηση και ανάκτηση πληροφορίας σε μία συλλογή από κείμενα. Μετά από την παρουσίαση του τρόπου λειτουργίας και διαχείρισης των χωροχρονικών δεδομένων του υβριδικού ευρετηρίου παρουσιάζονται κατάλληλες αλγοριθμικές επιλύσεις του ερωτήματος. Οι συγκεκριμένες αλγοριθμικές επιλύσεις του spatiotemporal keyword pattern query εκμεταλλεύονται διαφορετικές παραλλαγές του υβριδικού index που έχει δημιουργηθεί προκειμένου να αντιμετωπιστεί αποτελεσματικά και αποδοτικά η επίλυση του query.

Abstract

The increasing use of devices with GPS capabilities has led to the need of storing and managing large amounts of spatiotemporal data that can be used by appropriate services and applications for the export of useful information. The information discovery requires definition of appropriate data structures, such as semantic trajectories, in order to enable effective management of the spatiotemporal data and the implementation of appropriate queries for data mining. A semantic trajectory has consecutive sequences of sub-trajectories of a user that can be classified as a section of movement or non-movement, depending on whether the object at that time was moving or not. The creation of efficient database queries for semantic trajectories searching requires finding appropriate data miming algorithms for the extraction of movement information that they take into consideration the large volumes of information of this particular form of data. An important category of these mining queries are the spatiotemporal keyword pattern queries that are essentially sub-trajectories regular expression search with spatiotemporal and keyword constraints. For the effective implementation of spatiotemporal keyword pattern queries it is necessary to create efficient index structures that take into account the triple nature (textual and spatiotemporal) of these queries. In this thesis with the use of Neo4j graph database a R-Tree hybrid index was created to address these queries, based on the R-Tree spatial index of the Neo4j spatial library. The index has a hybrid form that combines a spatiotemporal and text index tightly such that both types of information can be used to prune the search space simultaneously during the spatiotemporal keyword query processing. The textual index is using the open-source library Lucene, written in Java that uses inverted indexes for search and retrieval from text collections. After the presentation of the hybrid index appropriate algorithms for the query are analyzed. These algorithms are using different variations of the hybrid R-Tree index in order to address effectively and efficiently the spatiotemporal keyword pattern query.

Περιεχόμενα

1 Εισαγωγή

Η αυξανόμενη χρήση συσκευών με GPS δυνατότητες, όπως κινητά τηλέφωνα ή tablets έχει οδηγήσει στην ανάγκη αποθήκευσης και διαχείρισης μεγάλου όγκου χωροχρονικών δεδομένων (spatiotemporal data) που μπορούν να χρησιμοποιηθούν από κατάλληλες υπηρεσίες και εφαρμογές για την εξαγωγή από αυτά χρήσιμων πληροφοριών και συμπερασμάτων. Οι συγκεκριμένες υπηρεσίες και εφαρμογές χαρακτηρίζονται από την γενική ορολογία Location-based Services (LBS) [1] .

Μία υποκατηγορία των LBS είναι οι διαδικτυακές εφαρμογές και υπηρεσίες που διαχειρίζονται χωροχρονικά δεδομένα, όπως το Foursquare ή το Facebook. Η διαχείριση και ανάλυση των δεδομένων από τις ανωτέρω υπηρεσίες που χαρακτηρίζονται με τον όρο Location-based Social Networking (LBSN) εφαρμογές [1], δεν παρακολουθεί την κίνηση του χρήστη των υπηρεσιών αυτών στο χώρο (καταγραφή χωροχρονικών δεδομένων). Περιέχουν κυρίως μία στατική διάσταση καθώς αναφέρονται σε ποιο γεωγραφικό χώρο βρισκόταν ο χρήστης μία δεδομένη χρονική στιγμή και όχι στο συνολικό ιστορικό κίνησης του χρήστη μέχρι να φτάσει για παράδειγμα στο συγκεκριμένο χώρο.

Η διαχείριση και παρακολούθηση της συνολικής κίνησης ενός αντικειμένου στο χώρο καταγράφοντας παράλληλα πρόσθετα πληροφοριακά στοιχεία για αυτήν, όπως τοποθεσία και μέσο κίνησης ή συγκεκριμένη δραστηριότητα που πραγματοποιείται από τον χρήστη εκείνη τη στιγμή, απαιτεί κατάλληλες δομές και βάσεις δεδομένων. Τα συγκεκριμένα συστήματα βάσεων δεδομένων πρέπει να παρέχουν τη δυνατότητα αποτελεσματικής επεξεργασίας των ανωτέρω δεδομένων και καταγραφής του συνολικού ιστορικού κινήσεων των αντικειμένων προς εξέταση και παρακολούθηση.

Η καταγραφή δεδομένων του ιστορικού κίνησης (χωροχρονικά και λεκτικά δεδομένα) απαιτεί τον ορισμό κατάλληλων αφαιρετικών δομών προκειμένου να είναι δυνατή η αποτελεσματική διαχείριση τους και η πραγματοποίηση κατάλληλων επερωτήσεων σε αυτά για την εξαγωγή πληροφοριών.

Στο δεύτερο κεφάλαιο της εργασίας αναλύονται και παρουσιάζονται οι σημαντικότερες δομές αναπαράστασης των τροχιών ενός αντικειμένου. Οι συγκεκριμένες δομές δίνουν τη δυνατότητα καταγραφής επιπρόσθετων σημασιολογικών πληροφοριών εκτός από τα χωροχρονικά δεδομένα που μπορεί να περιέχει μία τροχιά ενός αντικειμένου. Με τον όρο σημασιολογικές τροχιές [1] ορίζεται η διαδοχική ακολουθία από τμήματα κινήσεων ενός χρήστη που μπορούν να χαρακτηρισθούν ως υποτροχιές κινήσεως ή μη κινήσεως, αναλόγως αν το αντικείμενο τη συγκεκριμένη χρονική στιγμή πραγματοποιούσε κίνηση ή βρισκόταν σε μία συγκεκριμένη γεωγραφική περιοχή χωρίς την πραγματοποίηση της κίνησης.

Μία άλλη αφαιρετική δομή αναπαράστασης των σημασιολογικών τροχιών είναι οι συμβολικές τροχιές [4]. Με τον όρο συμβολική τροχιά ορίζεται μία χρονική ακολουθία από strings που αντιπροσωπεύουν μία συγκεκριμένη χωρική περιοχή που βρισκόταν ένα κινούμενο αντικείμενο σε ένα καθορισμένο χρονικό διάστημα.

Ο ορισμός των ανωτέρω αφαιρετικών δομών για σημασιολογικές τροχιές δίνει τη δυνατότητα πραγματοποίησης επερωτήσεων σε αυτές για την εξαγωγή συμπερασμάτων και πληροφοριών σχετικά με το περιεχόμενο τους. Η δημιουργία κατάλληλων επερωτήσεων σε βάσεις δεδομένων σημασιολογικών τροχιών απαιτεί την εύρεση κατάλληλων αλγοριθμικών τρόπων αποδοτικής εξαγωγής των απαιτούμενων πληροφοριών λαμβάνοντας υπόψη απαραίτητα τον μεγάλο όγκο διακινούμενων πληροφοριών καθώς και την ιδιαίτερη μορφή των δεδομένων αυτών.

Στη συνέχεια του δεύτερου κεφαλαίου παρουσιάζονται οι σημαντικότερες κατηγορίες επερωτήσεων στα ανωτέρω δεδομένα καθώς και οι ευρύτερες κατηγορίες που αφορούν συνδυαστικά queries σε χωρικά δεδομένα που διαθέτουν ταυτόχρονα και λεκτική πληροφορία. Τα ανωτέρω ερωτήματα μπορούν να οριστούν με τον ευρύτερο όρο των spatial keyword queries [9], που υποδεικνύει τον διττό χαρακτήρα των δεδομένων που αυτά περιέχουν καθώς και τη διαφορετική μορφή που αυτά έχουν που απαιτεί κατάλληλο χειρισμό από τα συστήματα βάσεων δεδομένων που τα διαχειρίζονται.

Μία σημαντική κατηγορία mining ερωτημάτων είναι τα spatiotemporal keyword pattern queries. Με την συγκεκριμένη υποκατηγορία ορίζονται ερωτήματα που αφορούν την αναζήτηση σημασιολογικών τροχιών με συγκεκριμένα χωροχρονικά και λεκτικά κριτήρια (patterns) στις επιμέρους σημασιολογικές υποτροχιές (episodes) τους. Ειδικότερα, στο ανωτέρω query ως όρισμα δίνεται μία χρονικά καθορισμένη ακολουθία από υποτροχιές με συγκεκριμένα χωροχρονικά και λεκτικά κριτήρια. Η αναπαράσταση της συγκεκριμένης ακολουθίας υποτροχιών έχει τη μορφή regular expression. Για την αποτελεσματική υλοποίηση των spatiotemporal keyword queries απαιτείται η δημιουργία κατάλληλων ευρετηριακών δομών που να λαμβάνουν υπόψη τη τριπλή φύση των ανωτέρω ερωτημάτων. Με τον όρο τριπλή φύση εννοούμε το γεγονός ότι τα ανωτέρω ερωτήματα έχουν ταυτόχρονα λεκτικά και χωροχρονικά κριτήρια. Στο δεύτερο κεφάλαιο παρουσιάζονται αλγόριθμοι αναζήτησης σημασιολογικών τροχιών με pattern queries, δηλαδή αλγόριθμοι αναζήτησης που έχουν regular expression ορίσματα.

Στη συνέχεια του δευτέρου κεφαλαίου παρουσιάζεται μία αναλυτική περιγραφή των σημαντικότερων μορφών ευρετηρίων για spatial keyword queries καθώς και των κατηγοριών που αυτά ανήκουν. Ένα ευρετήριο μπορεί να αποτελείται από 2 ξεχωριστά υπο-ευρετήρια ένα για την ευρετηρίαση των λεκτικών δεδομένων και ένα για τα χωρικά δεδομένα. Άλλες μορφές ευρετηρίων αντιμετωπίζουν τη διττή φύση

των δεδομένων με τη δημιουργία μίας hybrid (υβριδικής) ευρετηριακής μορφής για την αποτελεσματικότερη αντιμετώπιση των ανωτέρω ερωτημάτων.

Στο τρίτο κεφάλαιο παρουσιάζεται αναλυτικότερα ο ορισμός των ανωτέρω queries και προτείνεται μία συγκεκριμένη βάση δεδομένων με κατάλληλα προσαρμοσμένο ευρετήριο για την αντιμετώπιση των συγκεκριμένων queries. Ειδικότερα, λαμβάνοντας υπόψη το γεγονός ότι ο μεγάλος όγκος των δεδομένων των σημασιολογικών τροχιών μπορεί να αντιμετωπιστεί αποδοτικά από NoSQL συστήματα διαχείρισης βάσεων δεδομένων προτείνεται η χρήση μίας NoSQL βάσης δεδομένων, της Neo4j [26, 27].

Η Neo4j βάση δεδομένων ανήκει στην κατηγορία των Graph databases, δηλαδή χρησιμοποιεί γράφους για την αποθήκευση, διαχείριση και ανάκτηση των δεδομένων. Η συγκεκριμένη κατηγορία βάσεων δεδομένων είναι ιδανική για την διαχείριση δεδομένων που μπορούν να αναπαρασταθούν αποτελεσματικά σε γράφους, όπως η κίνηση ενός αντικειμένου, δηλαδή οι σημασιολογικές τροχιές.

Με τη χρήση της συγκεκριμένης βάσης δεδομένων και μίας κατάλληλης βιβλιοθήκης αυτής, το Neo4j Spatial [28], που δίνει τη δυνατότητα αποθήκευσης και διαχείρισης χωρικών δεδομένων δημιουργήθηκε μία κατάλληλη υποδομή για την πραγματοποίηση spatiotemporal keyword pattern queries. Ειδικότερα, πραγματοποιήθηκε τροποποίηση της δομής της συγκεκριμένης library για τη διαχείριση χωροχρονικών δεδομένων που περιέχουν επιπρόσθετα λεκτική (σημασιολογική) πληροφορία.

Η συγκεκριμένη library για την ευρετηρίαση των χωρικών αντικειμένων χρησιμοποιεί R-Δέντρα. Τα R-Δέντρα [2] είναι παρόμοια με τα B-δέντρα και έχουν σε ένα βαθμό ίδιες ιδιότητες και μεθόδους προσπέλασης και επεξεργασίας πληροφορίας. Τα φύλλα ενός R-Tree περιέχουν εγγραφές δεικτών ευρετηρίου για τα χωρικά αντικείμενα που αντιπροσωπεύουν στη βάση δεδομένων καθώς και ένα ελάχιστο ν-διάστατο πολύγωνο που περιβάλλει το χωρικό αντικείμενο (Minimum Bounding Rectangle - MBR). Αντίστοιχα με τα φύλλα, οι κόμβοι ενός R-Tree περιέχουν ένα ν-διάστατο πολύγωνο που καλύπτει όλα τα MBR των χαμηλότερων κόμβων αλλά και τη διεύθυνση των χαμηλότερων παιδιών-κόμβων που ανήκουν σε αυτούς.

Τα R-Trees είναι μία από τις σημαντικότερες μορφές ευρετηρίασης χωρικών δεδομένων. Ένας μεγάλος αριθμός από παραλλαγές [9] της συγκεκριμένης μορφής υπάρχουν για την εκτέλεση spatial keyword queries, όπως το IF-R*-Tree [10], το KR*-Tree [11] και το IR²-Tree [12]. Για την πραγματοποίηση των spatiotemporal keyword pattern queries χρησιμοποιήθηκε η συγκεκριμένη μορφή ευρετηρίου.

Για την αντιμετώπιση των χωροχρονικών αντικειμένων έγινε επέκταση του ευρετηρίου που χρησιμοποιείται από το Neo4j Spatial προκειμένου να λαμβάνεται

υπόψη και η χρονική διάσταση της πληροφορίας. Ειδικότερα, κάθε αντικείμενο και το αντίστοιχο MBR που το περιβάλλει περιέχει επιπλέον και τη χρονική πληροφορία. Με αυτόν τον τρόπο δημιουργήθηκε ένα R-Tree που κατά τη δημιουργία του λαμβάνεται υπόψη όχι μόνο η χωρική διάσταση των σημασιολογικών τροχιών αλλά και η χρονική στιγμή που τα semantic trajectories έχουν πραγματοποιηθεί.

Με δεδομένο όμως ότι μία σημασιολογική τροχιά εκτός από χωροχρονική πληροφορία έχει και επιπρόσθετη λεκτική πληροφορία, όπως έχει προαναφερθεί νωρίτερα, είναι απαραίτητη και η ευρετηρίαση της συγκεκριμένης πληροφορίας.

Επομένως, η προσθήκη λεκτική πληροφορίας στο index είναι απαραίτητη επειδή ένα spatiotemporal keyword pattern query περιέχει στα κριτήρια αναζήτησης και λεκτικά φίλτρα περιορισμού των δεδομένων. Για την αποτελεσματική αντιμετώπιση επομένως των ανωτέρω ερωτημάτων είναι απαραίτητο το index να έχει υβριδική μορφή, να περιλαμβάνει δηλαδή το R-Tree που θα χρησιμοποιηθεί εκτός από χωροχρονική πληροφορία και λεκτική.

Κατά συνέπεια, η προταθείσα επέκταση του R-Tree περιλαμβάνει την ευρετηρίαση και της λεκτικής πληροφορίας των σημασιολογικών τροχιών εκτός από την ευρετηρίαση της χωροχρονικής πληροφορίας που πραγματοποιείται με τα MBB (Minimum Bounding Box). Ειδικότερα, σε έναν κόμβο του δέντρου περιέχεται ένα MBB που περιβάλλει τα χωροχρονικά αντικείμενα των παιδιών κόμβων του και αντιστοιχεί ένα λεκτικό ευρετήριο (ανεστραμμένη λίστα) που περιέχει την λεκτική πληροφορία όλων των παιδιών του.

Ένα ανεστραμμένο ευρετήριο αντιστοιχεί για κάθε όρο που υπάρχει στα κείμενα μιας συλλογής κειμένων μία λίστα με τα κείμενα της συλλογής που αυτός υπάρχει. Σε αυτήν την περίπτωση η συλλογή κειμένων αντιπροσωπεύει το σύνολο της λεκτικής πληροφορίας των σημασιολογικών τροχιών.

Η δημιουργία των λεκτικών ευρετηρίων γίνεται μέσω της library Lucene [32]. Η Lucene είναι μία open-source βιβλιοθήκη υλοποιημένη στη γλώσσα προγραμματισμού Java. Παρέχει κατάλληλα εργαλεία για την δημιουργία μίας μηχανής αναζήτησης που μπορεί να ενσωματωθεί σε προγράμματα και υπηρεσίες προσαρμοσμένη κάθε φορά στις ιδιαίτερες απαιτήσεις που μπορεί να υπάρχουν ανά περίπτωση χρήσης. Η λειτουργία της συγκεκριμένης βιβλιοθήκης βασίζεται στη δημιουργία ανεστραμμένων ευρετηρίων για την αναζήτηση και ανάκτηση πληροφορίας σε μία συλλογή από κείμενα.

Στη συνέχεια του τρίτου κεφαλαίου παρουσιάζεται ο τρόπος αποθήκευσης και διαχείρισης των δεδομένων των σημασιολογικών τροχιών σε γράφους σύμφωνα με τις απαιτήσεις της βάσης Neo4j και της library Neo4j Spatial, καθώς και περιγραφή του τρόπου λειτουργίας της Lucene. Η περιγραφή του συγκεκριμένου τρόπου

ευρετηρίασης και αναζήτησης της Lucene περιλαμβάνει και την απαραίτητη θεωρητική θεμελίωση μέσα από την παρουσίαση των βασικών αρχών της ανάκτησης πληροφορίας (information retrieval).

Επιπρόσθετα, παρουσιάζονται αναλυτικά τα χαρακτηριστικά, οι λειτουργίες και οι κλάσεις του υβριδικού ευρετηρίου καθώς και η αποθήκευση της δομής του στη βάση δεδομένων Neo4j.

Στο τέταρτο κεφάλαιο της εργασίας, μετά από την παρουσίαση του τρόπου λειτουργίας και διαχείρισης των χωροχρονικών δεδομένων της βάσης δεδομένων Neo4j και της Neo4j Spatial library παρουσιάζεται κατάλληλες αλγοριθμικές επιλύσεις του ερωτήματος. Οι συγκεκριμένες αλγοριθμικές επιλύσεις του spatiotemporal keyword pattern query εκμεταλλεύονται διαφορετικές παραλλαγές του υβριδικού index που έχει δημιουργηθεί προκειμένου να αντιμετωπιστεί αποτελεσματικά και αποδοτικά η επίλυση του spatiotemporal keyword pattern query.

Τέλος, στο πέμπτο κεφάλαιο της εργασίας παρουσιάζεται γραφική διεπαφή πραγματοποίησης των ερωτημάτων και στο έκτο κεφάλαιο της εργασίας παρουσιάζονται σύντομα συμπεράσματα από τη χρησιμοποίηση της Neo4j Graph database στη διαχείριση σημασιολογικών τροχιών και την πραγματοποίηση pattern queries σε αυτήν.

2 Σημασιολογικές τροχιές και spatial keyword queries

2.1 Semantic Trajectories

Η καταγραφή και διαχείριση των χωροχρονικών δεδομένων ενός αντικειμένου έχει ως αποτέλεσμα την αποθήκευση πληροφοριών για την τροχιά (trajectory) [1] ενός αντικειμένου στο χώρο. Με το γενικό όρο trajectory ορίζεται η κίνηση του αντικειμένου στο χώρο και η σχετική πληροφορία που καταγράφεται για αυτήν την κίνηση.

Η πρωτογενής και ανεπεξέργαστη τροχιά ενός αντικειμένου στο χώρο (raw trajectory) περιέχει τις χωροχρονικές συντεταγμένες κίνησης του αντικειμένου, όπως αυτές έχουν καταγραφεί από μία συσκευή GPS. Ένα raw trajectory τ [1] κατά συνέπεια ορίζεται ως (o-id, traj-id, T), όπου ως o-id είναι ένα μοναδικό στοιχείο αναγνώρισης του αντικειμένου (όπως ένας μοναδικός αριθμός id), traj-id είναι ένα μοναδικό στοιχείο αναγνώρισης της τροχιάς του αντικειμένου και T είναι μία 3D πολυγωνική γραμμή που αναπαριστά την τροχιά του αντικειμένου στο χώρο και αποτελείται από μία ακολουθία διαδοχικών χωρικών συντεταγμένων (έστω x και y) και την αντίστοιχη χρονική στιγμή t (timestamp).

Ένα raw sub-trajectory τ’ (υποτροχιά) αντίστοιχα ορίζεται ως (o-id, traj-id, subtraj-id, T’), όπου (o-id, subtraj-id) είναι ένα μοναδικό στοιχείο αναγνώρισης της συγκεκριμένης υποτροχιάς του αντικειμένου και T’ είναι μία 3D πολυγωνική γραμμή που αναπαριστά την υποτροχιά του αντικειμένου στο χώρο και είναι τμήμα της πολυγωνικής γραμμής T, μεταξύ δύο χρονικών στιγμών.

Ο χαρακτηρισμός ενός sub-trajectory ως stop ή move προσδιορίζεται από συγκεκριμένες χωροχρονικές ιδιότητες του αντικειμένου. Ειδικότερα, ένα sub-trajectory μπορεί να χαρακτηριστεί ως stop αν ικανοποιούνται συγκεκριμένοι χωρικοί περιορισμοί και αντίστοιχα ως move αν ικανοποιούνται συγκεκριμένοι χρονικοί περιορισμοί. Οι ανωτέρω περιορισμοί προσδιορίζονται ανάλογα με το είδος της υπηρεσίας ή της εφαρμογής που αποθηκεύει και διαχειρίζεται την πληροφορία των αντικειμένων.

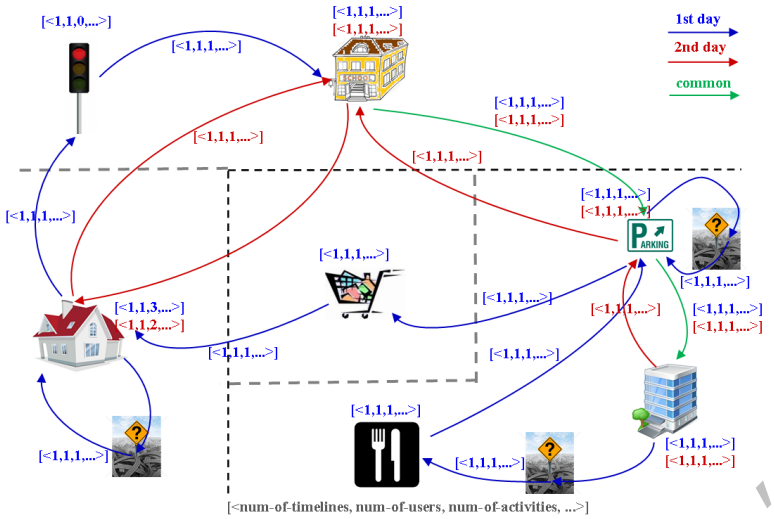
Ένα επεισόδιο (episode ή lifestep) αντιστοιχεί σε ένα συγκεκριμένο raw sub-trajectory και περιλαμβάνει επιπρόσθετα εκτός από την χωροχρονική πληροφορία που υπάρχει σε αυτό και σημασιολογική πληροφορία σχετική με την κίνηση του χρήστη, όπως λεκτική πληροφορία (κατάλληλα tags που περιέχουν πληροφορίες σχετικά με το συγκεκριμένο υποτμήμα της τροχιάς του χρήστη). Ειδικότερα, ένα episode ορίζεται ως (defineTag, MBB, episodeTag, activityTag, T-link), όπου

defineTag είναι ο χαρακτηρισμός του ως stop/move, MBB είναι το ελάχιστο τετράγωνο (Minimum Bounding Box) που περικλείει την συγκεκριμένη υποτροχιά του αντικειμένου μαζί με τις χρονικές στιγμές αρχής και τέλους που βρισκόταν το αντικείμενο σε αυτό, episodeTag και activityTag περιέχουν τη σημασιολογική (semantic) πληροφορία σχετική με την κίνηση του αντικειμένου και T-link προσδιορίζει την πολυγωνική γραμμή τ' που αφορά το συγκεκριμένο episode.

Μία σημασιολογική τροχιά (semantic trajectory) ενός κινούμενου αντικειμένου ορίζεται ως (o-id, semtraj-id, T_{sem}), όπου o-id και semtraj-id είναι τα στοιχεία αναγνώρισης του αντικειμένου και της τροχιάς και T_{sem} είναι η διαδοχική χρονικά ακολουθία από episodes που ανήκουν στο ίδιο trajectory τ.

Η διαχείριση και επεξεργασία των semantic trajectories μπορεί να πραγματοποιηθεί αποδοτικά με την αναπαράσταση και αποθήκευσή τους σε κατάλληλες βάσεις δεδομένων (Semantic Mobility Databases – SMD).

Μία δυναμική αναπαράσταση των περιεχομένων σε μία SMD, μπορεί να πραγματοποιηθεί με τη χρησιμοποίηση του σημασιολογικού δικτύου κινητικότητας (Semantic Mobility Network – SMN) [1]. Ένα semantic mobility network αποτελεί μία δυναμική αναπαράσταση ενός συνόλου από semantic trajectories που βρίσκονται σε μία SMD, χρησιμοποιώντας τη δομή των γράφων. Συγκεκριμένα, το SMN αποτελείται από ένα γράφο όπου το σύνολο των κορυφών αναπαριστούν το σύνολο των «stop» lifesteps των semantic trajectories της SMD και το σύνολο των ακμών αναπαριστούν το σύνολο των «move» lifesteps των semantic trajectories της SMD. Κάθε ακμή ή κορυφή περιέχει επιπρόσθετα ένα σύνολο από αλφαριθμητικές τιμές που χρησιμοποιούνται για την αποθήκευση σε αυτές σημασιολογικών πληροφοριών των lifesteps όλων των semantic trajectories που υπάρχουν στη SMD.



Σχήμα 1: Semantic Mobility Network [1]

Προκειμένου να είναι αποδοτική η ανάκτηση δεδομένων σημασιολογικών τροχιών από τη βάση και η πραγματοποίηση επερωτήσεων σε αυτά τα δεδομένα, είναι απαραίτητη η δημιουργία ενός κατάλληλου ευρετηρίου (index). Το SemTB-tree [1] βασίζεται στη μορφολογία του R-Tree [2] και του TB-Tree [3] και δίνει τη δυνατότητα ευρετηρίασης χωροχρονικών και λεκτικών δεδομένων στο ίδιο index.

Τα R-δέντρα [2] αποτελούν μία δομή χωρικού ευρετηρίου που αποτελεί γενίκευση των B-δέντρων, κατάλληλη για χωρικά αντικείμενα. Η δημιουργία των R-δέντρων πραγματοποιείται με κατάλληλες αλγοριθμικές τεχνικές δημιουργίας ευρετηρίων για αποδοτική αποθήκευση, επεξεργασία και αναζήτηση χωρικών δεδομένων σε ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων (DBMS) που δίνει τη δυνατότητα αποθήκευσης χωρικής πληροφορίας (Spatial DBMS).

Τα R-δέντρα είναι παρόμοια με τα B-δέντρα και έχουν σε ένα βαθμό ίδιες ιδιότητες και μεθόδους προσπέλασης και επεξεργασίας πληροφορίας. Ένα R-δέντρο είναι ισοσταθμισμένο και περιέχει κόμβους και φύλλα. Κάθε κόμβος περιέχει ένα μέγιστο (M) και ελάχιστο αριθμό στοιχείων (m), εξαιρουμένης της ρίζας. Όλα τα φύλλα του δέντρου βρίσκονται στο ίδιο επίπεδο. Τα φύλλα περιέχουν εγγραφές δεικτών ευρετηρίου για το πλήρες χωρικό αντικείμενο στη βάση και ένα ελάχιστο ν-διάστατο πολύγωνο που περιβάλλει το αντικείμενο (Minimum Bounding Rectangle - MBR). Οι κόμβοι του δέντρου περιέχουν αντίστοιχα ένα ν-διάστατο πολύγωνο που καλύπτει όλα τα MBR των χαμηλότερων κόμβων αλλά και τη διεύθυνση των χαμηλότερων παιδιών-κόμβων από αυτόν. Δεν υπάρχει περιορισμός όσον αφορά την αλληλοεπικάλυψη των διαστημάτων που καλύπτονται από τα MBR, δηλαδή μπορεί

ένα χωρικό αντικείμενο να επεκτείνεται σε παραπάνω από μία περιοχές που ορίζονται από κόμβους.

Για την αναζήτηση, ενημέρωση, εισαγωγή και διαγραφή στοιχείων από R-δέντρα υπάρχουν κατάλληλοι αλγόριθμοι. Όσον αφορά την αναζήτηση ακολουθείται παρόμοια λογική με τα B-δέντρα, υπάρχει όμως πιθανότητα λόγω της αλληλοεπικάλυψης που υπάρχει στα v-διάστατα πολύγωνα που περιβάλλουν τα χωρικά αντικείμενα να χρειαστεί η επίσκεψη περισσότερων του ενός μονοπατιών στο δέντρο. Η αναζήτηση ξεκινάει από τη ρίζα του δέντρου και προχωράει σε μονοπάτι στο δέντρο μόνο αν ο αντίστοιχος κόμβος υπό εξέταση ικανοποιεί τα χωρικά κριτήρια της αναζήτησης.

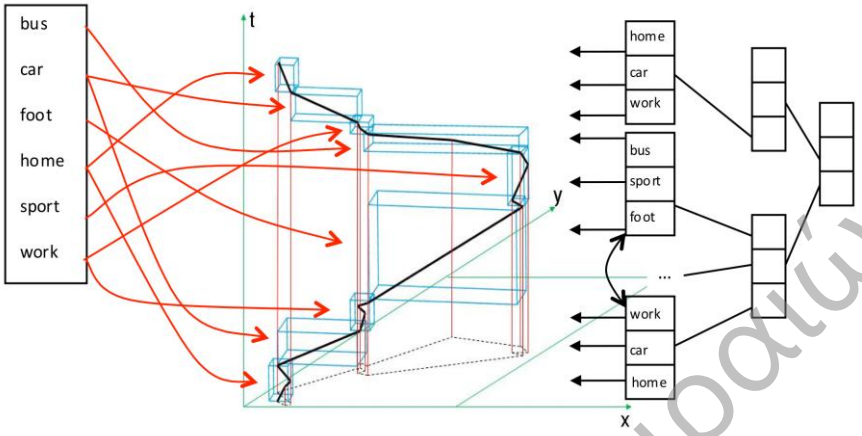
Στην εισαγωγή στοιχείων, ξεκινώντας από τη ρίζα και πηγαίνοντας προς τα φύλλα, διαλέγουμε κάθε φορά τον κόμβο που το v-διάστατο πολύγωνο του χρειάζεται την ελάχιστη αύξηση. Σε περίπτωση που υπάρχουν κόμβοι που έχουν στοιχεία περισσότερα από M, λόγω του αλγόριθμου εισαγωγής γίνεται διαχωρισμός τους που μεταδίδεται στη συνέχεια και σε υψηλότερα επίπεδα του δέντρου με αντίστοιχους διαχωρισμούς αλλάζοντας την αρχική διαμόρφωση του δέντρου.

Στον αλγόριθμο της διαγραφής σε περίπτωση που ένας κόμβος έχει λιγότερα από m στοιχεία, γίνεται ξανά εισαγωγή όλων των στοιχείων αυτού του κόμβου στο δέντρο. Ο λόγος που πραγματοποιείται ξανά εισαγωγή των στοιχείων είναι επειδή οι εισαγωγές στοιχείων σταδιακά αλλοιώνουν την δομή του δέντρου με αποτέλεσμα να υπάρχει μειωμένη απόδοση του στην διαχείριση του. Σε περίπτωση αλλαγής των χωρικών στοιχείων ενός αντικειμένου, ακολουθείται η ίδια φιλοσοφία διαγραφής και επαναεισαγωγής του στο δέντρο. Όσον αφορά την εισαγωγή μιας καινούριας εγγραφής σε κόμβο που περιέχει ήδη M στοιχεία, είναι απαραίτητη η διαίρεσή του σε δύο ξεχωριστούς κόμβους. Ο διαχωρισμός πρέπει να γίνει με τέτοιο τρόπο έτσι ώστε να μην είναι πιθανό να χρειαστεί επίσκεψη και των δύο κόμβων σε μία μελλοντική αναζήτηση. Το κριτήριο του διαχωρισμού είναι η ελάχιστη δυνατή περιοχή επικάλυψης των καινούριων v-διάστατων πολύγωνων που θα προκύψουν.

Τα R-δέντρα γενικότερα είναι αρκετά αποτελεσματικά όσον αφορά την αναζήτηση στον περιορισμό της χωρικής περιοχής αναζήτησης, δηλαδή στον περιορισμό όσον αφορά τα υποδέντρα τα μονοπάτια των οποίων θα αναζητήσουν.

Όσον αφορά συγκεκριμένα το SemTB-tree, όλα τα lifesteps των semantic trajectories με τα αντίστοιχα MBB χρησιμοποιούνται ως βάση στα φύλλα του δέντρου για τη δημιουργία του index, ενώ σε δεύτερη φάση δημιουργείται αντίστοιχα και ένα ανεστραμμένο αρχείο για την αποθήκευση της σημασιολογικής πληροφορίας. Οι

λεκτικές πληροφορίες είναι αντίστοιχα συνδεμένες με τα φύλλα του δέντρου ανάλογα με το lifestep στο οποίο αντιστοιχούν, δημιουργώντας έτσι ένα υβριδικό index. Η μορφολογία του δέντρου επιτρέπει την πραγματοποίηση επερωτήσεων στη βάση που έχουν ταυτόχρονα χωροχρονικά και λεκτικά κριτήρια.



Σχήμα 2: SemTB-tree [1]

Η αναζήτηση ενός συγκεκριμένου χωροχρονικού-λεκτικού μοτίβου (Spatio-Temporal-Textual Pattern query – ST²P) σε μία SMD σημαίνει ότι αναζητείται μία συγκεκριμένη σημασιολογική τροχιά που έχει μία ακολουθία από lifesteps που έχουν κάποια δεδομένα χωροχρονικά και λεκτικά κριτήρια το κάθε ένα από αυτά. Η ανωτέρω ακολουθία δεν σημαίνει απαραίτητα ότι κάθε ένα lifestep σε αυτή είναι χρονικά αμέσως μετά το επόμενο που βρίσκεται στην ακολουθία αλλά μπορεί στο ζητούμενο semantic trajectory να υπάρχει ενδιάμεσα από δύο lifesteps ένας ακαθόριστος αριθμός από άλλα lifesteps που δεν έχουν από την επερώτηση συγκεκριμένα κριτήρια.

Για παράδειγμα, ένα ST²P query μπορεί να έχει ως αντικείμενο αναζήτησης ένα semantic trajectory που να περιέχει μεταξύ άλλων 3 lifesteps που να έχουν την λεκτική πληροφορία home, school και work και να έχουν συγκεκριμένους χωροχρονικούς περιορισμούς. Δηλαδή μπορεί να αναζητούνται σημασιολογικές τροχιές που κάποια χρονική περίοδο έχουν ένα lifestep «home», μετά από μηδέν ή περισσότερα lifesteps ένα lifestep «work» και μετά από μηδέν ή περισσότερα lifesteps ένα lifestep «home», χωρίς απαραίτητα αυτό να είναι το τελευταίο lifestep.

Για την αλγοριθμική προσέγγιση του ST²P query θεωρούμε αρχικά έναν πίνακα L που περιέχει τα lifesteps που θα χρησιμοποιηθούν ως κριτήρια αναζήτησης για το query, έναν πίνακα W που θα περιέχει την πληροφορία αν τα lifesteps του πίνακα L

είναι συνεχόμενα χρονικά μεταξύ τους (χαρακτήρας «>») ή δεν είναι (χαρακτήρας «*»).

Algorithm *ST²P*

Input: an array of lifesteps *L*, an array of wildchars *W*, the *root* of the Sem-TBtree
Output: the IDs of the timelines that conform to pattern (*L*,*W*)

```
1. begin
2.    prev_sol, tags_sol = ∅
3.    for each ls_i in L
4.          tags_sol= pattern_tags(L(i), W(i), prev_sol, root)
5.          if tags_sol= ∅ then
6.          prev_sol = tags_sol
7.          break
8.          end if
9.          prev_sol = pattern_mbbs(L(i), W(i), prev_sol, tags_sol, root)
10.   end for
11.   return timelineIDs from prev_sol
12. end
```

Σχήμα 3: The ST²P query processing algorithm [1]

Ο αλγόριθμος, όπως φαίνεται στο Σχήμα 3, με μία επαναληπτική λούπα αρχικά εξετάζει σταδιακά τα lifesteps του πίνακα L. Η μεταβλητή tags_sol χρησιμοποιείται για την εξέταση υποψήφιων απαντήσεων του query όταν εξετάζονται λεκτικοί περιορισμοί, ενώ η μεταβλητή prev_sol χρησιμοποιείται για την αποθήκευση σε αυτήν των εκάστοτε απαντήσεων που έχουν βρεθεί κατά την εκτέλεση του αλγόριθμου. Με την επαναληπτική λούπα για κάθε lifestep του πίνακα L αρχικά εξετάζεται η ικανοποίηση των λεκτικών περιορισμών με την χρησιμοποίηση της μεθόδου pattern_tags που κάνει χρήση του SemTB-tree. Στη συνέχεια, εάν ικανοποιούνται οι λεκτικοί περιορισμοί και για τα συγκεκριμένα lifesteps που έχουν βρεθεί από την προηγούμενη μέθοδο υπάρχουν επιπρόσθετα χωροχρονικοί περιορισμοί, γίνεται χρήση της μεθόδου pattern_mbbs. Εάν δεν ικανοποιούνται οι λεκτικοί περιορισμοί για την συγκεκριμένη επανάληψη τότε παύει η συγκεκριμένη επαναληπτική εκτέλεση με την εντολή break.

Η μέθοδος pattern_tags ξεκινάει την αναζήτηση στο SemTB-tree από το ανεστραμμένο αρχείο του index (αριστερή πλευρά στο Σχήμα 2), ενώ η μέθοδος pattern_mbbs ξεκινάει την αναζήτηση στο index από το TB-tree (δεξιά πλευρά στο Σχήμα 2).

2.2 Symbolic Trajectories

Μία διαφορετική αφαίρεση των στοιχείων που περιέχονται σε raw trajectories είναι οι συμβολικές τροχιές (symbolic trajectories) [4]. Τα symbolic trajectories στην πιο απλή τους μορφή ορίζονται ως μία συνάρτηση που απεικονίζει χρόνο σε string labels. Ειδικότερα, η συνάρτηση αυτή μπορεί να αναπαρασταθεί ως η ακολουθία < $(i_1, l_1, \dots, (i_n, l_n)$ >, όπου το i_j αντιστοιχεί σε ένα χρονικό διάστημα και το l_j αντιστοιχεί σε ένα string label που αποτελεί αφαίρεση μίας συγκεκριμένης γεωμετρικής οντότητας. Τα χρονικά διαστήματα στην συγκεκριμένη ακολουθία δεν επικαλύπτονται μεταξύ τους και τοποθετούνται σε αυτήν με χρονική διαδοχή, ενώ τα string labels μπορούν να αντιστοιχούν σε συγκεκριμένες χωρικές οντότητες που αναπαριστούν γεωμετρικά την εκάστοτε υποτροχιά ενός trajectory και βρίσκονται αποθηκευμένες στο data framework που χρησιμοποιείται. Με αυτόν τον τρόπο τα symbolic trajectories αναπαριστούν αφαιρετικά μία τροχιά ενός κινούμενου αντικειμένου, έχοντας παράλληλα και σημασιολογικά στοιχεία για αυτήν. Αποτελούν με αυτόν τον τρόπο μία διαφορετική αφαιρετική επέκταση των raw trajectories.

```
< ([8:30 - 8:45], walk), ([8:45 - 9:13], train), ([9:13 - 9:19], walk) >
```

Σχήμα 4: Παράδειγμα ενός symbolic trajectory [4]

Η αποθήκευση και διαχείριση της πληροφορίας των symbolic trajectories σε βάση δεδομένων πραγματοποιείται μέσω data framework [5] που περιέχει κατάλληλο data type, το moving(label) ή mlabel. Το συγκεκριμένο framework περιέχει επιπρόσθετα κατάλληλα data types για την διαχείριση τροχιών κινούμενων αντικειμένων: moving(point) ή mpoint για την αναπαράσταση γεωμετρικών τροχιών αντικειμένων και moving(real) ή mreal για την αποθήκευση αριθμητικών μεγεθών, όπως για παράδειγμα η ταχύτητα του κινούμενου αντικειμένου. Η αντιστοίχιση των labels με συγκεκριμένες χωροχρονικές οντότητες μπορεί να πραγματοποιηθεί με κατάλληλους κωδικούς αριθμούς που υποδηλώνουν αναφορά σε κάποιο geometry repository. Για παράδειγμα το label μπορεί να έχει την τιμή «(cinema,114)». Η τιμή «114» αποτελεί reference σε κάποιο geometry repository του framework. Η σύνδεση των labels με γεωμετρικές οντότητες μπορεί να χρησιμοποιηθεί για την δημιουργία ερωτημάτων στο πλαίσιο του data framework που εκμεταλλεύονται ταυτόχρονα την δομή των symbolic trajectories και γεωμετρικές ιδιότητες αυτών. Η διαφορετική αντιμετώπιση και διαχείριση της χωρικής διάστασης είναι η σημαντικότερη εννοιολογική διαφορά σε σύγκριση με τη δομή που παρουσιάστηκε στην προηγούμενη ενότητα.

Τα symbolic trajectories κατά συνέπεια μπορούν να χρησιμοποιηθούν για την πραγματοποίηση pattern matching queries. Ειδικότερα, μπορεί να πραγματοποιηθεί αναζήτηση για trajectories που έχουν ένα συγκεκριμένο pattern, τόσο όσον αφορά χωροχρονικά κριτήρια, όσο και λεκτικά κριτήρια. Για παράδειγμα μπορεί να αναζητηθούν trajectories που έχουν την μορφή του Σχήματος 5.

```
* (monday taxi) X (_ bus) * // duration(X.time) > 20 * minute
```

Σχήμα 5: Symbolic trajectory pattern [4]

Στο συγκεκριμένο παράδειγμα του Σχήματος 5 αναζητούνται trajectories που περιέχουν την μετάβαση από ταξί σε λεωφορείο. Η μετάβαση αυτή πρέπει να έχει πραγματοποιηθεί ημέρα Δευτέρα και η χρονική διάρκεια χρήσης του λεωφορείου να είναι μεγαλύτερη από 20 λεπτά. Στη συγκεκριμένη υποτροχιά που περιλαμβάνει μετάβαση με λεωφορείο ορίζεται ότι δεν υπάρχει κάποια συγκεκριμένη χρονική ιδιότητα (όπως ημέρα πραγματοποίησης), χρησιμοποιώντας το χαρακτήρα «_».

Η χρησιμοποίηση μεταβλητών (στο παράδειγμα συμβολίζεται με το γράμμα X), δίνει τη δυνατότητα πρόσβασης σε περισσότερα δεδομένα που βρίσκονται στα subtrajectories της τροχιάς του αντικειμένου. Σε αυτήν την περίπτωση η μεταβλητή χρησιμοποιείται για την πρόσβαση στην χρονική διάρκεια μίας συγκεκριμένης υποτροχιάς που περιλαμβάνει την μετακίνηση με λεωφορείο. Οι μεταβλητές ορίζονται με κεφαλαία γράμματα και δίνουν τη δυνατότητα πρόσβασης σε δεδομένα της υποτροχιάς, όπως label, time, start και end. Τοποθετούνται πάντα μπροστά από το ατομικό pattern που αυτές συσχετίζονται, δηλαδή στο συγκεκριμένο παράδειγμα μπροστά από το «(_ bus)».

Χρησιμοποιώντας ως βασικό παράδειγμα για τη συνέχεια το symbolic trajectory του Σχήματος 6 (που περιλαμβάνει 5 υποτροχιές), μπορούμε να ορίσουμε το pattern « (_ “Queen Anne St”) T * A (_ “Queen Anne St”) » που αυτό ανήκει.

```
( ( (2013-01-17-9:02:30 2013-01-17-9:05:51 T F) "Queen Anne St")
  ( (2013-01-17-9:05:51 2013-01-17-9:10:16 T F) "Wimpole St")
  ( (2013-01-17-9:10:16 2013-01-17-9:13:48 T F) "Welbeck Way")
  ( (2013-01-17-9:13:48 2013-01-17-9:18:44 T F) "Welbeck St")
  ( (2013-01-17-9:18:44 2013-01-17-9:20:10 T F) "Queen Anne St") )
```

Σχήμα 6: Παράδειγμα ενός symbolic trajectory [4]

Το συγκεκριμένο pattern περιλαμβάνει τροχιές που ξεκινούν από την οδό «Queen Anne St» και καταλήγουν σε αυτήν ξανά. Ο χαρακτήρας «*» δηλαδή περιλαμβάνει 0 ή περισσότερες υποτροχιές σε ένα trajectory. Προκειμένου μία τροχιά να ανήκει σε ένα συγκεκριμένο pattern πρέπει να γίνει μία κατάλληλη ομαδοποίηση των υποτροχιών της έτσι ώστε να υπάρχει αντιστοιχία μεταξύ των ατομικών pattern units που υπάρχουν στο συνολικό pattern. Η αντιστοίχιση αυτή ονομάζεται binding.

Στο ανωτέρω παράδειγμα, με τον χαρακτήρα «*» (ατομικό pattern unit) και τη μεταβλητή T που ορίζονται μηδέν η περισσότερες υποτροχιές στο pattern σε συγκεκριμένη θέση, αντιστοιχούν η δεύτερη, τρίτη και τέταρτη υποτροχιά του παραδείγματος. Χρησιμοποιώντας επομένως τη μεταβλητή T μπορούμε να αποκτήσουμε πρόσβαση σε πληροφορίες για αυτό το σύνολο των υποτροχιών για τη δημιουργία λογικών συνθηκών. Για παράδειγμα έστω η τιμή «T.labels», που θα επιστρέψει το σύνολο με τα labels τους ή «T.card» που θα επιστρέψει τον αριθμό των δρόμων που αυτές περιλαμβάνουν και μπορεί να χρησιμοποιηθεί σε κάποια λογική συνθήκη. Με το ατομικό pattern unit «A (_ “Queen Anne St”)» θα αντιστοιχίσουμε στη συνέχεια την τελευταία (πέμπτη) υποτροχιά του παραδείγματος. Το συγκεκριμένο pattern unit περιλαμβάνει υποτροχιές που ανεξαρτήτως χρονικού διαστήματος περιέχουν την οδό «Queen Anne St». Κατά συνέπεια, το binding για το συγκεκριμένο παράδειγμα θα αντιστοιχεί στο binding του Σχήματος 7.

```
{ (T, ( (2013-01-17-9:05:51 2013-01-17-9:10:16 T F) "Wimpole St")
      ( (2013-01-17-9:10:16 2013-01-17-9:13:48 T F) "Welbeck Way")
      ( (2013-01-17-9:13:48 2013-01-17-9:18:44 T F) "Welbeck St") ),
  (A, ( (2013-01-17-9:18:44 2013-01-17-9:20:10 T F) "Queen Anne St") ) }
```

Σχήμα 7: Symbolic Trajectory Binding [4]

Χρησιμοποιώντας τα symbolic trajectory patterns queries είναι δυνατή η μετατροπή των τροχιών που επιστρέφουν ως αποτελέσματα σε μία διαφορετική μορφή που εμείς επιθυμούμε, που ονομάζεται rewriting. Κάνοντας χρήση του παραδείγματος που αναφέρθηκε προηγουμένως, έστω ότι επιθυμούμε να κατηγοριοποιήσουμε τροχιές που ξεκινάνε από την οδό «Queen Anne St» και καταλήγουν σε αυτήν ξανά και έχουν χρονική διάρκεια μικρότερη των 20 λεπτών ως «short walk», παρουσιάζοντας τα αποτελέσματα του συγκεκριμένου pattern ως μία μόνο οντότητα (unit).

```
D(_ "Queen Anne St")  *  A(_ "Queen Anne St")
        // (A.end - D.start) < (20 * minute)
=> X
        // X.label := "short walk", X.start := D.start, X.end := A.end
```

Σχήμα 8: Symbolic Trajectory Rewriting [4]

Αυτό μπορεί να πραγματοποιηθεί με τη χρησιμοποίηση αρχικά του query pattern και στη συνέχεια τα αποτελέσματα που επιστρέφονται από αυτό να γίνουν rewriting. Με τη δήλωση μίας καινούριας μεταβλητής (έστω «X») και ορίζοντας label σε αυτήν την τιμή «short walk», έναρξη την χρονική στιγμή έναρξης της πρώτης υποτροχιάς κάθε αποτελέσματος και λήξη αντίστοιχα μπορούμε να αλλάξουμε μορφή στα αποτελέσματα που επιστρέφονται ανάλογα με τις ανάγκες παρουσίασης και επεξεργασίας των αποτελεσμάτων. Αντίστοιχα για trajectories με χρονική διάρκεια μεγαλύτερη των 20 λεπτών θα μπορούσαν να έχουν διαφορετικό χαρακτηρισμό. Με τον τρόπο αυτό είναι δυνατή η εμφάνιση και η μετέπειτα επεξεργασία μόνο του μέρους της σημασιολογικής πληροφορίας που επιθυμούμε από trajectories που ικανοποιούν συγκεκριμένα κριτήρια αναζήτησης.

Η υλοποίηση των pattern queries σε symbolic trajectories πραγματοποιήθηκε με την επέκταση του Secondo DBMS [6]. Δημιουργήθηκαν κατάλληλες δομές δεδομένων (data structures) που αναπαριστούν αντίστοιχες κλάσεις αντικειμένων για την αποθήκευση και διαχείριση των παραμέτρων των query patterns. Για την περιγραφή των δομών δεδομένων που δημιουργήθηκαν θα χρησιμοποιηθούν τα 2 patterns του Σχήματος 9, έστω P_0 και P_1.

```
X * Y [(thursday, morning "Queen Anne St") | (_ "Welbeck St")]+ Z [()]?
     // (Y.end - X.start) < 20 * minute

X * Y [(thursday, morning "Queen Anne St") | (_ "Welbeck St")]+ Z [()]?
     // (Y.end - X.start) < 20 * minute
=> A Y
     // A.time := X.time, A.label := "start of trip"
```

Σχήμα 9: Symbolic trajectories patterns [4]

Το pattern query P_0 αναφέρεται σε όλες τις τροχιές που περνάνε τουλάχιστον μία φορά από την οδό «Queen Anne St» ημέρα «thursday» σε πρωινές ώρες ή περνάνε από την οδό «Welbeck St» ανεξαρτήτως χρονικού περιορισμού, είτε ακριβώς πριν την τελευταία υποτροχιά του trajectory ή στο τέλος του. Επιπρόσθετα, με τη χρήση

των μεταβλητών X και Y, δημιουργείται χρονικός περιορισμός μεταξύ των υποτροχιών που θα συσχετιστούν με τις συγκεκριμένες μεταβλητές.

Το pattern query P_1 αναφέρεται επίσης σε όλες τις τροχιές του pattern query P_0 με τη διαφορά ότι περιέχει επιπλέον έναν κανόνα rewriting για τον μετασχηματισμό των αποτελεσμάτων σε τροχιά-αποτέλεσμα με τη χρησιμοποίηση δύο μεταβλητών A και Y. Η μεταβλητή A περιέχει την χρονική στιγμή ακριβώς πριν το κινούμενο αντικείμενο περάσει από την οδό «Queen Anne St» ή «Welbeck St» καθώς και label με την τιμή «start of trip».

Ως παράδειγμα ελέγχου των συγκεκριμένων patterns θα χρησιμοποιηθεί το παράδειγμα του Σχήματος 6. Με το συμβολισμό M θα αναπαρασταθεί το σύνολο των υποτροχιών (unit items m_i) του παραδείγματος του σχήματος 6. Το πλήθος τους $|M_0|$ αντιστοιχεί σε 5. Αντίστοιχα με το συμβολισμό P θα αναπαρασταθεί το σύνολο των ατομικών στοιχείων του pattern query (atomic pattern elements p_i) του παραδείγματος του Σχήματος 6. Σύμφωνα με το Σχήμα 9 τα atomic pattern elements θα αριθμηθούν ακολουθώντας τη μορφή «0(1|2)+3?». Το πλήθος τους $|P_0|$ επομένως αντιστοιχεί σε 4

Για την αποθήκευση των πληροφοριών ενός atomic pattern element χρησιμοποιείται μία μεταβλητή τύπου string, ένα string set για τα χρονικά διαστήματα (time intervals), ένα string set για τα labels και ένα string (wildcard) για την περίπτωση ύπαρξης του «*» (kleene star), «+» ή κανένα από τα δύο. Αντίστοιχα αποθηκεύονται οι τιμές «star», «plus» ή «no». Για το παράδειγμα P_0 που αναφέρθηκε παραπάνω και το atomic pattern element p_1, αποθηκεύεται η μεταβλητή Y, δύο strings με τις τιμές «thursday» και «morning» για την χρονική πληροφορία, ένα set με labels με την τιμή «Queen Anne St» και η τιμή wildcard «no». Το atomic pattern element p_2 θα έχει επίσης την μεταβλητή Y, αλλά στο set με labels θα έχει την τιμή «Welbeck St» και η τιμή wildcard θα έχει την τιμή «plus».

Για την αποθήκευση των πληροφοριών μιας συνθήκης (condition object) χρησιμοποιείται μία μεταβλητή τύπου string για το συνολικό κείμενο (input) της συνθήκης και ένας vector για την αποθήκευση των μεταβλητών της συνθήκης (string) και τον αντίστοιχο αριθμητικό κωδικό που αυτές αντιστοιχούν (integer). Στην περίπτωση του παραδείγματος P_0 για την συνθήκη που αυτό περιέχει αποθηκεύεται στο input η τιμή « (Y.end – X.start) < 20 * minute » και στον vector οι τιμές «Y» και «X» των μεταβλητών και οι τιμές «end» και «start», με αριθμητικούς κωδικούς «3» και «2» αντίστοιχα. Οι τιμές «3» και «2» είναι προκαθορισμένες τιμές για τα συγκεκριμένα attributes.

Οι συνθήκες διαχωρίζονται σε δύο κατηγορίες, απλές και σύνθετες. Εάν η συνθήκη περιέχει μόνο μία μεταβλητή και η συγκεκριμένη μεταβλητή αναφέρεται σε ένα μόνο unit pattern τότε είναι απλή συνθήκη. Σε αντίθετη περίπτωση, η συνθήκη είναι

σύνθετη και χρησιμοποιείται ξεχωριστή μέθοδος υπολογισμού που θα αναλυθεί στη συνέχεια.

Η κλάση εκχώρησης (assignment) σε περίπτωση που υπάρχει rewriting, όπως στο παράδειγμα P_1, χρησιμοποιεί για κάθε μεταβλητή στο τμήμα των αποτελεσμάτων ένα string για την αποθήκευσή τους και κατάλληλο αριθμό για την αποθήκευση της θέσης τους στο τμήμα των αποτελεσμάτων («0» για τη μεταβλητή A και «1» για τη μεταβλητή Y). Περιέχεται επίσης μία boolean μεταβλητή που έχει την τιμή «true» αν η μεταβλητή υπάρχει επίσης στα pattern elements («false» για τη μεταβλητή A και «true» για τη μεταβλητή Y). Οι εντολές εκχώρησης αποθηκεύονται σε έναν πίνακα από strings και οι μεταβλητές που αυτές έχουν και οι παραμέτρους του δεξιού μέρους της κάθε εντολής εκχώρησης αποθηκεύονται σε έναν κατάλληλο πίνακα που περιέχει vectors. Οι εκχωρήσεις του παραδείγματος για τις τιμές «time» και «label» αποθηκεύονται στον array string και οι τιμές «X.time» και «start of the trip» στους κατάλληλους vectors.

Η κλάση pattern περιέχει το αποτέλεσμα της διαδικασίας ανάγνωσης του pattern query (parsing process). Συγκεκριμένα περιέχει ένα vector που έχει τα atomic pattern elements, ένα vector που περιέχει τα conditions, ένα vector που περιέχει τα easy conditions και ένα vector που περιέχει τα assignments. Για το παράδειγμα P_1, ο atomic pattern element vector περιέχει τέσσερα στοιχεία, ο condition vector περιέχει ένα στοιχείο, ο easy condition vector δεν περιέχει κανένα στοιχείο και ο assignment vector περιέχει δύο στοιχεία.

Η συνάρτηση που περιέχει την αλγοριθμική διαδικασία ταιριάσματος (matching process) μεταξύ μίας υποψήφιας τροχιάς και του query pattern βασίζεται σε ένα μη ντετερμινιστικό πεπερασμένο αυτόματο (Nondeterministic Finite Automaton - NFA).

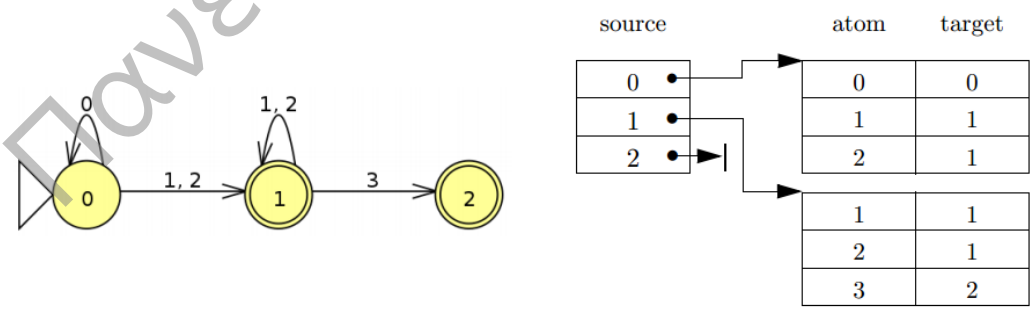
Ένα ντετερμινιστικό πεπερασμένο αυτόματο (Deterministic Finite Automaton - DFA). [7] είναι μία πεντάδα M=(K, Σ, δ, s, F), όπου K είναι ένα πεπερασμένο σύνολο καταστάσεων, Σ είναι ένα αλφάβητο, s είναι η αρχική κατάσταση που ανήκει στο K, F είναι ένα σύνολο τελικών καταστάσεων και είναι μικρότερο ή ίσο από το K και δ είναι η συνάρτηση μετάβασης από το KxΣ στο K. Οι κανόνες σύμφωνα με τους οποίους το αυτόματο M επιλέγει την επόμενη κατάσταση είναι κωδικοποιημένοι στη συνάρτηση μετάβασης. Αν το M είναι στην κατάσταση q που ανήκει στο K και το σύμβολο που διάβασε ως είσοδο είναι α που ανήκει στο Σ, τότε δ(q,α) ανήκει στο K και είναι η μονοσήμαντα ορισμένη κατάσταση στην οποία περνάει το M.

Ένα μη ντετερμινιστικό πεπερασμένο αυτόματο [7] αντίστοιχα είναι μία πεντάδα M=(K, Σ, δ, s, F), όπου K είναι ένα πεπερασμένο σύνολο καταστάσεων, Σ είναι ένα αλφάβητο, s είναι η αρχική κατάσταση που ανήκει στο K, F είναι ένα σύνολο τελικών καταστάσεων και είναι μικρότερο ή ίσο από το K και Δ είναι η σχέση μετάβασης που είναι υποσύνολο του K x (Σ U {e}) x K, όπου e είναι το κενό σύνολο. Αν το M είναι

στην κατάσταση έστω q και το επόμενο σύμβολο εισόδου είναι το α, τότε το M μπορεί να ακολουθήσει οποιαδήποτε μετάβαση της μορφής (q, α, p) ή (q, e, p). Στη δεύτερη περίπτωση δεν διαβάζει κανένα σύμβολο εισόδου.

Για κάθε μη ντετερμινιστικό πεπερασμένο αυτόματο υπάρχει ένα ισοδύναμο ντετερμινιστικό πεπερασμένο αυτόματο [7].

Η συνάρτηση μετάβασης έστω δ του NFA στην περίπτωση της class pattern που αναφέρθηκε προηγουμένως χρησιμοποιεί έναν vector για αντιστοιχίσεις με αριθμούς μεταξύ των καταστάσεων που βρίσκεται το αυτόματο κατά τη διαδικασία parsing των δεδομένων εισόδου. Ειδικότερα, ένα στοιχείο αντιστοίχισης (mapping element) $i \mapsto s_k$ στη θέση s_j στο vector αντιστοίχισης αντιπροσωπεύει μία μετάβαση από τη θέση s_j στη θέση s_k απαιτώντας το ταίριασμα του atomic pattern element p_i με ένα η περισσότερα στοιχεία από το M, όπου είναι το σύνολο των υποτροχιών (unit items m_i) του trajectory. Στη συνέχεια μία μετάβαση t θα δηλώνεται ως $s_j \xrightarrow{i} s_k$, όπου s_j, i και s_k θα αναφέρονται ως t.source, t.atom και t.target αντίστοιχα. Ως t.elem δηλώνεται το πλήθος των pattern element units που ανήκει το εκάστοτε atomic pattern element t.atom. Τέλος, με δ[s] συμβολίζεται το σύνολο των μεταβάσεων από την αρχική κατάσταση s. Στα παραδείγματα P_0 και P_1 το δ[0] αντιστοιχεί σε $\{0 \xrightarrow{0} 0, 0 \xrightarrow{1} 1, 0 \xrightarrow{2} 1\}$, ενώ το δ[2] είναι κενό. Το ανωτέρω σύνολο μεταβάσεων δείχνει ότι από την αρχική κατάσταση μπορούμε είτε να παραμείνουμε ξανά στην αρχική (πρώτη περίπτωση με «*»), είτε να μεταβούμε στην κατάσταση «1» με το πρώτο ή δεύτερο μέρος του «|» (δεύτερη περίπτωση) που μπορεί να είναι και τελική κατάσταση, όπως φαίνεται και στο Σχήμα 10. Η συνάρτηση μετάβασης (vector αντιστοίχισης) για το παράδειγμα P_1 μαζί με το αντίστοιχο NFA παρατίθεται επίσης στο Σχήμα 10.



Σχήμα 10: Transition function και Mapping vector [4]

Η μετατροπή του query pattern string γίνεται με συγκεκριμένη αλγοριθμική διαδικασία. Αρχικά, δεν λαμβάνονται υπόψη τυχόν regular expressions που υπάρχουν (όπως στο παράδειγμα [..|..]+ και [..]?) και όλα τα atomic pattern elements, τα conditions και τα assignments αποθηκεύονται στις κατάλληλες δομές δεδομένων που αναφέρθηκαν παραπάνω. Δημιουργείται στη συνέχεια ένα κατάλληλο string που περιέχει τα regular expressions και γίνεται αντικατάσταση σε αυτό όλων των atomic pattern elements με ακέραιους αριθμούς. Για το παράδειγμα P_0 το string θα περιέχει την τιμή «0(1|2)+3?». Το συγκεκριμένο string στη συνέχεια μετατρέπεται σε NFA με τον McNaughton-Yamada-Thompson αλγόριθμο [8]. Στο παράδειγμα P_0 αντιστοιχεί το NFA του Σχήματος 10.

Με κατάλληλη αλγοριθμική διαδικασία που απεικονίζεται στο Σχήμα 11, εφαρμόζονται οι μεταβάσεις του NFA για τη διαδικασία του ταιριάσματος (matching process) κατά την εκτέλεση του query.

Algorithm 1: *matchesWithoutCondition*

Input: P – a pattern with p atomic pattern elements;
including an NFA δ with n states and a set of final states F;
M – an mlabel of size m.

Output: *true*, if a final state of the NFA is active after processing the mlabel; *false* otherwise.

```
1 let S = {0};
2 for i = 0 to m − 1 do                                   // loop over trajectory
3 |  T = ∅;
4 |  foreach s ∈ S do T = T ∪ δ[s];                       // collect possible transitions
5 |  if T = ∅ then return false;
6 |  S = ∅;
7 |  foreach t ∈ T do                                    // loop over possible transitions
8 |  |  if match(m_i, p_t.atom) then S = S ∪ t.target;
9 return (S ∩ F ≠ ∅);
```

Σχήμα 11: Matching without conditions algorithm [4]

Αρχικά, στο σύνολο S τοποθετείται μόνο η αρχική κατάσταση του NFA. Στη συνέχεια με επαναληπτική λούπα για όλες τις υποτροχιές του trajectory συλλέγονται όλες οι δυνατές καταστάσεις μετάβασης για κάθε στοιχείο που περιέχεται κάθε φορά στο σύνολο S και τοποθετούνται στο σύνολο T. Αν το T σε κάποιο επαναληπτικό βήμα είναι κενό (είτε επειδή δεν υπάρχει άλλη μετάβαση είτε επειδή καμία θέση δεν είναι πλέον ενεργή), τότε σημαίνει ότι δεν βρέθηκε ταίριασμα για το trajectory και ο αλγόριθμος σταματάει την εκτέλεση.

Κατά την εκτέλεση της επαναληπτικής λούπας προστίθενται στο T σταδιακά όλες οι δυνατές μεταβάσεις προς άλλες καταστάσεις από το υπάρχον εκείνη τη στιγμή

σύνολο Τ, ελέγχοντας το ταίριασμα του τρέχοντος στοιχείου m_i με το αντίστοιχο atomic pattern element. Η συνάρτηση match ελέγχει το παραπάνω ταίριασμα και επιπλέον ελέγχει εάν οι easy conditions που αντιστοιχούν στο εκάστοτε atomic pattern element ικανοποιούνται. Μετά την εκτέλεση της επαναληπτικής λούπας επιστρέφεται true μόνο όταν τουλάχιστον μία από τις τελικές καταστάσεις είναι ενεργή.

Στη συνέχεια ακολουθεί εκτέλεση του αλγόριθμου για το παράδειγμα P_0 και το σύνολο M_0 (Σχήμα 6). Με δεδομένο ότι $|M_0| = 5$, θα πραγματοποιηθούν 5 επαναλήψεις κατά την εκτέλεση του αλγόριθμου.

Στην πρώτη επανάληψη, ξεκινώντας από την κατάσταση 0, όπως φαίνεται και στο Σχήμα 10, οι δυνατές μεταβάσεις, δηλαδή το σύνολο Τ αντιστοιχούν στο σύνολο $\{0 \xrightarrow{0} 0, 0 \xrightarrow{1} 1, 0 \xrightarrow{2} 1\}$ = δ_0. Στο unit m_0 κατά τη διαδικασία ταιριάσματος σε αυτό το βήμα αντιστοιχίζεται στα atomic pattern elements 0 και 1, και άρα οι καταστάσεις 0 και 1 γίνονται ενεργές, επομένως S= {0,1}. Στο δεύτερο επαναληπτικό βήμα το $T = \delta_0 \cup \{1 \xrightarrow{1} 1, 1 \xrightarrow{2} 1, 1 \xrightarrow{3} 2\} = \delta_0 \cup \delta_1$ και η συνάρτηση match θα επιστρέψει true μόνο για τα pattern elements p_0 και p_3, άρα το Σ θα είναι σε αυτήν την περίπτωση ίσο με {0,2}.

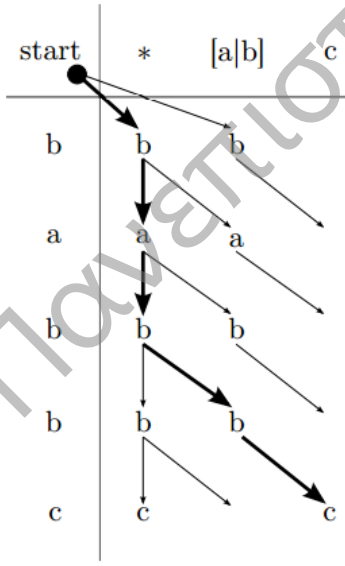
Στο τρίτο επαναληπτικό βήμα, καθώς δεν υπάρχει μετάβαση από την κατάσταση 2 (τελική κατάσταση) το Τ θα είναι ίσο με δ_0. Στην τέταρτη επανάληψη θα ισχύει πάλι το Τ=δ_0 και οι καταστάσεις 0 και 1 γίνονται ενεργές, καθώς τα p_0 και p_3 γίνονται match με m_3. Στην τελευταία επανάληψη το Τ είναι ίσο με $\delta_0 \cup \delta_1$, όπως στο πρώτο επαναληπτικό βήμα. Το τελευταίο στο Μ ταιριάζει με p_0, p_1 και p_3. Άρα θα έχουμε τελικά S = {0,1,2}. Συνεπώς η συνάρτηση επιστρέφει true καθώς στο S υπάρχει τελική κατάσταση.

Για την επαλήθευση του ταιριάσματος ενός pattern με ένα συγκεκριμένο symbolic trajectory είναι απαραίτητος ο έλεγχος αν η ακολουθία από τα pattern elements που αυτό περιέχει επαληθεύονται με το συγκεκριμένο trajectory, όπως περιγράφτηκε στον παραπάνω αλγόριθμο. Αν το pattern περιέχει conditions τότε για την επαλήθευση του ταιριάσματος είναι απαραίτητη η εύρεση ενός συγκεκριμένου binding που να τις επαληθεύει. Για τον υπολογισμό ενός binding που να επαληθεύει τα conditions ενός pattern query είναι απαραίτητη η καταγραφή του ιστορικού ταιριάσματος (matching history) κατά την εκτέλεση του προηγούμενου αλγόριθμου. Με τον όρο ιστορικό εννοείται η καταγραφή για το ποιο unit κάθε φορά έχει γίνει ταίριασμα με ποιο atomic pattern element και ποιοι είναι οι υποψήφιοι για ταίριασμα με το επόμενο διαδοχικό unit στο Μ.

Για την καταγραφή του συγκεκριμένου ιστορικού είναι απαραίτητη η τροποποίηση του αλγόριθμου του Σχήματος 11. Πριν την έναρξη της επαναληπτικής λούπας του ανωτέρω αλγόριθμου είναι απαραίτητη η δήλωση ενός δισδιάστατου πίνακα (που αποθηκεύεται στην κλάση match) ακέραιων αριθμών. Ο συγκεκριμένος πίνακας έχει διαστάσεις mxe, όπου m=$|M_0|$ και e είναι ο αριθμός των pattern elements και χρησιμοποιείται για την αποθήκευση όλων των δυνατών συνδυασμών bindings των μεταβλητών για τον σταδιακό έλεγχο των συνθηκών.

Ειδικότερα, σε κάθε επαναληπτική λούπα του αλγόριθμου όλοι οι δυνατοί διαδοχικοί συνδυασμοί ακέραιων αριθμών που αντιστοιχούν σε pattern elements αποθηκεύονται για κάθε atomic pattern element που ταιριάζεται με ένα unit από M. Με τον όρο διαδοχικοί εννοούμε που είναι δυνατοί σύμφωνα με τη σχέση μετάβασης του NFA. Κατά συνέπεια, τα στοιχεία του εκάστοτε set ακέραιων αριθμών που αποθηκεύονται στον πίνακα A σε κάθε βήμα της επαναληπτικής λούπας αντιστοιχούν σε δείκτες σε υποψήφια pattern units για ταίριασμα με το επόμενο unit που ακολουθεί από το M. Κατά την τελευταία επανάληψη, όταν δηλαδή i=m-1 η τιμή -1 αποθηκεύεται στον πίνακα A μόνο όταν μία τελική κατάσταση είναι ενεργή.

Στη συνέχεια ακολουθεί ένα παράδειγμα εκτέλεσης του αλγόριθμου με υποψήφιο symbolic trajectory το «babbc» και pattern το «X * Y [a|b] Z c». Καθώς το πρώτο atomic pattern element είναι το «*», το πρώτο unit από το symbolic trajectory μπορεί να γίνει ταίριασμα και με το «*» αλλά και με το «[a|b]». Άρα, θα έχουμε, όπως φαίνεται στο Σχήμα 12 σύνδεση από την έναρξη και στα δύο ανωτέρω pattern elements.



start: $\{0,1\}$			
	pattern elements		
unit	0	1	2
0	$\{0,1\}$	$\{2\}$	$\emptyset$
1	$\{0,1\}$	$\{2\}$	$\emptyset$
2	$\{0,1\}$	$\{2\}$	$\emptyset$
3	$\{0,1\}$	$\{2\}$	$\emptyset$
4	$\emptyset$	$\emptyset$	$\{-1\}$

Σχήμα 12: Matching with conditions example graph και table [4]

Καθώς το πρώτο unit είναι το «b», ταιριάζει με τα «*» και «b» από τα pattern units και ως δυνατά επόμενα pattern elements είναι τα «*», «[a|b]» (μετά από «*») και «c» (για το «b») αντίστοιχα. Άρα έχουμε τρεις συνδέσεις για την επόμενη γραμμή. Η διαδικασία αυτή ακολουθείται αντίστοιχα και για τις επόμενες γραμμές φτάνοντας στο τελευταίο unit «c». Υπάρχουν δύο δυνατά ταιριάσματα, ένα με το pattern unit «*» και ένα με το «c». Το τελευταίο μόνο όμως οδηγεί σε πλήρες ταίριασμα. Με έντονο χρωματισμό τα βέλη του Σχήματος 12 δείχνουν τη διαδρομή που οδηγεί σε πλήρες ταίριασμα και στο binding που επαληθεύει κατά συνέπεια το condition. Το binding που επαληθεύει πλήρως αντιστοιχεί σε X={0,1,2}, Y={3} και Z={4}.

Στη δεξιά πλευρά του Σχήματος 12, καταγράφεται το ιστορικό ταιριάσματος σε κατάλληλο πίνακα (A). Τα pattern elements και τα units αντιπροσωπεύονται με ακέραιους αριθμούς που αντιστοιχούν στη θέση τους και οι τιμές του πίνακα αντιστοιχούν σε πιθανά διαδοχικά pattern elements units που θα οδηγήσουν σε ταίριασμα με το επόμενο unit. Στο τελευταίο pattern unit υπάρχει η τιμή «-1» που όπως προαναφέρθηκε οδηγεί σε πλήρες ταίριασμα.

Στο Σχήμα 13 που ακολουθεί παρουσιάζεται ο αντίστοιχος πίνακας A για το παράδειγμα P_0 και symbolic trajectory το M_0.

start: {0, 1}				
		pattern elements		
	unit	X * 0	Y [({th, m} QA) | (_ WelSt)]+ 1	Z [()]? 2
QA	0	QA {0, 1}	QA {1, 2}	∅
Wim	1	Wim {0, 1}	∅	∅
WelW	2	WelW {0, 1}	∅	∅
WelSt	3	WelSt {0, 1}	WelSt {1, 2}	∅
QA	4	QA ∅	QA {−1}	QA {−1}

Σχήμα 13: Matching with conditions table [4]

Στη συνέχεια παρουσιάζονται στα Σχήματα 14 και 15 ο συνολικός αλγόριθμος για την πραγματοποίηση matching με conditions. Ο αλγόριθμος του Σχήματος 14 αποτελεί μία επαναληπτική λούπα που εκτελεί τον αλγόριθμο του Σχήματος 15 που ελέγχει την ύπαρξη binding των μεταβλητών που ικανοποιεί το σύνολο των conditions ενός pattern query.

Algorithm 2: *bindingExistsFrame*

Input: P – a pattern with e elements;
including an NFA δ;
B – an empty binding, i.e., a mapping from a string to a pair of integers
Output: *true*, if a binding is found that fulfills every condition; *false* otherwise.

```
1 foreach j ∈ {t.elem | t ∈ δ_0} do
2     if bindingExists(P, 0, j, B) then return true;        // abort if successful
3 return false;
```

Σχήμα 14: Αλγόριθμος bindingExistsFrame [4]

Ο αλγόριθμος του Σχήματος 15 χρησιμοποιεί αναδρομή για την εύρεση του κατάλληλου binding. Σκοπός του αλγόριθμου είναι να βρει μία διαδρομή στον πίνακα A που να οδηγεί σε τελική κατάσταση (στο NFA). Σε περίπτωση που βρεθεί ένα τέτοιο binding ο αλγόριθμος επιστρέφει true (γραμμή 9). Αρχικά, ελέγχεται αν το pattern element που εξετάζεται στην συγκεκριμένη περίπτωση περιλαμβάνει μεταβλητή έστω u (γραμμή 3). Αν η μεταβλητή αυτή δεν υπάρχει στο binding B τότε προστίθεται σε αυτό (γραμμή 6). Σε διαφορετική περίπτωση το binding της συγκεκριμένης μεταβλητής αυξάνεται κατά 1 (γραμμή 4). Η αναδρομή πραγματοποιείται στις γραμμές 11 και 12, όπου το διαδοχικό unit (i+1), το επόμενο pattern element k και το επεκταμένο binding ξαναδίνονται ως ορίσματα στη συνάρτηση. Σε περίπτωση που η αναδρομική κλήση της συνάρτησης δεν επιστρέψει true τότε οι προσθήκες που έχουν γίνει στο binding αφαιρούνται (γραμμές 13 έως 15).

Algorithm 3: *bindingExists*

Input: P – a pattern with e elements;
i – the current unit number;
j – the current pattern element number;
B – a binding, i.e., a mapping from a string to a pair of integers.

Output: *true*, if a complete binding fulfilling every condition is found starting from unit i and atomic pattern element j; *false* otherwise.

```
1  v = p_elem(j).getVar();
2  inserted = false;
3  if ¬(v is empty) then
4  |  if B contains v then B(v).right++;          // extend existing binding
5  |  else                                        // add new variable to binding
6  |  |  B(v) = (i, i);
7  |  |  inserted = true;
8  if A_{i,j} contains −1 then                    // complete match
9  |  if conditionsMatch(P, B) then return true;  // abort if successful
10 else
11 |  foreach k ∈ A_{i,j} do
12 |  |  if bindingExists(P, i + 1, k, B) then return true;   // abort if successful
13 if ¬(v is empty) then
14 |  if inserted then erase v from B;
15 |  else B(v).right--;
16 return false;
```

Σχήμα 15: Αλγόριθμος bindingExists [4]

Στο Σχήμα 16 παρουσιάζεται η εκτέλεση του αλγόριθμου για το παράδειγμα P_1 και symbolic trajectory το M_0. Στον συγκεκριμένο πίνακα, το επίπεδο της αναδρομής απεικονίζεται στην στήλη i και αντιστοιχεί επιπλέον και στον αριθμό του τρέχοντος unit. Το εκάστοτε κάθε φορά τρέχον pattern element αντιστοιχεί στην στήλη j. Ο έλεγχος κάθε φορά του τρέχοντος set που υπάρχει στο $A_{i,j}$ ξεκινάει από το μεγαλύτερο αριθμητικά set σε κάθε γραμμή του πίνακα καθώς αν βρεθεί ταίριασμα σε element που βρίσκεται δεξιότερα στον πίνακα σε μία γραμμή αυτού, τότε αυξάνουν οι πιθανότητες να βρεθούμε σε τελική κατάσταση γρηγορότερα. Σε αυτό το παράδειγμα πλήρες binding επιτυγχάνεται στην περίπτωση που i=m-1=4, όπως φαίνεται στην τελευταία γραμμή του πίνακα. Το επόμενο βήμα είναι ο έλεγχος αν το συγκεκριμένο binding που βρέθηκε επαληθεύει όλα τα conditions του pattern query.

i	j	B at call	B updated	final	proceed	undo B
0	1	$\emptyset$	$\{\mathtt{Y} \mapsto [0,0]\}$	no	yes	no
1	2	$\{\mathtt{Y} \mapsto [0,0]\}$	$\{\mathtt{Y} \mapsto [0,0], \mathtt{Z} \mapsto [1,1]\}$	no	no	yes
1	1	$\{\mathtt{Y} \mapsto [0,0]\}$	$\{\mathtt{Y} \mapsto [0,1]\}$	no	no	yes
0	0	$\emptyset$	$\{\mathtt{X} \mapsto [0,0]\}$	no	yes	no
1	1	$\{\mathtt{X} \mapsto [0,0]\}$	$\{\mathtt{X} \mapsto [0,0], \mathtt{Y} \mapsto [1,1]\}$	no	no	yes
1	0	$\{\mathtt{X} \mapsto [0,0]\}$	$\{\mathtt{X} \mapsto [0,1]\}$	no	yes	no
2	1	$\{\mathtt{X} \mapsto [0,1]\}$	$\{\mathtt{X} \mapsto [0,1], \mathtt{Y} \mapsto [2,2]\}$	no	no	yes
2	0	$\{\mathtt{X} \mapsto [0,1]\}$	$\{\mathtt{X} \mapsto [0,2]\}$	no	yes	no
3	1	$\{\mathtt{X} \mapsto [0,2]\}$	$\{\mathtt{X} \mapsto [0,2], \mathtt{Y} \mapsto [3,3]\}$	no	yes	no
4	2	$\{\mathtt{X} \mapsto [0,2], \mathtt{Y} \mapsto [3,3]\}$	$\{\mathtt{X} \mapsto [0,2], \mathtt{Y} \mapsto [3,3], \mathtt{Z} \mapsto [4,4]\}$	yes	no	no

Σχήμα 16: Εκτέλεση του αλγόριθμου bindingExists για το παράδειγμα P_1 και symbolic trajectory το M_0 [4]

2.3 Spatial Keyword Queries

Τα pattern queries σε σημασιολογικές τροχιές που αναφέρθηκαν στις προηγούμενες ενότητες ανήκουν στην ευρύτερη κατηγορία των mining queries. Με τον όρο spatial keyword queries κατηγοριοποιούνται queries που συνδυάζουν χωρικά και λεκτικά κριτήρια στην αναζήτηση αντικειμένων.

Τα ανωτέρω queries μπορούν να χωριστούν σε τέσσερις βασικές κατηγορίες όσον αφορά το είδος της αναζήτησης [9]:
-Boolean Range Query
-Boolean kNN Query
-Top-k kNN Query
-Spatio-Textual Similarity Joins [10]

Το Boolean Range Query (BRQ) αναφέρεται στην ανάκτηση όλων των αντικειμένων που έχουν όλα τα keywords αναζήτησης και βρίσκονται στην χωρική περιοχή που έχει δοθεί ως κριτήριο αναζήτησης.

Το Boolean kNN Query (BkQ) αναφέρεται στην ανάκτηση k αντικειμένων που έχουν όλα τα keywords αναζήτησης και είναι ταξινομημένα ανάλογα με την απόσταση από το χωρικό σημείο που έχει δοθεί ως κριτήριο αναζήτησης.

Το Top-k kNN Query (TkQ) αναφέρεται στην ταξινομημένη βαθμολογικά ανάκτηση k αντικειμένων. Ο βαθμός σχετικότητας με τα κριτήρια αναζήτησης προκύπτει από κατάλληλη συνάρτηση που συνδυάζει την απόσταση από τη περιοχή που έχει δοθεί

ως κριτήριο αναζήτησης και τη λεκτική πληροφορία πόσο σχετική είναι με τα keywords αναζήτησης.

Τα Spatio-Textual Similarity Joins [10] αναφέρεται στην ανάκτηση αντικειμένων που βρίσκονται στην ίδια χωρική περιοχή και έχουν ταυτόχρονα λεκτική ομοιότητα (textual similarity) μεταξύ τους.

Για την υλοποίηση των παραπάνω κατηγοριών queries χρησιμοποιείται ένα μεγάλο εύρος από διαφορετικά DBMS. Ο μεγάλος αριθμός των διαφορετικών υλοποιήσεων έχει ως αποτέλεσμα τη χρησιμοποίηση και υλοποίηση διαφορετικών ειδών και τρόπων ευρετηρίασης των δεδομένων. Μία ευρύτερη κατηγοριοποίηση που καλύπτει το μεγαλύτερο αριθμό των διαφορετικών indexes που έχουν αναπτυχθεί τα ταξινομεί λαμβάνοντας υπόψη τρεις παράγοντες [9]:
-Τον τρόπο ευρετηρίασης των χωρικών δεδομένων (spatial indexing scheme)
-Την κατηγορία του λεκτικού ευρετηρίου που χρησιμοποιείται (text index)
-Τον βαθμό σύνδεσης των δύο ανωτέρω ευρετηρίων

Όσον αφορά το spatial indexing scheme, τα ευρετήρια μπορούν να ταξινομηθούν σε τρεις διαφορετικές κατηγορίες [9]:
-Ευρετήρια που βασίζονται στα R-Δέντρα (R-Tree based indices)
-Ευρετήρια που βασίζονται σε χωρικά πλέγματα (Grid based indices)
-Ευρετήρια που βασίζονται σε γραμμές διάσχισης του χώρου (Space filling curve based indices)

Τα R-Tree based indices χρησιμοποιούν το R-Tree ή κάποια παραλλαγή αυτού για την ευρετηρίαση των δεδομένων. Τα περισσότερα χωρικά-λεκτικά ευρετήρια (geo-textual indices) ανήκουν σε αυτήν την κατηγορία και συνδυάζουν το R-Tree με κάποιο ανεστραμμένο αρχείο για το text index.

Τα Grid based indices συνδυάζουν ένα grid index με ένα text index. Ένα grid index χωρίζει τη χωρική περιοχή σε έναν προκαθορισμένο αριθμό από μικρότερες πολυγωνικές περιοχές. Το text index σε αυτά είναι συνήθως ανεστραμμένο αρχείο και συνδυάζεται σε μεγαλύτερο ή μικρότερο βαθμό με το grid index.

Τα Space filling curve based indices συνδυάζουν ένα text index (κυρίως ανεστραμμένο αρχείο) με μία γραμμή διάσχισης του χώρου. Κυρίως χρησιμοποιούνται Hilbert curve ή Z-curve για την δημιουργία του ευρετηρίου.

Όσον αφορά τώρα το text indexing scheme, τα ευρετήρια μπορούν να χωριστούν σε δύο κατηγορίες, τα ανεστραμμένα αρχεία (inverted files) και τα bitmaps.

Σχετικά με την πρώτη κατηγορία, αν υποθέσουμε ότι έχουμε μία συλλογή από κείμενα που ανήκουν στη σημασιολογική πληροφορία κάποιων αντικειμένων, ένα ανεστραμμένο ευρετήριο αντιστοιχεί για κάθε όρο που υπάρχει στα κείμενα της συλλογής μία λίστα με τα κείμενα της συλλογής που αυτός υπάρχει. Τα bitmap αντιστοιχούν σε συμβολοσειρές που βρίσκονται σε κόμβους του εκάστοτε δέντρου

ευρετηρίου. Κάθε bit τους συμβολίζει την ύπαρξη ή όχι ενός όρου από τις συλλογές των κειμένων των παιδιών των κόμβων.

Όσον αφορά τον τρίτο παράγοντα, τον βαθμό σύνδεσης μεταξύ των χωρικών και λεκτικών ευρετηρίων, μπορούμε να διαχωρίσουμε τα ευρετήρια σε δύο κυρίως κατηγορίες: στα ευρετήρια που δεν υπάρχει μεγάλος βαθμός σύνδεσης μεταξύ τους και στα ευρετήρια που οι δύο κατηγορίες έχουν μεγάλο βαθμό ενοποίησης μεταξύ τους.

Στη συνέχεια γίνεται σύντομη ανασκόπηση των σημαντικότερων geo-textual indices καθώς και σύγκριση της απόδοσης τους σε σχέση με ευρετήρια που ανήκουν σε διαφορετικές κατηγορίες [9].

Τα περισσότερα geo-textual indices χρησιμοποιούν για την ευρετηρίαση των χωρικών δεδομένων τα R-trees και για τα λεκτικά δεδομένα ανεστραμμένα αρχεία. Το IF-R*-Tree [11] χρησιμοποιεί το R-Tree για την ευρετηρίαση των χωρικών δεδομένων. Δημιουργείται για κάθε διαφορετικό λεκτικό όρο ένα ξεχωριστό R-Tree για τα χωρικά δεδομένα που περιέχουν τον συγκεκριμένο όρο. Κατά συνέπεια, ο αριθμός των διαφορετικών R-Trees αντιστοιχεί στον αριθμό των διαφορετικών όρων που υπάρχουν στα κείμενα της συλλογής. Το R*-Tree-IF [11] αποτελεί μία παραλλαγή του IF-R*-Tree που δημιουργείται μόνο ένα R-Tree λαμβάνοντας υπόψη τη χωρική διάσταση των αντικειμένων μόνο. Στη συνέχεια για κάθε φύλλο του δέντρου δημιουργείται ένα ανεστραμμένο αρχείο για την ευρετηρίαση των όρων που υπάρχουν στα αντικείμενα του φύλλου. Το ευρετήρια μπορούν να χρησιμοποιηθούν για BRQ και BkQ.

Το KR*-Tree [12] χρησιμοποιεί επίσης το R-Tree για την ευρετηρίαση των χωρικών δεδομένων. Κάθε κόμβος του δέντρου σε αυτό το ευρετήριο περιέχει ένα set από τα keywords των κόμβων που είναι «παιδιά» της. Οι κόμβοι του δέντρου περιέχονται σε ανεστραμμένη λίστα, δηλαδή για κάθε keyword αναφέρεται σε ποιους κόμβους υπάρχει, προκειμένου να περιοριστεί το εύρος της αναζήτησης (pruning) στο R-Tree. Αρχικά η αναζήτηση ξεκινάει με λεκτικά κριτήρια και βρίσκει το set των κόμβων που ικανοποιούν τα συγκεκριμένα κριτήρια της αναζήτησης Στη συνέχεια ελέγχεται αν ικανοποιούνται για τους συγκεκριμένους κόμβους και τα χωρικά κριτήρια. Η αναζήτηση στη συνέχεια γίνεται για τα παιδιά των συγκεκριμένων κόμβων που ικανοποιούν τις χωρικές και λεκτικές απαιτήσεις, μέχρι το επίπεδο των φύλλων του δέντρου. Το ευρετήριο χρησιμοποιείται για BRQ και BkQ.

Το IR²-Tree [13] χρησιμοποιεί το R-Tree για την ευρετηρίαση των χωρικών δεδομένων. Για την διαχείριση των λεκτικών δεδομένων χρησιμοποιούνται signature files [14]. Κάθε κορυφή του δέντρου περιέχει σε μορφή bitmap ένα signature file που

αποτελεί την ένωση όλων των signatures των παιδιών του. Το ευρετήριο χρησιμοποιείται για BRQ και BkQ.

Το Hybrid Spatial-Keyword Indexing (SKI) [15] χρησιμοποιεί το R-Tree για την ευρετηρίαση των χωρικών δεδομένων και bitmaps για τα λεκτικά δεδομένα. Κάθε κορυφή του δέντρου που βρίσκεται ακριβώς πάνω από τα φύλλα του δέντρου ονομάζεται super-node. Ένας super-node περιέχει ένα ανεστραμμένο αρχείο με bitmaps. Κάθε όρος που υπάρχει στα κείμενα της συλλογής αντιστοιχεί σε ένα bitmap. Στη συνέχεια ελέγχεται αν ένα αντικείμενο περιέχει ή όχι έναν όρο παίρνοντας κατάλληλη τιμή στο bitmap του κάθε όρου αν περιέχεται. Το ευρετήριο χρησιμοποιείται για BRQ και BkQ.

Το Inverted File R-Tree (IR-Tree) [16,17] χρησιμοποιεί επίσης το R-Tree για την ευρετηρίαση των χωρικών δεδομένων. Κάθε κορυφή του δέντρου περιέχει μία «περίληψη» των χωρικών και λεκτικών περιεχομένων των κόμβων των παιδιών του. Επιπλέον, σε κάθε κορυφή του δέντρου περιέχεται ένας δείκτης προς ένα ανεστραμμένο αρχείο που περιέχει όλα τα λεκτικά δεδομένα των αντικειμένων που υπάρχουν στους κόμβους που είναι παιδιά αυτού του κόμβου. Τα φύλλα του δέντρου επίσης περιέχουν δείκτη προς ανεστραμμένο αρχείο με τα περιεχόμενα των αντικειμένων που υπάρχουν σε αυτά. Ειδικότερα, όσον αφορά τη χωρική διάσταση των δεδομένων κάθε κορυφή περιέχει ένα MBR που περικλείει όλα τα αντικείμενα που υπάρχουν στα παιδιά του, όπως και στο κανονικό R-Tree. Παράλληλα περιέχει όμως και ένα ψευδο-έγγραφο (pseudo-document) που αντιπροσωπεύει όλους τους λεκτικούς όρους που υπάρχουν στα παιδιά της κορυφής. Η αντιπροσώπευση επιτυγχάνεται για κάθε όρο με την τοποθέτηση στο ψευδο-έγγραφο του μέγιστου βάρους για κάθε όρο από τα βάρη των όρων που βρίσκονται στα παιδιά του κόμβου. Για παράδειγμα, έστω ότι υπάρχει ο λεκτικός όρος «restaurant», όπως φαίνεται και στα Σχήματα 17 και 18.

Vocabulary	InvFile 4 Posting lists	InvFile 5 Posting lists	InvFile 6 Posting lists	InvFile 7 Posting lists
Chinese	$\langle O_1.doc, 5\rangle$	$\langle O_3.doc, 7\rangle$	$\langle O_5.doc, 4\rangle$	$\langle O_7.doc, 1\rangle$
Spanish	$\langle O_2.doc, 5\rangle$	$\langle O_8.doc, 3\rangle$		$\langle O_6.doc, 4\rangle$,$\langle O_7.doc, 1\rangle$
restaurant	$\langle O_1.doc, 5\rangle$,$\langle O_2.doc, 5\rangle$	$\langle O_4.doc, 7\rangle$,$\langle O_5.doc, 4\rangle$,$\langle O_8.doc, 3\rangle$	$\langle O_6.doc, 3\rangle$,$\langle O_7.doc, 4\rangle$	
food		$\langle O_3.doc, 1\rangle$,$\langle O_4.doc, 1\rangle$		$\langle O_7.doc, 1\rangle$

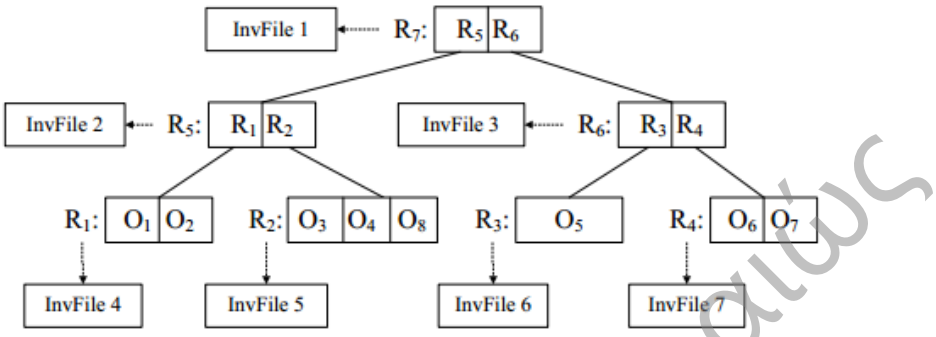
Table 1: Posting Lists for InvFile 4, 5, 6, and 7

Vocabulary	InvFile 2 Posting lists	InvFile 3 Posting lists	InvFile 1 Posting lists
Chinese	$\langle R_1.doc, 5\rangle$,$\langle R_2.doc, 7\rangle$	$\langle R_3.doc, 4\rangle$,$\langle R_4.doc, 1\rangle$	$\langle R_5.doc, 7\rangle$,$\langle R_6.doc, 4\rangle$
Spanish	$\langle R_1.doc, 5\rangle$,$\langle R_2.doc, 3\rangle$	$\langle R_4.doc, 4\rangle$	$\langle R_5.doc, 5\rangle$,$\langle R_6.doc, 4\rangle$
restaurant	$\langle R_1.doc, 5\rangle$,$\langle R_2.doc, 7\rangle$	$\langle R_3.doc, 4\rangle$,$\langle R_4.doc, 4\rangle$	$\langle R_5.doc, 7\rangle$,$\langle R_6.doc, 4\rangle$
food	$\langle R_2.doc, 1\rangle$	$\langle R_4.doc, 1\rangle$	$\langle R_5.doc, 1\rangle$,$\langle R_6.doc, 1\rangle$

Table 2: Posting Lists for InvFile 1, 2, and 3

Σχήμα 17: Ανεστραμμένες λίστες στο IR-Tree index [16]

Το βάρος του λεκτικού όρου είναι 7 στην εγγραφή «R2» στην ανεστραμμένη λίστα για το Inverted File 2 στο ψευδοέγγραφο καθώς είναι το μέγιστο βάρος του όρου από τα 3 έγγραφα που υπάρχουν στην κορυφή R2.



Σχήμα 18: IR-Tree index [16]

Κατά συνέπεια, το κριτήριο για την απόσταση μεταξύ ενός ψευδοέγγραφου και των δεδομένων ενός ερωτήματος λαμβάνει υπόψη τόσο τη λεκτική απόσταση από τους όρους του ερωτήματος και του ψευδοέγγραφου, όσο και την χωρική απόσταση. Η μέτρηση της συνδυασμένης συνολικής απόστασης επιτυγχάνεται με κατάλληλη συνάρτηση που λαμβάνει υπόψη και τους δύο παράγοντες και ονομάζεται Minimum Spatial Textual Distance (MINDST) για τα ψευδοέγγραφα και DST για τα κανονικά αντικείμενα προς εξέταση [16]. Στο Σχήμα 19 παρουσιάζεται ο αλγόριθμος [16] για TkQ.

Algorithm 2 $\mathrm{LkT}(Query, Index, k)$

```
1:  Queue ← NewPriorityQueue();
2:  Queue.Enqueue(Index.RootNode, 0);
3:  while not Queue.IsEmpty() do
4:      Element ← Queue.Dequeue();
5:      if Element is an object then
6:          if not Queue.IsEmpty() and D_ST(Query, Object) >
                Queue.First().Key then
7:              Queue.Enqueue(Object, D_ST(Query, Object));
8:          else
9:              Report Element as the next nearest object;
10:             if k nearest objects have been found then
11:                 break;
12:     else if Element is a leaf node then
13:         for each entry(Object) in leaf node Element do
14:             Queue.Enqueue(Object, D_ST(Query, Object));
15:     else
16:         for each entry(Node) in node Element do
17:             Queue.Enqueue(Node, MIND_ST(Query, Node));
```

Σχήμα 19: IR-Tree Location Aware Top-k Text Retrieval Algorithm [16]

Ο αλγόριθμος χρησιμοποιεί για τη διάσχιση του δέντρου τον αλγόριθμο BFS (Best-first traversal algorithm) για την ανάκτηση των k αντικειμένων. Ειδικότερα, γίνεται χρήση μίας ουράς προτεραιότητας (priority queue) για την αποθήκευση σε αυτήν των κορυφών και των κόμβων που δεν έχει γίνει ακόμη «επίσκεψη». Για την σύγκριση των αντικειμένων και των κορυφών χρησιμοποιούνται οι τιμές των MINDST και DST για να διαλέξει ο αλγόριθμος ποιο κόμβο θα επισκεφθεί.

Για την παρουσίαση του αλγόριθμου θα εκτελεστεί ο αλγόριθμος για το δέντρο του Σχήματος 18, με k=1 και query το «Chinese restaurant». Στο Σχήμα 20 παρουσιάζονται υπολογισμένες οι τιμές των MINDST και DST για το συγκεκριμένο παράδειγμα.

Objects	Dist.	D_{ST}	Rectangles	Dist.	$MIND_{ST}$
O_1	2	0.238	R_1	2	0.238
O_2	5	0.512	R_2	2	0.1048
O_3	6	0.481	R_3	4	0.368
O_4	7	0.517	R_4	5	0.42
O_5	3	0.53	R_5	0.5	0.05119
O_6	9	0.58	R_6	1	0.269
O_7	8	0.55	R_7	0	0
O_8	8	0.686			

Σχήμα 20: IR-Tree Location Aware Top-k Text Retrieval Algorithm example [16]

Στο πρώτο βήμα ο αλγόριθμος θα επισκεφθεί το root του δέντρου (R_7) και θα μπουν στην ουρά τα παιδιά του, δηλαδή Queue = {(R_5,0.05119), (R_6,0.269)}. Στη συνέχεια θα βγει από την ουρά το R_5 και θα μπουν στην ουρά τα παιδιά του, δηλαδή Queue = {(R_2,0.1048), (R_1,0.238), (R_6,0.269)}. Έπειτα, θα βγει το R_2 και θα μπουν τα παιδιά του δηλαδή τα O_3, O_4 και O_8, δηλαδή Queue = {(R_1,0.238), (R_6,0.269), (O_3,0.481), (O_4,0.517), (O_8,0.686)}. Στη συνέχεια θα βγει το R_1 και θα μπουν τα παιδιά του δηλαδή τα O_1 και O_2, δηλαδή Queue = {(O_1,0.238), (R_6,0.269), (O_3,0.481), (O_2,0.512), (O_4,0.517), (O_8,0.686)}. Τέλος, θα βγει από την ουρά πλέον το αντικείμενο O_1 και όχι κάποιο ψευδοέγγραφο, με αποτέλεσμα τον τερματισμό του αλγόριθμου.

Το ανωτέρω ευρετήριο μπορεί να χρησιμοποιηθεί μπορεί να χρησιμοποιηθεί και στις υπόλοιπες δύο κατηγορίες ερωτημάτων (BRQ και BkQ). Υπάρχει μεγάλος αριθμός από διαφορετικές παραλλαγές του συγκεκριμένου ευρετηρίου, όπως το DIR-tree, το CIR-tree και το CDIR-tree [9,16,17,18]. Το DIR-tree λαμβάνει υπόψη κατά την κατασκευή του δέντρου και τα χωρικά και τα λεκτικά δεδομένα με σκοπό τη δημιουργία του μικρότερου δυνατού MBR κάθε φορά με ταυτόχρονα όμως όσον το δυνατόν περισσότερη ομοιογένεια στα λεκτικά δεδομένα όσον αφορά τα αντικείμενα που βρίσκονται κάθε φορά στο εκάστοτε MBR. Το CIR-tree αντίθετα ομαδοποιεί τα αντικείμενα για την κατασκευή του IR-Tree λαμβάνοντας υπόψη μόνο την λεκτική περιγραφή τους. Το CDIR-tree συνδυάζει τις τεχνικές των δύο παραπάνω ευρετηρίων.

Το IR-Tree [19] παρότι έχει ίδια ονομασία με το προηγούμενο ευρετήριο διαφέρει από αυτό καθώς ενώ το IR-Tree [16,17] χρησιμοποιεί ανεστραμμένα αρχεία για κάθε κορυφή ξεχωριστά, το IR-Tree [19] χρησιμοποιεί ένα συνολικό ανεστραμμένο αρχείο για όλες τις κορυφές. Ο τρόπος οργάνωσης του ανεστραμμένου αρχείου είναι ίδιος με το KR*-Tree [12], με τη διαφορά ότι το KR*-Tree [12] δεν χρησιμοποιεί βάρη για τους λεκτικούς όρους. Το ευρετήριο χρησιμοποιείται για TkQ.

Το WIR-Tree [20] είναι επίσης μία παραλλαγή του IR-Tree [16,17]. Το συγκεκριμένο ευρετήριο χωρίζει τα αντικείμενα έτσι ώστε κάθε ομάδα να περιέχει όσο το δυνατόν λιγότερους κοινούς λεκτικούς όρους. Ξεκινάει τον διαχωρισμό με τον όρο που περιέχεται στα περισσότερα αντικείμενα και συνεχίζει τον διαχωρισμό με τον δεύτερο στη σειρά όρο, κ.ο.κ. Ο διαχωρισμός σταματάει όταν κάθε ομάδα περιέχει έναν προκαθορισμένο αριθμό από αντικείμενα. Κάθε ομάδα που δημιουργείται αντιστοιχεί σε ένα φύλλο του δέντρου. Το δέντρο ακολουθεί τη δομή του IR-Tree [16,17], χρησιμοποιώντας ως αντικείμενα για τον σχηματισμό του δέντρου τις συγκεκριμένες ομάδες που έχουν συγκροτηθεί με τον διαχωρισμό των όρων.. Στην περίπτωση που χρησιμοποιείται για BRQ, τότε το ανεστραμμένο αρχείο αντικαθίσταται με μια μορφή ανεστραμμένου bitmap και το ευρετήριο ονομάζεται WIBR-tree. Το ευρετήριο μπορεί να χρησιμοποιηθεί και για τα τρία είδη queries.

Το Spatial Inverted Index (S2I) [21] χρησιμοποιεί μία παραλλαγή του R-Tree που ονομάζεται aR-tree και αντιστοιχεί αντικείμενα στο δέντρο λαμβάνοντας υπόψη τον όρο που περιέχεται σε αυτά με τη μεγαλύτερη συχνότητα. Στο aR-tree κάθε κόμβος περιέχει μία τιμή που έχει τη μέγιστη επίδραση όσον αφορά το βαθμό λεκτικής σχετικότητας από τους κόμβους-παιδιά και τους όρους που αυτοί περιέχουν. Οι σπάνιοι όροι αντίθετα βρίσκονται σε ένα ξεχωριστό ανεστραμμένο αρχείο. Το ευρετήριο χρησιμοποιείται για BRQ, BkQ και TkQ.

Το ευρετήριο Text Primary Index (TS) [22] είναι grid-based index και χρησιμοποιεί ξεχωριστές ανεστραμμένες λίστες για κάθε όρο της συλλογής και κάθε μία από αυτές αντιστοιχίζεται με ένα ξεχωριστό χωρικό πλέγμα. Είναι δηλαδή text-first index. Το Spatial Primary Index (ST) [22] αποτελεί μία διαφορετική παραλλαγή του TS που είναι spatial-first index. Το ευρετήριο χρησιμοποιείται για BRQ.

Το Spatial Keyword Inverted File (SKIF) [23] χρησιμοποιεί μία μορφή ανεστραμμένου αρχείου που περιλαμβάνει την αποθήκευση τόσο χωρικών όσο λεκτικών δεδομένων προκειμένου να γίνεται ταυτόχρονη διαχείρισή τους. Για την λεκτική πληροφορία δημιουργείται ευρετήριο με ανεστραμμένη λίστα. Αντίστοιχα για τα χωρικά δεδομένα, δημιουργούνται ξεχωριστά χωρικά πλέγματα και ελέγχεται αν κάθε χωρικό αντικείμενο περιέχει ή όχι σε αυτές, δημιουργώντας έτσι μία ανεστραμμένη λίστα για τα χωρικά αντικείμενα, που έχει παρόμοια μορφή με την ανεστραμμένη λίστα για τη λεκτική πληροφορία, αντί όμως για λεκτικούς όρους, χρησιμοποιούνται χωρικά πλέγματα (grid cells). Το ευρετήριο χρησιμοποιείται για BRQ.

Το SFC-QUAD index [24] περιέχει σε συμπιεσμένη μορφή τα περιεχόμενα ενός ανεστραμμένου αρχείου. Η χωρική πληροφορία περιέχεται στο ανεστραμμένο αρχείο με τη μορφή αριθμητικών τιμών που αντιστοιχούν σε χωρικές θέσεις σε μία καμπύλη Z-curve. Για την προσπέλαση της Z-curve χρησιμοποιείται μία κατάλληλη μορφή δέντρου που ονομάζεται Quad-tree και γίνεται χρήση αυτής για την αντιστοίχιση του εκάστοτε query και των χωρικών του κριτηρίων με αριθμητικές τιμές που αντιστοιχούν στην καμπύλη που διασχίζει το χώρο. Το ευρετήριο χρησιμοποιείται για BRQ.

Στη συνέχεια ακολουθεί στο Σχήμα 21 ένας συγκεντρωτικός πίνακας που δείχνει κάθε ευρετήριο σε ποια κατηγορία ανήκει, καθώς και το βαθμό ενοποίησης μεταξύ του χωρικού και του λεκτικού index που χρησιμοποιεί. Το σύμβολο √ δείχνει το είδος του query που χρησιμοποιήθηκε αρχικά το ευρετήριο και το σύμβολο △ τις δυνατότητες επέκτασης του query στις υπόλοιπες κατηγορίες των queries.

Index	Abbr	Spatial part	Textual part	Combination scheme	BkQ	TkQ	BRQ
ST [19]	ST	Grid	inverted file	spatial-first			√
TS [19]	TS	Grid	inverted file	text-first			√
IF-R*-Tree [26]	IF-R*	R*-Tree	inverted file	text-first	△		√
R*-Tree-IF [26]	R*-IF	R*-Tree	inverted file	spatial-first		△	√
SF2I [5]	SF2I	SFC	inverted file	spatial-first			√
KR*-Tree [12]	KR*	R*-Tree	inverted file	tightly combined	△		√
IR^2-Tree [9]	IR^2	R-Tree	bitmaps	tightly combined	√		△
IR-Tree [7,20]	IR	R-Tree	inverted file	tightly combined	△	√	△
IR-Tree [16]	IRLi	R-Tree	inverted file	tightly combined		√	
SKIF [14]	SKIF	Grid	inverted file	tightly combined			√
SKI [4]	SKI	R-Tree	bitmaps	spatial-first	√		
S2I [18]	S2I	R-Tree	inverted file	text-first	△	√	△
WIBR-Tree [21]	WIBR	R-Tree	inverted bitmaps	tightly combined	√		△
SFC-QUAD [6]	SFC-Quad	SFC	inverted file	tightly combined			√

Σχήμα 21: Geo-textual indices comparison [9]

Αναμφισβήτητα, η επιλογή του καταλληλότερου index πρέπει να γίνει λαμβάνοντας υπόψη το είδος της εφαρμογής και της υπηρεσίας που θα χρησιμοποιηθεί, τις απαιτήσεις αυτών καθώς και το είδος και τον όγκο των δεδομένων που θα διαχειρισθούν. Η εφαρμογή των παραπάνω 12 ευρετηρίων σε συγκριτικές δοκιμές μεταξύ τους χρησιμοποιώντας τα ίδια data sets προς ευρετηρίαση οδήγησε στα παρακάτω συμπεράσματα [9]:

-Τα Grid based indices υστερούν σε αποδοτικότητα σε BRQ σε σύγκριση με τις υπόλοιπες δύο κατηγορίες. Για τα συγκεκριμένα data sets που χρησιμοποιήθηκαν καλύτερη απόδοση για το BRQ είχε το SFC-Quad index.

-Όταν το data set έχει σχετικά μικρό μέγεθος σε BkQ και ο αριθμός των keywords είναι μικρός το S2I index έχει μεγαλύτερη απόδοση σε σύγκριση με τα υπόλοιπα ευρετήρια που μπορούν να διαχειριστούν τη συγκεκριμένη κατηγορία ερωτημάτων. Αντίθετα, όταν ο αριθμός των keywords είναι μεγαλύτερος και λαμβάνονται υπόψη απαιτήσεις όγκου αποθήκευσης των δεδομένων τότε το WIBR index έχει καλύτερη απόδοση. Κατά συνέπεια όταν δεν υπάρχουν μεγάλες απαιτήσεις τόσο ως προς τον όγκο των δεδομένων όσο και προς την πολυπλοκότητα των ερωτημάτων προτιμότερο ευρετήριο είναι το S2I. Σε αντίθετη περίπτωση καλύτερες αποδόσεις έχει το WIBR index.

-Στο TkQ το S2I index έχει καλές αποδόσεις όταν ο αριθμός των keywords είναι μικρός. Όταν το ερώτημα έχει μεγαλύτερο αριθμό από keywords και λαμβάνονται υπόψη απαιτήσεις όγκου αποθήκευσης των δεδομένων τότε το CDIR index είναι

καλύτερη επιλογή. Επιπρόσθετα αν η επιλογή της k παραμέτρου είναι αρκετά μεγάλη τότε το IR-Tree [19] έχει επίσης καλές αποδόσεις.

Συμπερασματικά, όπως αναφέρθηκε και νωρίτερα, τα spatial keyword queries έχουν ως αντικείμενο αναζήτησης συγκεκριμένα χωρικά αντικείμενα που περιλαμβάνουν ταυτόχρονα και λεκτική πληροφορία. Δεν περιλαμβάνεται δηλαδή η χρονική πληροφορία τόσο ως κριτήρια αναζήτησης όσο και ως δομικό στοιχείο στα αντικείμενα προς αναζήτηση. Η αναζήτηση επομένως σημασιολογικών τροχιών είτε στο σύνολο τους είτε συγκεκριμένων υποτροχιών αυτών που επαληθεύουν συγκεκριμένα χωροχρονικά και λεκτικά κριτήρια αναζήτησης δεν μπορεί να πραγματοποιηθεί χωρίς την τροποποίηση των χρησιμοποιούμενων ευρετηρίων και των αλγόριθμων αναζήτησης προκειμένου να περιλαμβάνεται και το χρονικό στοιχείο.

Επιπρόσθετα, η αναζήτηση χρησιμοποιώντας ως κριτήριο αναζήτησης patterns (regular expressions) με χωροχρονικά και λεκτικά κριτήρια που μπορούν να παρουσιάζουν σημασιολογικές τροχιές δεν μπορεί επίσης να πραγματοποιηθεί χωρίς την τροποποίηση των αλγόριθμων αναζήτησης και των αντίστοιχων ευρετηρίων που παρουσιάστηκαν προηγουμένως.

Η εφαρμογή παραλλαγών του R-Tree και των ανεστραμμένων ευρετηρίων από την πλειοψηφία των περιπτώσεων που αναφέρθηκαν νωρίτερα μπορούν να χρησιμοποιηθούν με κατάλληλη τροποποίηση ή επέκταση τους προκειμένου να γίνει αποθήκευση της πληροφορίας των σημασιολογικών τροχιών. Η ανωτέρω δομή που θα επιτρέπει την αναζήτηση με τη χρήση regular expressions σε σημασιολογικές τροχιές χρησιμοποιώντας χωροχρονικά και λεκτικά κριτήρια πρέπει να λαμβάνει υπόψη τα συγκεκριμένα χαρακτηριστικά των semantic trajectories που αναφέρθηκαν στις προηγούμενες ενότητες προκειμένου να πραγματοποιηθούν αποδοτικές αναζητήσεις στα δεδομένα των τροχιών.

Στην επόμενη ενότητα παρουσιάζεται μία επέκταση του R-Tree που ικανοποιεί τις παραπάνω απαιτήσεις και μπορεί να χρησιμοποιεί σε αναζήτηση (pattern queries) σε σημασιολογικές τροχιές.

3 Ορισμός δομών επίλυσης προβλήματος

3.1 Ορισμός προβλήματος

Η αναζήτηση σημασιολογικών τροχιών που ικανοποιούν συγκεκριμένα χωροχρονικά και λεκτικά κριτήρια στις υποτροχιές τους, εκφρασμένα στο πλαίσιο των regular expressions, αποτελεί μία μορφή query που ανήκουν στο ευρύτερο πλαίσιο των pattern queries. Όπως έχει περιγραφεί και στην προηγούμενη ενότητα, τα ST²P queries [1] και τα Symbolic Trajectory pattern queries [4] ανήκουν στο ευρύτερο πλαίσιο των pattern queries.

Ειδικότερα, με τον όρο spatiotemporal keyword pattern query στη συνέχεια θα εννοούμε την αναζήτηση συγκεκριμένων σημασιολογικών τροχιών (Semantic Trajectories) που οι υποτροχιές τους και η σημασιολογική πληροφορία που αυτές περιέχουν (Episodes) ικανοποιούν συγκεκριμένα κριτήρια που εκφράζονται με regular expressions. Ως κριτήριο της αναζήτησης των συγκεκριμένων semantic trajectories θα δίνονται δηλαδή συγκεκριμένα χαρακτηριστικά (χωροχρονικά και λεκτικά) από episodes.

Τα ανωτέρω episodes θα παρατίθενται με συγκεκριμένη σειρά υποδηλώνοντας έτσι την συγκεκριμένη χρονική ακολουθία που τα ανωτέρω θα πρέπει να βρίσκονται στα semantic trajectories που θα επιστρέφονται από το ερώτημα ως αποτέλεσμα της αναζήτησης.

Κατά συνέπεια, η αναζήτηση πραγματοποιείται σε μία βάση δεδομένων που περιέχει σημασιολογικές τροχιές με κριτήριο αναζήτησης ένα regular expression από episodes.

Η βάση δεδομένων που χρησιμοποιήθηκε για την αποθήκευση και διαχείριση των δεδομένων των Semantic Trajectories είναι η Neo4j [27]. Η Neo4j αποτελεί μία NoSQL Graph database.

3.2 NoSQL Συστήματα Διαχείρισης Βάσεων Δεδομένων

Τα NoSQL (Not Only SQL) συστήματα και βάσεις δεδομένων χρησιμοποιούν μεθόδους και μοντέλα αποθήκευσης, ανάκτησης και διαχείρισης δεδομένων που δεν βασίζονται στις παραδοσιακές αρχιτεκτονικές των σχεσιακών συστημάτων βάσεων δεδομένων (RDBMS).

Οι NoSQL βάσεις δεδομένων χρησιμοποιούνται στη διαχείριση μεγάλου όγκου δεδομένων (Big Data) και σε real-time web εφαρμογές. Βασίζονται κυρίως σε shared-nothing αρχιτεκτονικές που επιτρέπουν την γρήγορη επέκταση του συστήματος με αύξηση των υπολογιστικών κόμβων χωρίς υπερβολικό κόστος καθώς χρησιμοποιούν κυρίως hardware χαμηλού κόστους (commodity hardware). Τα αποθηκευμένα δεδομένα στις συγκεκριμένες βάσεις για λόγους ταχύτερης πρόσβασης σε αυτά από διαφορετικούς χρήστες αλλά και για ανάκτησης τους σε περίπτωση εσφαλμένης λειτουργίας του συστήματος λόγω κατάρρευσης υπολογιστικών κόμβων του συστήματος, αντιγράφονται και αναπαράγονται σε πολλαπλά αντίγραφα.

Σημαντικά χαρακτηριστικά των NoSQL συστημάτων είναι τα παρακάτω [25]:

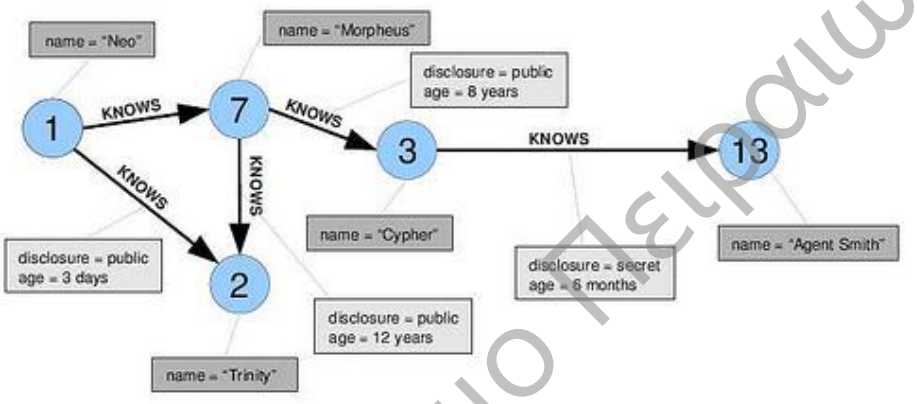
- Το οριζόντιο scaling των δεδομένων, δηλαδή ο διαμοιρασμός τους σε πολλούς κόμβους επιτυγχάνοντας έτσι γρήγορη απόκριση σε απλές αναγνώσεις και εγγραφές μικρού αριθμού δεδομένων.
- Αντιγραφή των δεδομένων σε πολλούς κόμβους.
- Η ύπαρξη μίας μη σύνθετης διεπαφής για την διαχείριση και επεξεργασία των δεδομένων, σε αντίθεση με τις σχεσιακές βάσεις δεδομένων.
- Η μη τήρηση ACID (Atomicity, Consistency, Isolation και Durability) κριτηρίων καθώς σε αρκετές περιπτώσεις το κόστος υλοποίησης τους σε μεγάλη κλίμακα θα ήταν απαγορευτικό ή θα υπήρχε ιδιαίτερη πολυπλοκότητα στην εφαρμογή τους.
- Αποδοτική χρήση των κατανεμημένων ευρετηρίων και της μνήμης για την αποθήκευση και ανάκτηση των δεδομένων. Δυνατότητες χρησιμοποίησης της λανθάνουσας μνήμης των κόμβων για τα δεδομένα που χαρακτηρίζονται από μεγάλη συχνότητα ανάκτησης για επίτευξη καλύτερης απόδοσης.
- Δεν υπάρχει ένα προκαθορισμένο σχήμα της βάσης. Κάθε εγγραφή μπορεί να έχει ένα μεταβλητό αριθμό από πεδία και μπορεί να διαμορφωθεί διαφορετικά ανάλογα με την εφαρμογή που χρησιμοποιεί τη βάση δεδομένων.

Οι βασικότερες κατηγορίες των NoSQL συστημάτων είναι οι παρακάτω [26]:

- Simple key-value store
- Table-oriented NoSQL data stores
- Document-oriented data stores
- Graph databases

3.3 Neo4j Graph Database

Η Neo4j [27] ως graph database χρησιμοποιεί δομές από γράφους με κορυφές και ακμές για την αναπαράσταση και αποθήκευση των δεδομένων. Η Neo4j είναι υλοποιημένη στη γλώσσα προγραμματισμού Java και υπάρχει ως open-source έκδοση αλλά ταυτόχρονα είναι διαθέσιμη και ανάλογη εμπορική έκδοση της για εταιρική χρήση. Χαρακτηριστικό παράδειγμα χρησιμοποίησής της είναι σε δεδομένα κοινωνικών δικτύων.

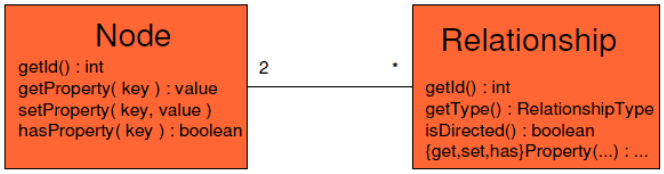


Σχήμα 22: Παράδειγμα αναπαράστασης δεδομένων σε γράφο [28]

Κάθε κορυφή (Node) και ακμή (Relationship) που χρησιμοποιείται για την αποθήκευση των δεδομένων και την αναπαράστασή τους μπορεί να περιέχει έναν μεταβλητό αριθμό από ζεύγη key-value ως properties. Πρόκειται δηλαδή για μία graph database που έχει χαρακτηριστικά της κατηγορίας key-value των No-SQL databases, υποστηρίζοντας ταυτόχρονα και σχέσεις μεταξύ των αντικειμένων των εγγραφών που υπάρχουν στην βάση. Κατά την εκτέλεση των transactions υποστηρίζεται το μοντέλο ACID.

Επιπρόσθετα, επειδή πραγματοποιείται αποθήκευση των δεδομένων στους κόμβους και στις ακμές ως ζεύγη key-value, η database μπορεί να χρησιμοποιηθεί αποτελεσματικά και για την αποθήκευση ημιδομημένων δεδομένων. Δεν υπάρχει προκαθορισμένο schema της βάσης σε αντίθεση με τις σχεσιακές βάσεις δεδομένων.

Για την διαχείριση και επεξεργασία των εγγραφών, υπάρχουν κατάλληλες μέθοδοι που μπορούν να χρησιμοποιηθούν για την ανάκτηση των αντικειμένων και των σχέσεων που υπάρχουν μεταξύ τους, όπως φαίνεται στο αντίστοιχο σχήμα. Υπάρχουν επίσης κατάλληλες μέθοδοι και για την διάσχιση των κορυφών και των ακμών.



Σχήμα 23: Μέθοδοι επεξεργασίας και ανάκτησης δεδομένων από ακμές και κορυφές [28]

Η Neo4j είναι fully transactional (υποστηρίζει δηλαδή JTA/JTS, XA, 2PC, Tx recovery και deadlock detection τεχνικές) σε αντίθεση με άλλες No-SQL βάσεις. Παράλληλα, υπάρχει ειδική έκδοση της βάσης, η Neo4j HA (High Availability) που υποστηρίζει eventual consistency. Ειδικότερα, υποστηρίζονται 0 ή περισσότεροι slaves κόμβοι που ρυθμίζονται και παρακολουθούνται από έναν master κόμβο με την τεχνική Zookeeper όπως στην No-SQL database Cassandra.

Για την επικοινωνία των εφαρμογών και υπηρεσιών με την βάση δεδομένων μπορεί να χρησιμοποιηθεί κατάλληλο REST interface. Η υλοποίηση των επερωτήσεων στην βάση μπορεί να γίνει και με τη χρήση κατάλληλου Java Transaction API. Εκτός από το Java API επιπρόσθετα υποστηρίζεται μεγάλος αριθμός γλωσσών για αποτελεσματική επικοινωνία με τη βάση όπως Python, Jython, Ruby και Clojure.

Η μορφολογία των queries διαφέρει σε σύγκριση με παραδοσιακές σχεσιακές βάσεις δεδομένων καθώς μπορεί να χρησιμοποιηθεί προαιρετικά μία ειδική γλώσσα που ονομάζεται Cypher. Σκοπός των επερωτήσεων είναι η διάσχιση των κορυφών ξεκινώντας από μία συγκεκριμένη κορυφή και διασχίζοντας την να πραγματοποιηθεί ανάκτηση των επιθυμητών δεδομένων χρησιμοποιώντας τις κατάλληλες μεθόδους της χρησιμοποιούμενης διεπαφής. Εκτός από τις πληροφορίες που είναι επιθυμητή η συλλογή τους, απαραίτητος είναι και ο ορισμός ενός κριτηρίου για τον τερματισμό του ερωτήματος και της διάσχισης του γράφου.

Για την πραγματοποίηση των spatiotemporal keyword pattern queries σε σημασιολογικές τροχιές που αναπαριστούνται ως χωρικά αντικείμενα χρησιμοποιήθηκε μία library για τη συγκεκριμένη βάση δεδομένων. Το Neo4j Spatial [28] είναι μία library που δίνει την δυνατότητα αποθήκευσης, διαχείρισης και αναζήτησης χωρικών δεδομένων στη βάση Neo4j. Για την αναπαράσταση των χωρικών αντικειμένων χρησιμοποιείται από το Neo4j Spatial ένα open-source spatial API σε Java, το JTS Topology Suite [30].

Το Neo4j Spatial μπορεί να χρησιμοποιηθεί για διαχείριση και αποθήκευση δεδομένων από χωρικά αντικείμενα, δίνοντας επιπρόσθετα μέσω του JTS Topology Suite, τη δυνατότητα πραγματοποίησης κάποιων βασικών επερωτήσεων, όπως τομή ή επικάλυψη δύο χωρικών αντικειμένων.

Για την πραγματοποίηση spatiotemporal keyword pattern queries σε semantic trajectories πραγματοποιήθηκε επέκταση των κλάσεων του JTS Topology Suite για την διαχείριση και αποθήκευση χρονικής και λεκτικής πληροφορίας και των κλάσεων του Neo4j Spatial προκειμένου να είναι δυνατή η αποθήκευση της χωροχρονικής και λεκτικής πληροφορίας στη βάση δεδομένων Neo4j.

3.4 Βασικοί ορισμοί προβλήματος

Στη συνέχεια περιγράφονται οι βασικοί ορισμοί [1] που θα χρησιμοποιηθούν για τη διαχείριση, αποθήκευση και ανάκτηση των σημασιολογικών (χωροχρονικών και λεκτικών) πληροφοριών από τη βάση δεδομένων, το ευρετήριο και τις υλοποιημένες μεθόδους αναζήτησης.

3.4.1 RawTrajectory

Το RawTrajectory ορίζεται από ένα σύνολο 3 τιμών: (object ID, trajectory ID, 3D polyline). Ο συνδυασμός των τιμών object ID και trajectory ID είναι μοναδικός και ορίζει την ταυτότητα του κινούμενου αντικειμένου και την συγκεκριμένη τροχιά. Η 3D polyline αναφέρεται στην χωροχρονική τροχιά του αντικειμένου, δηλαδή σε ένα χωροχρονικό linestring.

3.4.2 RawSubTrajectory

Το RawSubTrajectory που είναι επίσης RawTrajectory ορίζεται από ένα σύνολο 4 τιμών: (object ID, trajectory ID, subtrajectory ID, 3D polyline). Ο συνδυασμός των τιμών object ID, trajectory ID και subtrajectory ID είναι μοναδικός και ορίζει την ταυτότητα του κινούμενου αντικειμένου και το συγκεκριμένο κομμάτι της τροχιάς του αντικειμένου. Η 3D polyline αναφέρεται στο συγκεκριμένο τμήμα της χωροχρονικής τροχιάς του αντικειμένου, δηλαδή σε ένα χωροχρονικό linestring.

3.4.3 Stop/Move

Ένα RawSubTrajectory χαρακτηρίζεται ως Stop ή ως Move αναλόγως αν ικανοποιούνται συγκεκριμένοι χωροχρονικοί περιορισμοί (για παράδειγμα ελάχιστη χωρική μετατόπιση έτσι ώστε να θεωρείται το αντικείμενο ακίνητο τη συγκεκριμένη χρονική περίοδο) όσον αφορά την εκάστοτε κάθε φορά χρησιμοποίηση της οντολογίας.

3.4.4 Episode

Ένα Episode ορίζεται από ένα σύνολο 5 τιμών: (defineTag, MBB, episodeTag, activityTag, T-link). Η πρώτη τιμή χαρακτηρίζει ένα αντικείμενο ως Stop ή Move, η δεύτερη τιμή αφορά το Minimum Bounding Box (MBB) που περιβάλλει τη συγκεκριμένη χωροχρονική υποτροχιά, η τρίτη και η τέταρτη τιμή αφορούν περαιτέρω σημασιολογικές (λεκτικές) πληροφορίες και η τελευταία τιμή το συγκεκριμένο αντικείμενο RawSubTrajectory που αφορά η κλάση.

3.4.5 SemanticTrajectory

Ένα SemanticTrajectory ορίζεται από ένα σύνολο 3 τιμών: (object ID, SemanticTrajectory ID, T-sem). Η πρώτη τιμή αφορά το ID του αντικειμένου που πραγματοποιεί την τροχιά, η δεύτερη τιμή το ID του semantic trajectory και η τρίτη τιμή είναι η ακολουθία των αντικειμένων episodes σε χρονολογική σειρά που πραγματοποιήθηκαν και ανήκουν στο ίδιο trajectory.

3.5 Υλοποίηση βασικών κλάσεων στη JTS library

Στη συνέχεια ακολουθεί σύντομη περιγραφή της υλοποίησης των ορισμών που περιγράφτηκαν στην προηγούμενη ενότητα επεκτείνοντας βασικές κλάσεις της JTS library. Η τροποποίηση των κλάσεων πραγματοποιήθηκε προκειμένου οι καινούριες κλάσεις να περιέχουν την επιπρόσθετη χρονική και λεκτική διάσταση της πληροφορίας.

3.5.1 Κλάση RawSubTrajectory

Η κλάση RawSubTrajectory αποτελεί κλάση για την αναπαράσταση ενός linestring στο χωροχρόνο και αποτελεί επέκταση της κλάσης LineString της JTS. Η κλάση περιέχει τις χωροχρονικές συντεταγμένες που σχηματίζουν την υποτροχιά του αντικειμένου καθώς και κατάλληλες λεκτικές πληροφορίες σε String: trajectoryID και objectId σύμφωνα με τον ορισμό της οντολογίας.

3.5.2 Κλάση Episode

Η επέκταση της κλάσης RawSubTrajectory με ονομασία Episode περιέχει κατάλληλους κατασκευαστές και μεθόδους για τη δημιουργία του αντικειμένου με επιπρόσθετη χρονική πληροφορία, όπως αναφέρθηκε και στον ορισμό του Episode προηγουμένως. Περιέχει δηλαδή κατάλληλες μεθόδους για τη δημιουργία του Minimum Bounding Box (MBB) που περιβάλλει το αντικείμενο και μεθόδους για τον ορισμό και ανάκτηση των λεκτικών defineTag, episodeTag και activityTag που προσδίδουν στο αντικείμενο λεκτική πληροφορία.

3.5.3 Κλάση MBB

Η κλάση Envelope της JTS αποτελεί κλάση για την αναπαράσταση ενός bounding box στο χώρο. Η επέκταση της κλάσης με ονομασία MBB (Χωροχρονικό Bounding Box) περιέχει κατάλληλους κατασκευαστές και μεθόδους για τη δημιουργία του αντικειμένου με επιπρόσθετη χρονική πληροφορία. Επιπρόσθετα, περιέχει μεθόδους που δημιουργήθηκαν για την υποβοήθηση της χωροχρονικής και λεκτικής αναζήτησης που ελέγχουν εάν ένα MBB επικαλύπτεται χωροχρονικά με ένα άλλο MBB και μέθοδο για την χωροχρονική επέκτασή του έτσι ώστε να περιλαμβάνεται σε αυτό ένα MBB που δίνεται ως όρισμα (χωροχρονικό union).

3.5.4 Κλάση RawTrajectory

Η κλάση χρησιμοποιείται για την αναπαράσταση ενός συνολικού trajectory και αποτελείται από επιμέρους RawSubTrajectory αντικείμενα και κατάλληλες μεθόδους για τον ορισμό και την ανάκτηση των trajectoryID και objectID.

3.5.5 Κλάση SemanticTrajectory

Η κλάση χρησιμοποιείται για την αναπαράσταση ενός συνολικού σημασιολογικού trajectory και αποτελείται από επιμέρους Episode αντικείμενα και κατάλληλες μεθόδους για τον ορισμό και την ανάκτηση των trajectoryID και objectID.

3.6 Αποθήκευση και ανάκτηση σημασιολογικών αντικειμένων

Όπως προαναφέρθηκε, η library Neo4j Spatial για την αναπαράσταση και αποθήκευση των γεωγραφικών αντικειμένων στη Neo4j database περιέχει κατάλληλες κλάσεις για την αποθήκευση στη graph database χωρικής πληροφορίας και ανάκτηση της από αυτή και δημιουργία κατάλληλων JTS χωρικών αντικειμένων. Για την διαχείριση χωροχρονικής και λεκτικής πληροφορίας πραγματοποιήθηκε τροποποίηση των βασικών κλάσεών της library προκειμένου να χρησιμοποιούν τις επεκτάσεις και τροποποιήσεις των κλάσεων που διαχειρίζονται επιπρόσθετα και χρονική-λεκτική πληροφορία.

Η δημιουργία και αποθήκευση σημασιολογικών αντικειμένων στη βάση, απαιτεί τη χρησιμοποίηση κατάλληλης κλάσης που θα ορίζει τον τρόπο αποθήκευσης των χωροχρονικών και λεκτικών πληροφοριών των αντικειμένων στη βάση. Η κλάση αυτή αποτελεί επέκταση της βασικής κλάσης GeometryEncoder και δίνει τη δυνατότητα δημιουργίας custom encoding κλάσεων που ορίζουν τον τρόπο αποθήκευσης των αντικειμένων στη βάση.

Μέσω της συγκεκριμένης κλάσης ορίζεται η μορφή που θα αποθηκευτεί η πληροφορία στον γράφο (encoding), δηλαδή στους κόμβους και στις ακμές που υπάρχουν σε αυτόν.

Επιπρόσθετα, η ανάκτηση της σημασιολογικής πληροφορίας από τη βάση (decoding) και η δημιουργία των κατάλληλων αντικειμένων που αναφέρθηκαν στην προηγούμενη ενότητα ορίζεται από την ίδια κλάση. Ειδικότερα, στο συγκεκριμένο

αντικείμενο ορίζεται το είδος των αντικειμένων που θα δημιουργηθούν μετά την ανάγνωση της πληροφορίας που υπάρχει στη βάση δεδομένων.

Η διαχείριση των αντικειμένων μετά την ανάκτηση τους από τη βάση γίνεται στη συνέχεια μέσω του αντικειμένου LayerST (Layer SpatioTemporal) που δίνει τη δυνατότητα πρόσβασης στο σύνολο των σημασιολογικών αντικειμένων που υπάρχουν στη βάση δεδομένων.

Στη συνέχεια αναφέρονται 2 διαφορετικές encoding κλάσεις που έχουν δημιουργηθεί και αφορούν την αναπαράσταση Episodes και SemanticTrajectories και χρησιμοποιούνται για την αποθήκευση και ανάκτηση των δεδομένων από τις διαφορετικές αλγοριθμικές υλοποιήσεις του spatiotemporal keyword pattern query που θα αναφερθούν στη συνέχεια.

3.6.1 EpisodeEncoder

Η κλάση αναλαμβάνει την αποθήκευση episode αντικειμένων στη Neo4j database δημιουργώντας κατάλληλα node properties και αντιστοιχώντας ένα episode με όλα τα δεδομένα του ανά κόμβο. Κατά την ανάκτηση από τη βάση επιστρέφει episode αντικείμενα.

3.6.2 SemanticTrajectoryEncoder

Η κλάση αναλαμβάνει την αποθήκευση SemanticTrajectory αντικειμένων στη Neo4j database δημιουργώντας κατάλληλα node properties και αντιστοιχώντας ένα episode του SemanticTrajectory με όλα τα δεδομένα του ανά κόμβο. Κατά την ανάκτηση από τη βάση επιστρέφει SemanticTrajectory αντικείμενα.

3.7 Αναπαράσταση και αποθήκευση των semantic trajectories και episodes στη βάση Neo4j

Η αποθήκευση στη βάση δεδομένων Neo4j των χωροχρονικών εγγραφών πραγματοποιείται με την αρχική δημιουργία ενός node (κόμβος) με id 0, ο οποίος έχει directed relationship (ακμή) με id 0 και type «SPATIAL» με ένα δεύτερο node με id 1. Ο δεύτερος κόμβος έχει property με ονομασία type και τιμή «SPATIAL» και

συνδέεται με ένα directed relationship με id 3 και type «LAYER» με έναν κόμβο με id 2, που περιέχει τις πληροφορίες για το spatial layer του Neo4j Spatial.

Ο κόμβος με id 2 περιέχει τα παρακάτω properties:

- «layer_class» με τιμή την ονομασία της κλάσης που χρησιμοποιείται για τη δημιουργία του αντικειμένου LayerST που θα διαχειρίζεται το σύνολο των χωροχρονικών αντικειμένων της βάσης.
- «layer» με τιμή που αφορά την ονομασία του spatiotemporal layer.
- «ctime» με τιμή που αφορά την χρονική στιγμή δημιουργίας της βάσης. Η τιμή είναι σε Long και μπορεί να μετατραπεί σε timestamp μέσω του πακέτου Timestamp της Java.
- «geomencoder_config» με τιμή που αφορά τις παραμέτρους που θα αποθηκευτούν στα geometry nodes σύμφωνα με την custom Encoder κλάση που έχει δημιουργηθεί.
- «geomencoder» με τιμή που αφορά την συγκεκριμένη Encoder κλάση που θα χρησιμοποιηθεί για την αποθήκευση και ανάκτηση των δεδομένων.

Ο συγκεκριμένος κόμβος με id 2 έχει τα παρακάτω directed relationships:

- «GEOMETRIES» με προορισμό τους γεωμετρικούς κόμβους (geometry node) που περιέχουν πληροφορίες για τα χωρικά αντικείμενα.
- «RTREE_ROOT» με προορισμό κόμβους που περιέχουν references (που δηλώνονται με ακμές με type «RTREE_REFERENCE») σε geometry nodes και λειτουργεί ως αρχή του index της βάσης και θα περιγραφεί στην επόμενη ενότητα αναλυτικότερα.
- «RTREE_METADATA» με προορισμό κόμβο που μπορεί να χρησιμοποιηθεί για την αποθήκευση metadata.

Με αρχικό κόμβο με id 2 και προορισμό με directed relationships τους κόμβους που παραθέτονται στη συνέχεια δημιουργούνται geometry nodes για την αποθήκευση της χωροχρονικής πληροφορίας.

Ένας geometry node έχει ως properties το «gtype» και το «bbox» που αντιστοιχούν στο geometry type σύμφωνα με τους OpenGIS geometry type numbers και στο bounding box (τιμή σε double[]) του συγκεκριμένου αντικειμένου geometry. Το bounding box είναι ένας πίνακας double[] που περιέχει τις δύο χωροχρονικές συντεταγμένες του ελάχιστου bounding box που περιβάλλει το αντικείμενο.

Ειδικότερα, ο κάθε geometry κόμβος ενός episode περιέχει κατάλληλες ιδιότητες (node properties) για την αποθήκευση των πληροφοριών ενός episode (σύνολο χωροχρονικών συντεταγμένων και των υπόλοιπων λεκτικών πληροφοριών). Περιέχει δηλαδή πίνακες για την αποθήκευση των χωρικών συντεταγμένων (ιδιότητες «x» και

«y»), των timestamps (ιδιότητα «timestamp») και του MBB (ιδιότητα «bbox») καθώς και κατάλληλα properties για την αποθήκευση των episode ID (ιδιότητα «ID»), object ID, trajectory ID, next episode ID, tag (define tag και activity tag) και episode tag (stop/move). Το σύνολο των episodes που ανήκουν στο ίδιο semantic trajectory συνδέονται μεταξύ τους με directed relationships με type «NEXT».

Στη συνέχεια ακολουθεί περιγραφή ενός παραδείγματος αποθήκευσης στη βάση Neo4j των δεδομένων μίας σημασιολογικής τροχιάς από τα δοκιμαστικά δεδομένα του Hermes Attica data set, που θα περιγραφεί στην Ενότητα 4.1.

Το συγκεκριμένο semantic trajectory (trajectory ID 1 και object ID 1) περιέχει 639 χωροχρονικές συντεταγμένες που αντιστοιχούν σε 5 episodes. Τα episodes έχουν τα παρακάτω δεδομένα:

-Episode με ID 1: 10 συντεταγμένες, episode tag «STOP» και activity/define tag «HOME;RELAXING»
-Episode με ID 2: 321 συντεταγμένες, episode tag «MOVE» και activity/define tag «TRANSPORTATION;WALKING»
-Episode με ID 3: 11 συντεταγμένες, episode tag «STOP» και activity/define tag «SCHOOL;STUDYING»
-Episode με ID 4: 286 συντεταγμένες, episode tag «MOVE» και activity/define tag «TRANSPORTATION;WALKING »
-Episode με ID 5: 11 συντεταγμένες, episode tag «STOP» και activity/define tag «HOME;RELAXING»

Οι κόμβοι που θα δημιουργηθούν από το Neo4j Spatial μαζί με τα αντίστοιχα properties που αυτοί έχουν για την δημιουργία της βάσης και την αναπαράσταση του αντικειμένου είναι οι παρακάτω:

-Κόμβος με id 0, που αποτελεί υποχρεωτικό αρχικό κόμβο σε κάθε βάση Neo4j.

-Κόμβος με id 1 και property «type» με τιμή «spatiotemporal».

-Κόμβος με id 2 και τα properties «layer», «ctime», «geomencoder», «layer_class», και geomencoder_config» και τιμές που έχουν περιγραφεί νωρίτερα.

-Κόμβος με id 3 και τα properties «bbox» και «childIndex».

-Κόμβος με id 4 και τα properties «maxNodeReferences» και «totalGeometryCount».

-Κόμβος με id 5 και τα properties «gtype», «bbox», «x», «y», «timestamp», «objectID», «trajectoryID», «ID», «flag», «tags», «nextID» και τιμές που αντιστοιχούν στις 10 συντεταγμένες του πρώτου episode.

-Κόμβος με id 6 και τα properties «gtype», «bbox», «x», «y», «timestamp», «objectID», «trajectoryID», «ID», «flag», «tags», «nextID» και τιμές που αντιστοιχούν στις 321 συντεταγμένες του δεύτερου episode.

-Κόμβος με id 7 και τα properties «gtype», «bbox», «x», «y», «timestamp», «objectID», «trajectoryID», «ID», «flag», «tags», «nextID» και τιμές που αντιστοιχούν στις 11 συντεταγμένες του τρίτου episode.

-Κόμβος με id 8 και τα properties «gtype», «bbox», «x», «y», «timestamp», «objectID», «trajectoryID», «ID», «flag», «tags», «nextID» και τιμές που αντιστοιχούν στις 286 συντεταγμένες του τέταρτου episode.

-Κόμβος με id 9 και τα properties «gtype», «bbox», «x», «y», «timestamp», «objectID», «trajectoryID», «ID», «flag», «tags», «nextID» και τιμές που αντιστοιχούν στις 11 συντεταγμένες του πέμπτου episode.

Οι ακμές που θα δημιουργηθούν από το Neo4j Spatial μαζί με τα αντίστοιχα properties που αυτές έχουν για την δημιουργία της βάσης και την αναπαράσταση του αντικειμένου είναι οι παρακάτω:

-Relationship με id 0, αρχή τον κόμβο με id 0, τέλος τον κόμβο με id 1 και type «SPATIAL».

-Relationship με id 1, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 3 και type «RTREE_ROOT».

-Relationship με id 2, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 4 και type «RTREE_METADATA».

-Relationship με id 3, αρχή τον κόμβο με id 1, τέλος τον κόμβο με id 2 και type «LAYER».

-Relationship με id 4, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 5 και type «GEOMETRIES».

-Relationship με id 5, αρχή τον κόμβο με id 3, τέλος τον κόμβο με id 5 και type «RTREE_REFERENCE».

-Relationship με id 6, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 6 και type «GEOMETRIES».

-Relationship με id 7, αρχή τον κόμβο με id 3, τέλος τον κόμβο με id 6 και type «RTREE_REFERENCE».

-Relationship με id 8, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 7 και type «GEOMETRIES».

-Relationship με id 9, αρχή τον κόμβο με id 3, τέλος τον κόμβο με id 7 και type «RTREE_REFERENCE».

-Relationship με id 10, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 8 και type «GEOMETRIES».

-Relationship με id 11, αρχή τον κόμβο με id 3, τέλος τον κόμβο με id 8 και type «RTREE_REFERENCE».

-Relationship με id 12, αρχή τον κόμβο με id 2, τέλος τον κόμβο με id 9 και type «GEOMETRIES».

-Relationship με id 13, αρχή τον κόμβο με id 3, τέλος τον κόμβο με id 9 και type «RTREE_REFERENCE».

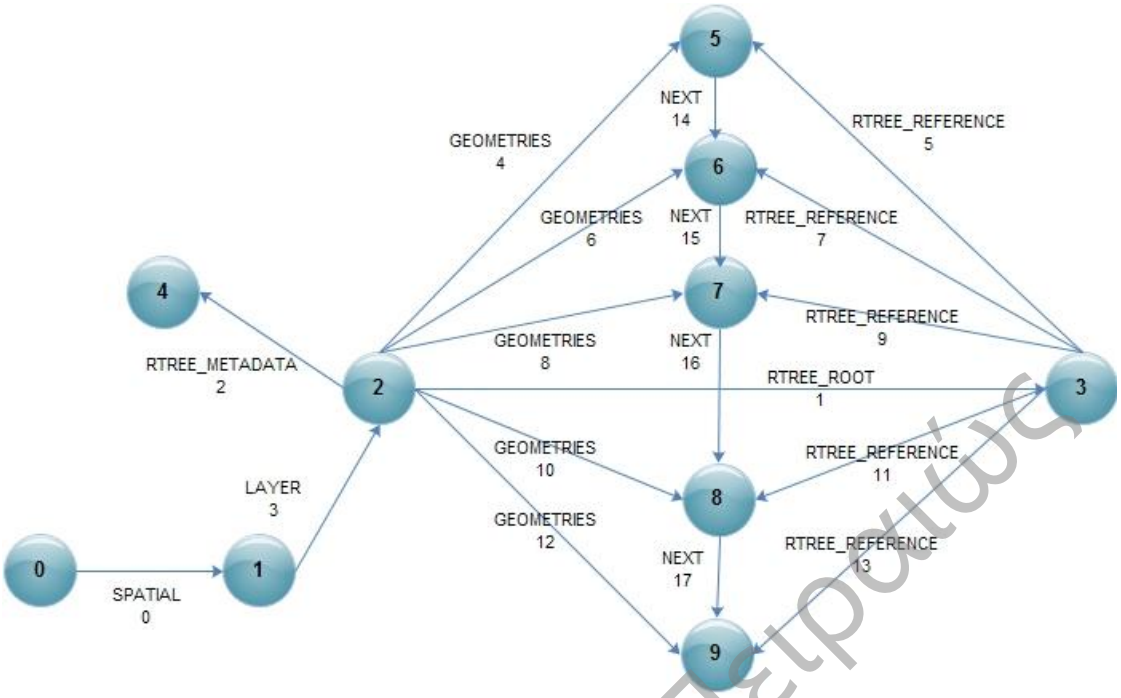
-Relationship με id 14, αρχή τον κόμβο με id 5, τέλος τον κόμβο με id 6 και type «NEXT».

-Relationship με id 15, αρχή τον κόμβο με id 6, τέλος τον κόμβο με id 7 και type «NEXT».

-Relationship με id 16, αρχή τον κόμβο με id 7, τέλος τον κόμβο με id 8 και type «NEXT».

-Relationship με id 17, αρχή τον κόμβο με id 8, τέλος τον κόμβο με id 9 και type «NEXT».

Στο Σχήμα 24 που ακολουθεί στη συνέχεια γίνεται αναπαράσταση του συνόλου των ακμών και των κορυφών που έχουν δημιουργηθεί στο Neo4j Spatial όπως έχουν περιγραφεί παραπάνω. Μέσα στον κάθε κόμβο αναγράφεται το id του, ενώ σε κάθε ακμή αναγράφεται το type της και το id της. Αντίστοιχα σε περίπτωση προσθήκης δεύτερου Semantic Trajectory στη βάση, θα δημιουργηθούν καινούριοι geometry κόμβοι που θα αποθηκευτεί η πληροφορία και αντίστοιχα Relationships τύπου «GEOMETRIES» και «RTREE_REFERENCE». Με την προσθήκη μεγαλύτερου αριθμού από Semantic Trajectories το δέντρο που σχηματίζεται από τα Relationships «RTREE_REFERENCE» και με ρίζα τον κόμβο με id 3 θα γίνει πολυεπίπεδο, όπως θα αναλυθεί στην επόμενη ενότητα.



Σχήμα 24: Αναπαράσταση της πληροφορίας στο Neo4j Spatial στην περίπτωση που η βάση δεδομένων περιέχει ένα semantic trajectory

3.8 Χρονική και λεκτική επέκταση του Neo4j Spatial Index

Η library Neo4j Spatial χρησιμοποιεί ένα R-tree index που υπάρχει υλοποιημένο στη Neo4j (πακέτο org.neo4j.collections.rtree). Όπως αναφέρθηκε και νωρίτερα, για την αποθήκευση ομοειδών αντικειμένων geometry χρησιμοποιείται η κλάση Layer. Μέσω της συγκεκριμένης κλάσης τα γεωμετρικά αντικείμενα που αποθηκεύονται σε ένα Layer αποκτούν ταυτόχρονα και ένα R-tree index για την ανάκτηση τους στη συνέχεια. Το R-Tree index που δημιουργείται δίνει τη δυνατότητα εκτέλεσης χωρικών queries στα γεωγραφικά αντικείμενα.

Για την πραγματοποίηση των spatiotemporal keyword pattern queries σε semantic trajectories, έγινε επέκταση και τροποποίηση του ανωτέρω index προκειμένου να αποθηκεύεται σε αυτό και η χρονική και λεκτική πληροφορία (Hybrid 3D R-Tree). Ειδικότερα, το ευρετήριο που δημιουργήθηκε έχει υβριδική μορφή καθώς αποθηκεύεται σε αυτό χωροχρονική και λεκτική πληροφορία. Το ευρετήριο ακολουθεί το πλαίσιο των hybrid indexes IR-Tree index [16] και KR*-Tree [12] που

αποθηκεύουν χωρική και λεκτική πληροφορία. Στη συνέχεια ακολουθούν διαφορετικές μορφές του Hybrid 3D R-Tree που έχουν δημιουργηθεί.

3.8.1 Episode-based Hybrid 3D R-Tree (EB Hybrid 3D R-Tree)

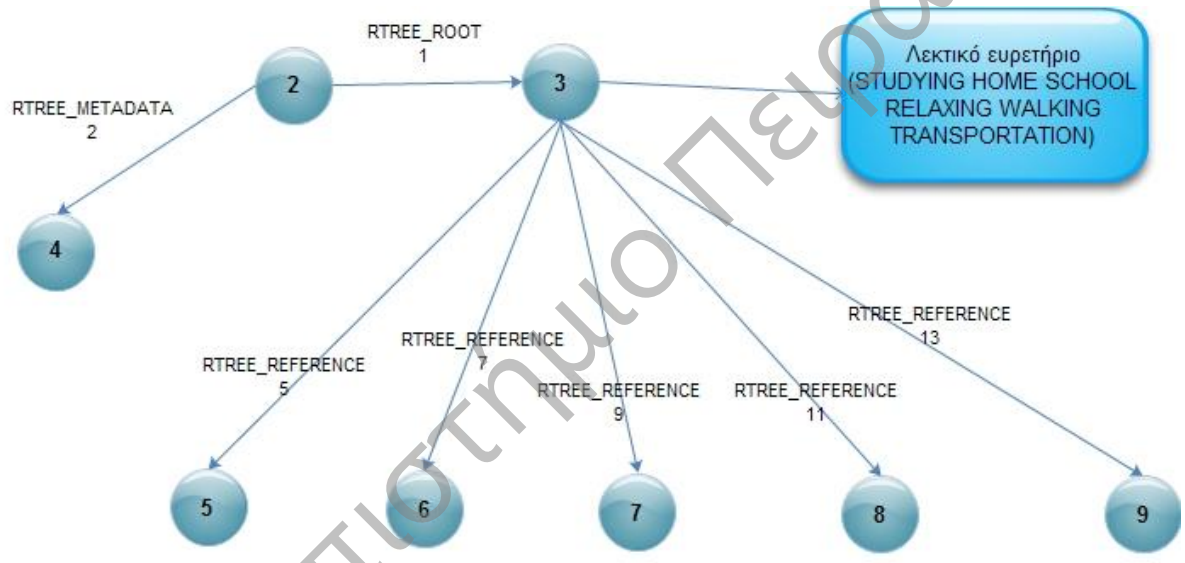
Κατά τη δημιουργία των σημασιολογικών αντικειμένων και την αποθήκευση τους στη βάση δημιουργείται όπως προαναφέρθηκε κόμβος που περιέχει ένα directed relationship με type «RTREE_ROOT» προς κόμβο που αποτελεί ουσιαστικά την ρίζα του 3D R-Tree. Από τον συγκεκριμένο κόμβο υπάρχουν directed relationships με type «RTREE_CHILD» προς κόμβους που χρησιμοποιούνται για την ομαδοποίηση των references προς τους geometry nodes και αποτελούν τα αρχικά «κλαδιά» του δέντρου από την ρίζα. Από κάθε ένα κόμβο που καταλήγει η relationship «RTREE_CHILD» υπάρχουν directed relationships με type «RTREE_REFERENCE» προς τους geometry nodes. Στους γεωμετρικούς κόμβους όπως αναφέρθηκε νωρίτερα καταλήγουν και οι ακμές με type «GEOMETRIES».

Επομένως, ξεκινώντας από τη ρίζα του δέντρου η ομαδοποίηση πρώτα γίνεται μέσω των κόμβων των «RTREE_CHILD» relationships, οι οποίοι στη συνέχεια δείχνουν μέσω των «RTREE_REFERENCE» relationships στους geometry nodes. Σε περίπτωση που υπάρχει μικρός αριθμός από geometry nodes στη βάση (καθορίζεται από την ιδιότητα «maxNodeReferences» του κόμβου με id 4), τότε από τη ρίζα του δέντρου υπάρχουν μόνο «RTREE_REFERENCE» relationships, όπως στο Σχήμα 24 και 25. Αντίστοιχα, σε περίπτωση μεγαλύτερου αριθμού δεδομένων τότε το δέντρο γίνεται πολυεπίπεδο και οι κόμβοι μέσω των «RTREE_CHILD» relationships δείχνουν σε κόμβους που επίσης περιέχουν «RTREE_CHILD» relationships. Στο επίπεδο των φύλλων του δέντρου και σε αυτήν την περίπτωση υπάρχουν «RTREE_REFERENCE» relationships.

Κάθε κόμβος που καταλήγει από τη ρίζα ένα relationship «RTREE_CHILD» περιλαμβάνει όπως αναφέρθηκε «RTREE_REFERENCE» relationships σε geometry nodes ή «RTREE_CHILD» relationships. Και στις δύο περιπτώσεις οι κόμβοι περιέχουν την ιδιότητα «bbox» που αποτελεί το χωροχρονικό bounding box που περιβάλλει κάθε αντικείμενο ή τα αντικείμενα των παιδιών-κόμβων. Ειδικότερα, οι κόμβοι που καταλήγει ένα relationship «RTREE_CHILD» περιέχουν επίσης την ίδια ιδιότητα «bbox» όπως οι geometry nodes που αντιστοιχεί σε ένα χωροχρονικό bounding box που περιβάλλει τα bounding boxes των nodes που αυτός δείχνει μέσω των «RTREE_REFERENCE» ή RTREE_CHILD relationships.

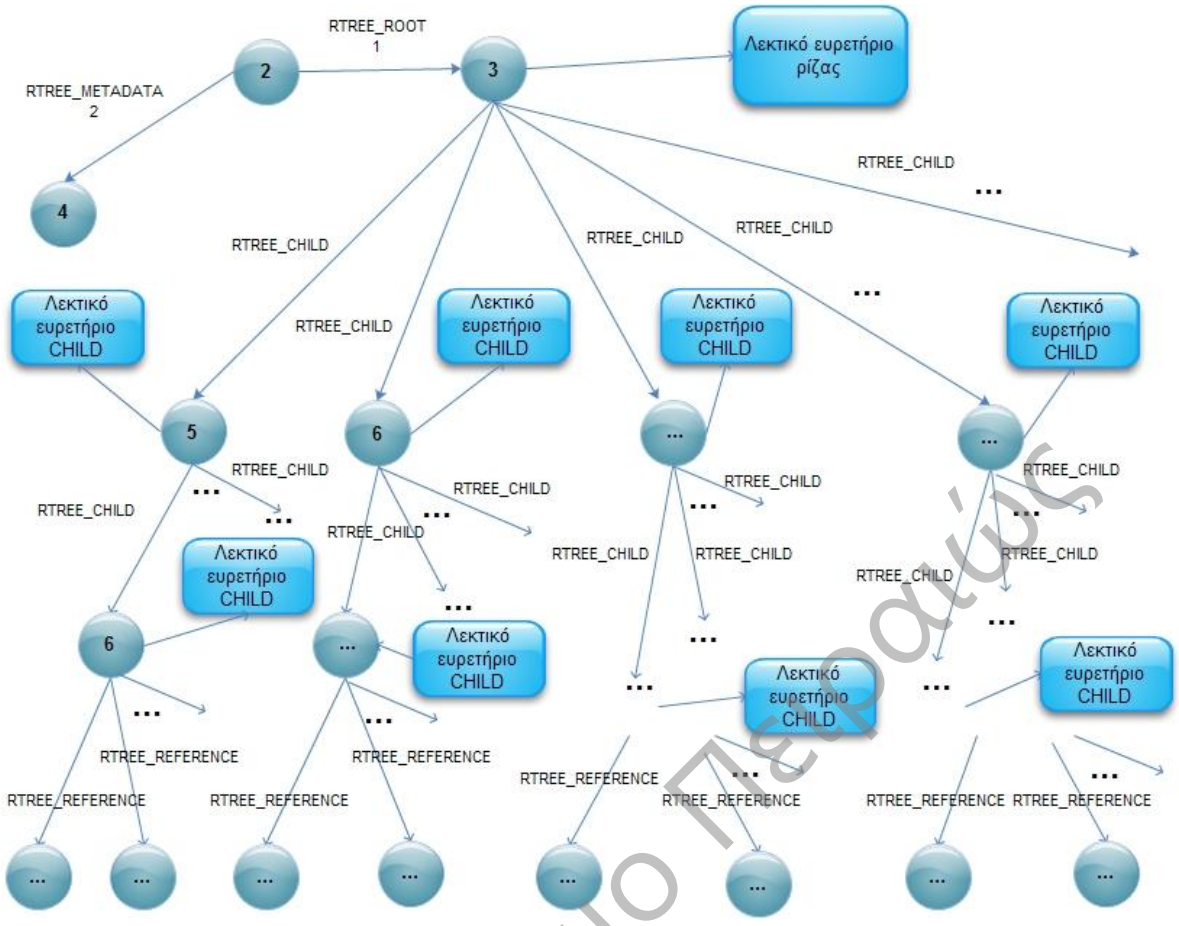
Εκτός από την ιδιότητα «bbox» κάθε geometry node έχει επίσης την ιδιότητα «tags» που περιέχει λεκτικά δεδομένα για το συγκεκριμένο αντικείμενο. Αντίστοιχα, για τον κόμβο που καταλήγει από την ρίζα ένα relationship «RTREE_CHILD» υπάρχει ένα λεκτικό ευρετήριο (ανεστραμμένο αρχείο) που περιέχει το σύνολο των tags των geometry nodes που αυτός δείχνει, σε αναλογία με το IR-Tree (Σχήμα 18). Το όνομα του ευρετηρίου βρίσκεται σε κατάλληλο property με όνομα «childIndex».

Με τον ίδιο τρόπο, η ρίζα του δέντρου περιέχει αντίστοιχα την ιδιότητα «bbox» και κατάλληλο λεκτικό ευρετήριο. Τα ανωτέρω περιέχουν την πληροφορία που υπάρχει στους nodes που αυτή δείχνει μέσω των «RTREE_CHILD» relationships. Στο Σχήμα 25 που ακολουθεί στη συνέχεια γίνεται αναπαράσταση του index για το παράδειγμα που αναφέρθηκε προηγουμένως από το Hermes Attica data set (που αναλύεται στην Ενότητα 4.1) που περιέχεται στη βάση δεδομένων ένα Semantic Trajectory.



Σχήμα 25: Παράδειγμα αναπαράστασης ενός semantic trajectory στο Hybrid 3D R-Tree index

Στο παρακάτω σχήμα ακολουθεί μία γενικότερη μορφή του Hybrid 3D R-Tree index.



Σχήμα 26: Γενική μορφή του Hybrid 3D R-Tree index

3.8.2 Semantic-trajectory-based Hybrid 3D R-Tree (STB Hybrid 3D R-Tree)

Το δέντρο ακολουθεί τη φιλοσοφία και τη δομή του ευρετηρίου της προηγούμενης ενότητας με τη διαφορά ότι το Hybrid 3D R-Tree σε αυτήν την περίπτωση δεν είναι κατασκευασμένο λαμβάνοντας υπόψη ως «δομικό στοιχείο» τα episodes των semantic trajectories, αλλά τα ίδια τα semantic trajectories ως ατομική μονάδα.

Κάθε semantic trajectory που αποθηκεύεται στη βάση έχει και σε αυτόν τον αλγόριθμο μία διαδοχική σειρά από κόμβους που αντιστοιχούν στα episodes που αυτό αποτελείται και συνδέονται μεταξύ τους με «NEXT» relationships. Οι ιδιότητες που έχουν οι κόμβοι είναι ίδιες όπως και στην περίπτωση του Episode-based Hybrid 3D R-Tree. Εκτός από τους παραπάνω κόμβους του semantic trajectory υπάρχει ένας επιπλέον αρχικός κόμβος. Από τον συγκεκριμένο κόμβο υπάρχει μία relationship τύπου «FIRST» που καταλήγει στο πρώτο episode του semantic

trajectory. Ο ανωτέρω αρχικός κόμβος περιέχει πληροφορίες για το σύνολο της σημασιολογικής τροχιάς και περιλαμβάνει τις παρακάτω ιδιότητες:

- «gtype» με τιμή που αντιστοιχεί στο geometry type σύμφωνα με τους OpenGIS geometry type numbers.
- «bbox» με τιμή που αντιστοιχεί στο bounding box όλου του semantic trajectory.
- «objectID» με τιμή που αφορά το object ID του αντικείμενου που πραγματοποίησε την σημασιολογική τροχιά.
- «trajectoryID» με τιμή που αφορά το trajectory ID της σημασιολογικής τροχιάς.
- «tagsArray» με τιμή που αφορά τη συνολική λεκτική πληροφορία του semantic trajectory σε κατάλληλο πίνακα. Η σειρά των tags των episodes στον πίνακα αντιστοιχεί στη χρονική σειρά που βρίσκονται τα tags στα επιμέρους episodes.

Ως αρχικό «δομικό» στοιχείο ευρετηρίασης χρησιμοποιείται δηλαδή η πληροφορία του ανωτέρω κόμβου. Με τις πληροφορίες που υπάρχουν στον συγκεκριμένο κόμβο δημιουργείται ένα R-Tree που έχει σημαντικά μικρότερο μέγεθος σε σύγκριση με το ευρετήριο της προηγούμενης ενότητας. Και σε αυτήν την περίπτωση δημιουργούνται κατάλληλα λεκτικά ευρετήρια για την ρίζα και τα «RTREE_CHILD» υποεπίπεδα του δέντρου.

3.8.3 Enhanced Semantic-trajectory-based Hybrid 3D R-Tree (ESTB Hybrid 3D R-Tree)

Το συγκεκριμένο ευρετήριο ακολουθεί τη φιλοσοφία και τη δομή του Semantic-trajectory-based Hybrid 3D R-Tree με τη διαφορά ότι περιέχει επιπρόσθετη ατομική χωροχρονική και λεκτική πληροφορία για κάθε tag που υπάρχει σε ένα episode ενός trajectory. Ειδικότερα, κάθε κόμβος του δέντρου περιέχει το σύνολο των tags που υπάρχουν στα παιδιά του αποθηκευμένα σε ξεχωριστά properties για κάθε tag.

Σε κάθε property-tag γίνεται αντιστοίχιση του μέγιστου MBB που το περιβάλλει. Το MBB που το περιβάλλει αντιστοιχεί στο χωροχρονικό union των MBB του συγκεκριμένου tag για όσα παιδιά του κόμβου το περιέχουν.

Για παράδειγμα ας υποθέσουμε ότι ορισμένα φύλλα έχουν το tag «BANK» για ένα episode του trajectory που αυτά αντιστοιχούν. Τα φύλλα αυτά ανήκουν σε ένα υποδέντρο που σχηματίζεται από ένα κόμβο του ανώτερου επιπέδου του δέντρου που δείχνει σε αυτούς. Στο συγκεκριμένο κόμβο του ανώτερου επιπέδου θα υπάρχει

το tag «BANK» με καταχωρημένο ένα MBB που θα αντιστοιχεί στο χωροχρονικό union των MBB των φύλλων που περιέχουν το episode με το συγκεκριμένο tag.

Αντίστοιχα θα υπάρχει ένα διαφορετικό MBB για παράδειγμα για το tag «CAFE» αποθηκευμένο σε κατάλληλο property με ονομασία «CAFE» που θα αφορά διαφορετικό χωροχρονικό συμβάν στη σημασιολογική τροχιά.

3.9 Λεκτικό ευρετήριο στο Neo4j

Για την διαχείριση και ανάκτηση αλφαριθμητικών δεδομένων (για παράδειγμα ονομασία ενός χωρικού αντικειμένου) δημιουργήθηκαν για κάθε κόμβο του ευρετηρίου λεκτικά ευρετήρια, που παρέχουν τη δυνατότητα αναζήτησης λαμβάνοντας υπόψη και αλφαριθμητικά δεδομένα. Η βάση Neo4j δίνει τη δυνατότητα δημιουργίας index σε αντικείμενα node και relationship μέσω της βιβλιοθήκης της Lucene. Μέσω του Neo4j index API μπορεί να γίνει add, remove και update ενός αντικειμένου της βάσης ή properties αυτού στο λεκτικό index.

Η αναζήτηση χρησιμοποιώντας ένα λεκτικό ευρετήριο που υπάρχει στο Neo4j μπορεί να πραγματοποιηθεί με δύο τρόπους: μέσω της μεθόδου get() και της μεθόδου query(). Η πρώτη μέθοδος χρησιμοποιείται για την ακριβή αναζήτηση αντικειμένων (exact matches), ενώ η δεύτερη παραμετροποιήσιμη μέθοδος δίνει τη δυνατότητα μέσω της Lucene Query Syntax να δημιουργηθούν παραμετροποιήσιμα queries (για παράδειγμα « like “*example*” »). Μέσω της τελευταίας μεθόδου για κάθε αποτέλεσμα μπορεί να επιστραφεί και σχετική βαθμολογία σχετικότητας (score) μέσω της κλάσης IndexHits.

Εκτός από το βασικό Index μέσω των κατάλληλων κλάσεων μπορούν να δημιουργηθούν στο Neo4j τα παρακάτω indexes:

- Fulltext index (Lucene fulltext index για αδόμητα κείμενα)
- Numeric Index (για αριθμητικά δεδομένα)
- Automatic Index (δυνατότητα ενεργοποίησης και απενεργοποίησης αυτόματου index στη βάση πριν ή μετά την εισαγωγή δεδομένων σε αυτήν)

3.9.1 Apache Lucene

Η δημιουργία του λεκτικού ευρετηρίου γίνεται μέσω της library Lucene [32]. Η Lucene είναι μία open-source βιβλιοθήκη που παρέχει εργαλεία για την δημιουργία μίας μηχανής αναζήτησης (indexing, searching και retrieval). Η λειτουργία της βασίζεται στη δημιουργία ανεστραμμένων ευρετηρίων για την αναζήτηση και ανάκτηση πληροφορίας σε μία συλλογή από κείμενα. Ένα ανεστραμμένο ευρετήριο αντιστοιχεί για κάθε όρο που υπάρχει στα κείμενα της συλλογής μία λίστα με τα κείμενα της συλλογής που αυτός υπάρχει.

3.9.2 Ανάκτηση Πληροφορίας (Information Retrieval)

Στις επόμενες υποενότητες, ακολουθεί η απαραίτητη θεωρητική θεμελίωση σχετικά με την ανάκτηση πληροφορίας για την κατανόηση του τρόπου λειτουργίας της Lucene.

3.9.2.1 Τεχνικές αναπαράστασης

Οι τεχνικές ανάκτησης πληροφορίας [33] που χρησιμοποιούνται στη Lucene είναι οι παρακάτω:

- Δειγματοποίηση (Tokenization): Είναι η διαδικασία που ακολουθείται για τον διαχωρισμό των κειμένων της συλλογής σε κομμάτια που ονομάζονται tokens, αφαιρώντας συγκεκριμένους χαρακτήρες, όπως για παράδειγμα τα σημεία στίξης.
- Λέξεις αποκλεισμού (Stop Words): Αφαίρεση των λέξεων που δεν προσδίδουν περιεχόμενο στο κείμενο, όπως άρθρα, αντωνυμίες, προθέσεις κτλ.
- Κανονικοποίηση (Normalization): Καθορισμός εννοιολογικής ισοδυναμίας μεταξύ όρων που συναντούνται στα κείμενα (για παράδειγμα ίδια αναζήτηση για τον όρο color και color).
- Στελέχωση κειμένου (Stemming): Οι λέξεις των κειμένων υποβιβάζονται στη ρίζα τους με στόχο την ανεξαρτησία τους από τις μορφολογικές παραλλαγές τους.
- Ευρετηρίαση (Indexing): Η διαδικασία που ακολουθείται για την δημιουργία του ευρετηρίου.

3.9.2.2 Αλγοριθμική διαδικασία δημιουργίας του ανεστραμμένου ευρετηρίου

Η διαδικασία επομένως που ακολουθείται για την κατασκευή του ανεστραμμένου ευρετηρίου σε περιγραφική γλώσσα είναι η παρακάτω:

- Εισαγωγή στο σύστημα των εγγράφων που θα γίνουν indexed.
- Tokenization του κειμένου.
- Γλωσσολογική επεξεργασία των tokens.
- Δημιουργία του ανεστραμμένου ευρετηρίου, με δημιουργία για κάθε όρο της λίστας των κειμένων που αυτός περιέχεται.

3.9.2.3 Μοντέλα ανάκτησης πληροφορίας

Τα μοντέλα ανάκτησης πληροφορίας [33] που χρησιμοποιούνται κυρίως στη Lucene είναι το δυαδικό (Boolean) μοντέλο και το διανυσματικό (Vector Space) μοντέλο. Επιπρόσθετες υλοποιήσεις δίνουν τη δυνατότητα επέκτασης της βιβλιοθήκης με άλλα μοντέλα, όπως το πιθανολογικό (Probalistic) μοντέλο.

3.9.2.4 Δυαδικό (Boolean) μοντέλο

Το δυαδικό μοντέλο βασίζεται στη θεωρία συνόλων και χρησιμοποιεί για την αναζήτηση και ανάκτηση κειμένων τους λογικούς τελεστές AND, OR και NOT. Για να γίνει αναζήτηση ενός εγγράφου επομένως πρέπει να ικανοποιείται πλήρως η λογική συνθήκη που δίνεται ως είσοδο και περιέχει τους παραπάνω τελεστές σε συνδυασμό με ζητούμενους όρους από το κείμενο. Κατά συνέπεια, ένα έγγραφο της συλλογής είτε ικανοποιεί το ερώτημα, είτε όχι, δεν υπάρχει δηλαδή μερική ικανοποίηση.

3.9.2.5 Διανυσματικό (Vector Space) μοντέλο

Το διανυσματικό μοντέλο βασίζεται στο μετασχηματισμό των κειμένων σε διανύσματα με συντεταγμένες πραγματικούς αριθμούς χρησιμοποιώντας την έννοια του διανυσματικού χώρου από τη γραμμική άλγεβρα. Κάθε έγγραφο αναπαριστάται ως ένα διάνυσμα με m διαστάσεις, όπου m αντιστοιχεί στον αριθμό των μοναδικών όρων που συναντούνται σε όλα τα έγγραφα της συλλογής. Κάθε έγγραφο και ερώτημα αναπαριστάται δηλαδή με συντεταγμένες w_{ij}, όπου i αντιστοιχεί στον όρο i και έγγραφο j. Τα βάρη που ανατίθενται στους όρους του ευρετηρίου δεν είναι σε αυτήν την περίπτωση δυαδικά (όπως στο δυαδικό μοντέλο) και χρησιμοποιούνται για τον υπολογισμό του βαθμού ομοιότητας μεταξύ του ερωτήματος και του αποθηκευμένου εγγράφου. Σε αυτό το μοντέλο δηλαδή λαμβάνονται υπόψη και τα

έγγραφα που δεν ικανοποιούν πλήρως τις συνθήκες του ερωτήματος και είναι δυνατή η διάταξη των κειμένων της απάντησης σε φθίνουσα σειρά ως προς την σχετικότητα τους με το ερώτημα.

3.9.2.6 Υπολογισμός σημαντικότητας όρων

Ο υπολογισμός της σημαντικότητας ενός όρου του κειμένου (βάρος του όρου) πραγματοποιείται από τη Lucene με τη χρήση της συχνότητας tf-idf, που προκύπτει από τις παρακάτω ορολογίες:

- Συχνότητα εμφάνισης (term frequency) ενός όρου - tf: (αριθμός εμφανίσεων της λέξης κλειδί (όρου) στο συγκεκριμένο έγγραφο) / (συνολικό αριθμό όλων των όρων που περιέχονται στο συγκεκριμένο έγγραφο).
- Αντίστροφη συχνότητα εγγράφου (inverse document frequency) ενός όρου– idf: log [(πλήθος εγγράφων της συλλογής) / (πλήθος εγγράφων που περιέχουν τον συγκεκριμένο όρο)] + 1. Η συγκεκριμένη συχνότητα παίρνει μικρές τιμές για τις λέξεις κλειδιά που εμφανίζονται συχνά και μεγαλύτερες τιμές για λέξεις κλειδιά που εμφανίζονται σπάνια.
- Συντελεστής tf-idf: το γινόμενο των δύο παραπάνω συχνοτήτων.

3.9.2.7 Υπολογισμός ομοιότητας εγγράφων και ερωτημάτων

Στο διανυσματικό μοντέλο ο υπολογισμός της ομοιότητας μεταξύ ερωτημάτων και εγγράφων της συλλογής πραγματοποιείται με την αναπαράσταση τους σε διανύσματα κειμένου και στη συνέχεια υπολογισμού της συνάφειας τους με έναν από τους παρακάτω τρόπους:

- Ευκλείδεια απόσταση των διανυσμάτων
- Εσωτερικό γινόμενο των διανυσμάτων
- Συνημίτονο της γωνίας των δύο διανυσμάτων

Στη Lucene χρησιμοποιείται ο εξής συνδυαστικός τρόπος για την εύρεση του βαθμού ομοιότητας:

(Εσωτερικό γινόμενο των διανυσμάτων) / (Ευκλείδεια απόσταση των διανυσμάτων)

Η βιβλιοθήκη περιέχει επίσης τη δυνατότητα βελτίωσης της βαθμολογίας που προκύπτει από το διανυσματικό μοντέλο, με την προσθήκη λειτουργιών ορισμού custom βαθμού σημαντικότητας συγκεκριμένων όρων και εγγράφων.

3.9.2.8 Αλγοριθμική διαδικασία δημιουργίας ευρετηρίου και αναζήτησης στη Lucene

Συμπερασματικά, ο αλγόριθμος που ακολουθείται για την ευρετηρίαση και αναζήτηση όρων στη Lucene σε περιγραφική γλώσσα είναι ο παρακάτω:

- Εισαγωγή στο σύστημα των εγγράφων που θα γίνουν indexed
- Tokenization του κειμένου
- Γλωσσολογική επεξεργασία των tokens (stemming κτλ)
- Δημιουργία του ανεστραμμένου ευρετηρίου και υπολογισμός των ανεστραμμένων συχνοτήτων
- Εισαγωγή του ερωτήματος αναζήτησης
- Χρησιμοποίηση του δυαδικού μοντέλου
- Χρησιμοποίηση του διανυσματικού μοντέλου
- Υπολογισμός του βαθμού ομοιότητας του ερωτήματος με τα έγγραφα της συλλογής
- Κατάταξη των εγγράφων ανάλογα με το βαθμό ομοιότητας
- Επιστροφή των αποτελεσμάτων

3.9.3 Περίπτωση χρήσης της Lucene

Στη συνέχεια ακολουθεί μία απλή περίπτωση χρήσης [34] της βιβλιοθήκης. Υποθέτουμε αρχικά ότι έχουμε τα παρακάτω 5 έγγραφα στη συλλογή:

1. My sister is coming for the holidays.
2. The holidays are a chance for family meeting.
3. Who did your sister meet?
4. It takes an hour to make fudge.
5. My sister makes awesome fudge.

Η ευρετηρίαση θα έχει ως αποτέλεσμα την εξαγωγή των όρων από κάθε έγγραφο και την δημιουργία των λιστών που χρειάζονται για τη δημιουργία του ευρετηρίου. Η δημιουργία των λιστών στο ευρετήριο θα έχει την παρακάτω μορφή για τους όρους «My» και «fudge»:

- My --> 1,5
- fudge --> 4,5

Αν υποθέσουμε ότι το ερώτημα αποτελείται από τους παραπάνω δύο όρους [my,fudge] τότε για να βρούμε την ομοιότητα μεταξύ των εγγράφων της συλλογής θα πρέπει να γίνει η ίδια διαδικασία που ακολουθήθηκε και στα έγγραφα της συλλογής, δηλαδή εξαγωγή των όρων που αποτελείται το ερώτημα: «my» και «fudge».

Η πλήρης λίστα των εγγράφων που περιέχουν τους όρους του ερωτήματος και έχουν μεγαλύτερο βαθμό ομοιότητας με το ερώτημα είναι τα [1,4,5]. Επειδή όμως το έγγραφο 5 περιέχει και τους δύο όρους του ερωτήματος έχει μεγαλύτερη ομοιότητα από τα υπόλοιπα δύο έγγραφα, άρα μία πιθανή κατάταξη των εγγράφων θα είναι 5, 1 και 4.

3.9.4 Lucene Query Syntax

Στη συνέχεια ακολουθεί κατάλληλος πίνακας που περιέχει τη μορφολογία σύνταξης ενός query στη Lucene.

Type	*Syntax*	*Description*
Token	`t`	Match token `t`
Phrase	`"cs"`	Match tokens in `cs` in exact order without gaps
Field	`f:Q`	Match query *Q* in field *f*
Wildcard, Char	`cs1?cs2`	Match tokens starting with `cs1`, ending with `cs2`, with any char between
Wildcard, Seq	`cs1*cs2`	Match tokens starting with `cs1`, ending with `cs2`, with any char sequence between
Fuzzy	`t~`	Match token `t` approximately
Fuzzy, Weighted	`t~d`	Match token `t` within minimum similarity *d*
Proximity	`P~n`	Match tokens in phrase *P* within distance *n*
Range, Inclusive	`f:[t1 TO t2]`	Match tokens lexicographically between tokens `t1` and `t2` inclusive
Range, Exclusive	`f:(t1 TO t2)`	Match tokens lexicographically between tokens `t1` and `t2` exclusive
Boosting	`P^d`	Match phrase *P*, boosting score by *d*
Disjunction	`Q1 OR Q2`	Match query *Q1* or query *Q2* (or both)
Conjunction	`Q1 AND Q2`	Match query *Q1* and match query *Q2*
Difference	`Q1 NOT Q2`	Match query *Q1* but not query *Q2*
Must	`+P`	Token or phrase *P* must appear
Mustn't	`-P`	Token or phrase *P* must not appear
Grouping	`(Q)`	Match query *Q* (disambiguates parsing)

Σχήμα 27: Lucene Query Syntax [35]

3.9.5 Βασικές κλάσεις της Lucene

Στο συνέχεια ακολουθεί αναλυτική περιγραφή των βασικών κλάσεων [36] της Lucene.

3.9.5.1 Indexing κλάσεις της Lucene

3.9.5.1.1 Κλάση Field

Ένα αντικείμενο Document αποτελείται από μία συλλογή από fields. Η αναζήτηση και η ευρετηρίαση πραγματοποιείται πάνω στην πληροφορία που υπάρχει στα fields. Για τη δημιουργία ενός αντικειμένου field απαιτούνται η ονομασία του (name), η πληροφορία που θα αποθηκευτεί σε αυτό (value) καθώς και ένα set από flags που προσδιορίζουν τον τρόπο που θα αποθηκευτεί στο ευρετήριο το συγκεκριμένο field. Χαρακτηριστικά παραδείγματα field είναι τα «title» και «content».

3.9.5.1.2 Κλάση Document

Ένα αντικείμενο Document αντιπροσωπεύει ένα κείμενο της συλλογής προς ευρετηρίαση. Η αναζήτηση συγκεκριμένων όρων μέσω του ευρετηρίου θα έχει ως αποτέλεσμα την επιστροφή αντικειμένων Document που ανταποκρίνονται σε μεγαλύτερο βαθμό στα κριτήρια της αναζήτησης.

3.9.5.1.3 Κλάση IndexWriter

Η κλάση IndexWriter είναι υπεύθυνη για τη δημιουργία του ανεστραμμένου ευρετηρίου, την ενημέρωση του καθώς και για την προσθήκη στο ευρετήριο των κειμένων της συλλογής προς ευρετηρίαση.

3.9.5.1.4 Κλάση Analyzer

Η κλάση Analyzer είναι υπεύθυνη για την προεπεξεργασία και κανονικοποίηση των Documents πριν την προσθήκη τους στο ευρετήριο. Ειδικότερα, χρησιμοποιούνται κατάλληλες μέθοδοι και κλάσεις που περιέχουν τεχνικές tokenization (κατακερματισμός σε όρους), μετατροπή των κεφαλαίων σε μικρά γράμματα, αφαίρεση των σημείων στίξης, αφαίρεση αριθμητικών, τεχνικές λημματοποίησης, αποκοπής καταλήξεων (stemming) και αφαίρεση τετριμμένων λέξεων (stopwords) που υπάρχουν στα κείμενα της συλλογής.

3.9.5.2 Διαδικασία δημιουργίας ευρετηρίου

Για τη δημιουργία ενός ευρετηρίου απαιτούνται τα παρακάτω βήματα:

- Ορισμός μία λίστας από αρχεία.
- Δημιουργία ενός instance ενός αντικειμένου analyzer.
- Δημιουργία ενός αντικειμένου IndexWriter με την τοποθεσία του ευρετηρίου, το instance του Analyzer και κατάλληλο flag για την δημιουργία του ευρετηρίου.
- Δημιουργία των αντικειμένων documents για κάθε κείμενο της συλλογής.
- Προσθήκη αντικειμένων τύπου Field στο Document με τη χρησιμοποίηση της μεθόδου addField.
- Προσθήκη των αντικειμένων Document στο αντικείμενο IndexWriter με τη χρησιμοποίηση της μεθόδου addDocument.

3.9.5.3 Βασικές κλάσεις για searching

Οι βασικές κλάσεις που χρησιμοποιούνται για την αναζήτηση στα περιεχόμενα ενός ευρετηρίου είναι οι παρακάτω:

- IndexSearcher: Παρέχει μεθόδους για την αναζήτηση σε ένα ευρετήριο.
- QueryParser: Περιέχει ως όρισμα το ερώτημα του χρήστη και δημιουργεί ένα αντικείμενο Query.
- Query: Περιέχει τα κριτήρια αναζήτησης που δημιουργήθηκαν από το αντικείμενο QueryParser.
- Hits: Περιέχει τα αντικείμενα τύπου Document που επεστράφησαν μετά την αποστολή ενός Query στο ευρετήριο.

Για την επιτυχή αναζήτηση είναι απαραίτητο το ερώτημα της αναζήτησης να υποστεί ίδια επεξεργασία με το κείμενο που έχει γίνει indexed. Πρέπει δηλαδή να χρησιμοποιείται ο ίδιος Analyzer και για το Indexing αλλά και για την αναζήτηση, διαφορετικά τα αποτελέσματα δεν είναι έγκυρα.

Για την αναζήτηση στο ευρετήριο ακολουθούνται τα παρακάτω στάδια:

- Αρχικοποίηση ενός αντικειμένου IndexSearcher με την τοποθεσία του ευρετηρίου στο οποίο θα πραγματοποιηθεί η αναζήτηση.
- Δημιουργία ενός αντικειμένου QueryParser με ένα πεδίο στο οποίο θα πραγματοποιηθεί η αναζήτηση και τον κατάλληλο Analyzer.
- Χρησιμοποίηση της μεθόδου parse στο αντικείμενο QueryParser για τη δημιουργία ενός αντικειμένου Query.
- Χρησιμοποίηση του αντικειμένου Query στη μέθοδο αναζήτησης του IndexSearcher, ο οποίος θα επιστρέψει ένα αντικείμενο τύπου Hits που θα περιέχει τη λίστα των αντικειμένων Document που ικανοποιούν την αναζήτηση.

4 Αλγοριθμική προσέγγιση

Στη συνέχεια ακολουθεί περιγραφή των αλγόριθμων αναζήτησης που έχουν υλοποιηθεί με κατάλληλες μεθόδους στο πακέτο των κλάσεων του Hybrid 3D R-Tree και αναφέρθηκαν σε προηγούμενη ενότητα επιγραμματικά. Χρησιμοποιήθηκε το Hermes Attica data set (αναλύεται στην επόμενη ενότητα) ως δεδομένα αναζήτησης για τον έλεγχο ορθότητας των αποτελεσμάτων τους. Στη συνέχεια ακολουθεί περιγραφή των δοκιμαστικών δεδομένων.

4.1 Δοκιμαστικά δεδομένα

Για την παρουσίαση και τον έλεγχο των αλγόριθμων που έχουν αναπτυχθεί στις παρακάτω ενότητες χρησιμοποιήθηκαν τα δοκιμαστικά δεδομένα «Hermes Attica». Το σύνολο των δεδομένων που χρησιμοποιήθηκε για την πειραματική αξιολόγηση εξήχθησαν από το Hermoupolis [31] γεννήτορα συνθετικών σημασιολογικών τροχιών. Τα δεδομένα αυτά αποτελούνται από ένα σύνολο 1.450.738 εγγραφών και αναπαριστούν με χωροχρονικές συντεταγμένες σημασιολογικές τροχιές (semantic trajectories) αντικειμένων.

Η μορφή που έχουν οι εγγραφές είναι η παρακάτω:

obj_id	traj_id	subtraj_id	t	lon	lat	episode	tags	
2934	1	4	8/5/2013 17:06	237.148.111.397.776	379.456.391.156.175	MOVE	TRANSPORTATION	CAR

Σχήμα 28: Δοκιμαστικά δεδομένα Hermes Attica Dataset

Οι στήλες «traj_id» και «subtraj_id» έχουν προκύψει με την τμηματοποίηση μιας τροχιάς (trajectory) ενός αντικειμένου σε επιμέρους τροχιές (subtrajectories). Η στήλη «obj_id» αναφέρεται στο id του αντικειμένου που πραγματοποιεί την τροχιά. Δηλαδή η μοναδικότητα μιας εγγραφής προκύπτει από την τριπλέτα «object_id, traj_id, subtraj_id».

Η στήλη «t» περιέχει τη χρονική στιγμή (timestamp) που το αντικείμενο βρίσκεται σε συγκεκριμένες χωρικές συντεταγμένες (στήλες «lon» και «lat»). Η στήλη «episode»

χαρακτηρίζει αν την συγκεκριμένη χωροχρονική στιγμή το αντικείμενο ήταν σε κίνηση (τιμή «MOVE») ή όχι (τιμή «STOP»), ενώ η στήλη «tags» περιέχει επιπρόσθετες λεκτικές πληροφορίες για την συγκεκριμένη χωρική θέση/χρονική στιγμή του αντικειμένου (για παράδειγμα μέσο μεταφοράς ή τοποθεσία που βρίσκεται το αντικείμενο).

4.2 Episode-based Hybrid 3D R-Tree Search Algorithm (EB)

Για την υλοποίηση του spatiotemporal keyword pattern query ο αλγόριθμος εκμεταλλεύεται την υλοποίηση του υβριδικού Episode-based 3D R-Tree που περιγράφτηκε σε προηγούμενη ενότητα. Ο αλγόριθμος εξετάζει μέσω του υβριδικού ευρετηρίου κάθε episode που βρίσκεται στη βάση προκειμένου να γίνει έλεγχος αν ταιριάζει με το pattern που έχει δοθεί ως όρισμα καθώς και με τα episodes που προηγουμένως έχουν βρεθεί ότι ταιριάζουν.

Ο αλγόριθμος εξετάζει ταυτόχρονα τα χωροχρονικά και λεκτικά κριτήρια κάθε κόμβου προκειμένου στη συνέχεια να επισκεφθεί τα παιδιά του μέχρι σταδιακά να φτάσει στα φύλλα του δέντρου. Ειδικότερα, ξεκινώντας από τη ρίζα του δέντρου και ελέγχοντας αν ικανοποιούνται οι απαιτήσεις για το πρώτο episode σε αυτήν, σταδιακά κατεβαίνει στα υπόλοιπα επίπεδα του δέντρου, μέχρι να φτάσει στο επίπεδο των φύλλων του δέντρου και να βρεθεί ένα ή περισσότερα episodes που ικανοποιούν τα κριτήρια της αναζήτησης. Στη συνέχεια γίνεται σταδιακά έλεγχος για όλο το query pattern, για το σύνολο του semantic trajectory που έχει βρεθεί ότι ανήκουν τα συγκεκριμένα episodes, χωρίς την διάσχιση ξανά του δέντρου από τη ρίζα, αλλά κάνοντας διάσχιση μόνο των κόμβων που αντιστοιχούν στα episodes του συγκεκριμένου trajectory.

Προκειμένου να επιτευχθεί η ανωτέρω επαλήθευση, όλα τα episodes που ακολουθούν χρονικά στη συνέχεια μετά το πρώτο episode και ανήκουν στην ίδια σημασιολογική τροχιά ελέγχονται επαναληπτικά προκειμένου να βρεθεί εάν ικανοποιούν τις χωροχρονικές και λεκτικές απαιτήσεις του συνολικού regular expression. Στη συνέχεια ακολουθούν τα βήματα του αλγόριθμου σε ψευδογλώσσα.

4.2.1 Περιγραφή του Episode-based Hybrid 3D R-Tree Search Algorithm

-Έλεγξε αν ικανοποιούνται τα χωροχρονικά και λεκτικά κριτήρια για το πρώτο episode του regular expression με τα χωροχρονικά και λεκτικά δεδομένα που βρίσκονται στη ρίζα του δέντρου. Αν δεν ικανοποιούνται ο αλγόριθμος σταματάει.

-Αν ικανοποιούνται τα κριτήρια αναζήτησης με τα δεδομένα της ρίζας του δέντρου, τότε γίνεται σταδιακά έλεγχος για ποια παιδιά της ρίζας του δέντρου ικανοποιούν τα ίδια κριτήρια. Αν δεν βρεθούν παιδιά της ρίζας που να τα ικανοποιούν ο αλγόριθμος σταματάει.

-Για τα παιδιά της ρίζας του δέντρου που ικανοποιούν τα κριτήρια της αναζήτησης, επιστρέφονται επαναληπτικά τα παιδιά τους μέχρι να γίνει διάσχιση όλων των επιπέδων του δέντρου και να φτάσει η αναζήτηση στο επίπεδο των φύλλων του δέντρου. Πραγματοποιείται έλεγχος αν οι συγκεκριμένοι κόμβοι στο επίπεδο των φύλλων του δέντρου (geometry nodes) ικανοποιούν τα κριτήρια της αναζήτησης και τα επιπρόσθετα κριτήρια που αφορούν το υπόλοιπο μέρος του trajectory, μέσω της διάσχισης και του ελέγχου των υπόλοιπων κόμβων του semantic trajectory που έχει βρεθεί ως υποψήφιο προς επαλήθευση του query pattern.

-Η διαδικασία επαναλαμβάνεται σταδιακά για όλα τα episodes που έχουν βρεθεί και στη συνέχεια επιστρέφονται ως αποτελέσματα της αναζήτησης τα semantic trajectories που ικανοποιούν τα κριτήρια του query.

4.2.2 Υλοποίηση αλγόριθμου EB

Στη συνέχεια ακολουθεί αναλυτική περιγραφή της υλοποίησης.

Input: Λίστα με Minimum Bounding Box αντικείμενα για τους χωροχρονικούς περιορισμούς (envelopes), Λίστα με String για τους λεκτικούς περιορισμούς (tags), Βάση δεδομένων Neo4j προς αναζήτηση (layer)

Output: Λίστα με τα Semantic Trajectories που ικανοποιούν τις απαιτήσεις του ερωτήματος (results)

Η λίστα tags μπορεί να περιλαμβάνει και τα “*” wildchars των regular expressions. Αντίστοιχα, η λίστα envelopes μπορεί να περιλαμβάνει και «κενά» αντικείμενα MBB που υποδηλώνουν ότι στη συγκεκριμένη περίπτωση δεν υπάρχει χωροχρονικός περιορισμός.

Αλγόριθμος EB

1. begin
2. firstEpisodes=Ø , results=Ø

```
3. firstEpisodes=searchTreeEpisodeChild(layer, envelopes[0], tags[0])
4. for each episode in firstEpisodes
5.       if checkEpisodeSTL(episode, envelopes, tags) then
6.           results.add(getSemanticTrajectoryFromEpisode(episode))
7.       end if
8. end for
9. return results
10. end
```

Στη συνέχεια ακολουθεί περιγραφή των βημάτων του αλγόριθμου.

Γραμμή 2: Δημιουργία και αρχικοποίηση λίστας από episodes που θα ικανοποιούν τις απαιτήσεις του πρώτου spatiotemporal keyword pattern element.
Γραμμή 3: Διάσχιση του δέντρου και αναζήτηση episodes που ικανοποιούν τις απαιτήσεις του πρώτου spatiotemporal keyword pattern element.
Γραμμή 4-5: Για κάθε episode που έχεις βρει από την προηγούμενη διάσχιση, διάσχισε επαναληπτικά τα episodes που αυτό είναι συνδεμένο μέσω των αντίστοιχων relationships και κάνε έλεγχο αν ικανοποιούνται οι απαιτήσεις των υπόλοιπων spatiotemporal keyword pattern elements.
Γραμμή 6: Αν οι απαιτήσεις ικανοποιούνται, τότε πρόσθεσε το συγκεκριμένο semantic trajectory στη λίστα των σημασιολογικών τροχιών που θα επιστραφούν ως αποτελέσματα.
Γραμμή 9: Επέστρεψε τη συγκεκριμένη λίστα σημασιολογικών τροχιών.

Η συνάρτηση checkEpisodeSTL (STL-Spatial Temporal Textual) δέχεται ως όρισμα έναν συγκεκριμένο κόμβο που επαληθεύει τις χωροχρονικές και λεκτικές απαιτήσεις και αντιστοιχεί δηλαδή σε ένα episode που επαληθεύει το πρώτο spatiotemporal keyword όρισμα του ερωτήματος αναζήτησης. Στη συνέχεια πραγματοποιεί διάσχιση των επόμενων κόμβων που συνδέεται ο συγκεκριμένος κόμβος και ανήκουν στην ίδια σημασιολογική τροχιά προκειμένου να πραγματοποιηθεί έλεγχος εάν επαληθεύεται το σύνολο των απαιτήσεων των spatiotemporal keyword pattern ορισμάτων.

4.2.3 Παράδειγμα εκτέλεσης του αλγόριθμου EB

Στη συνέχεια ακολουθεί περιγραφή εκτέλεσης του αλγόριθμου για το παράδειγμα από το Hermes Attica data set που έχει αναφερθεί σε προηγούμενη ενότητα. Στο Σχήμα 25 υπάρχει απεικόνιση της μορφής του index για το συγκεκριμένο παράδειγμα. Υποθέτουμε ότι ως κριτήριο αναζήτησης έχει δοθεί το « HOME, * , SCHOOL, * , HOME » μαζί με τις ανάλογες χωροχρονικές συντεταγμένες για τις

περιοχές «HOME» και «SCHOOL» που αντιστοιχούν στα bounding boxes του συγκεκριμένου παραδείγματος.

Αρχικά ο αλγόριθμος πραγματοποιεί έλεγχο αν τα χωροχρονικά και λεκτικά κριτήρια του «HOME» επαληθεύονται από τα δεδομένα της ρίζας του δέντρου. Στη συνέχεια η επαλήθευση συνεχίζεται με τους κόμβους που δείχνει η ρίζα του δέντρου και για όλα τα επίπεδα του δέντρου μέχρι το επίπεδο των φύλλων του δέντρου που αντιστοιχούν σε συγκεκριμένα episodes. Ένας από τους κόμβους επαληθεύει τις απαιτήσεις και κατά συνέπεια το συγκεκριμένο episode που αυτός αναφέρεται ανταποκρίνεται στις απαιτήσεις του συγκεκριμένου query.

Στη συνέχεια γίνεται έλεγχος αν υπάρχει μεταγενέστερο χρονικά episode για το συγκεκριμένο semantic trajectory που ανήκει το episode που έχει βρεθεί νωρίτερα με τα χωροχρονικά και λεκτικά κριτήρια του «SCHOOL» στο δέντρο. Ένας από τους κόμβους της σημασιολογικής τροχιάς επαληθεύει τις χωροχρονικές και λεκτικές απαιτήσεις και ανήκει στο semantic trajectory του προηγούμενου κόμβου που έχει βρεθεί. Άρα, το συγκεκριμένο episode που αυτός αναφέρεται ικανοποιεί τις απαιτήσεις του query.

Στη συνέχεια γίνεται έλεγχος αν υπάρχει episode στο trajectory με τα χωροχρονικά και λεκτικά κριτήρια του «HOME». Ένας από τους κόμβους που ανήκουν στο trajectory επαληθεύει τις χωροχρονικές και λεκτικές απαιτήσεις που έχουν δοθεί στο query. Άρα, η σημασιολογική τροχιά που έχει βρεθεί ικανοποιεί τις απαιτήσεις του ερωτήματος αναζήτησης. Επομένως ως αποτέλεσμα του αλγόριθμου επιστρέφεται το συγκεκριμένο trajectory με tags «HOME; RELAXING, TRANSPORTATION; WALKING, SCHOOL; STUDYING, TRANSPORTATION; WALKING, HOME; RELAXING».

4.3 Semantic-Trajectory-based Hybrid 3D R-Tree Search Algorithm (STB)

Ο συγκεκριμένος αλγόριθμος ακολουθεί τη φιλοσοφία και τη δομή του πρώτου αλγόριθμου με τη διαφορά ότι το ευρετήριο (Semantic Trajectory-based Hybrid 3D R-Tree) σε αυτήν την περίπτωση δεν είναι κατασκευασμένο λαμβάνοντας υπόψη ως «δομικό στοιχείο» τα episodes των semantic trajectories, αλλά τα ίδια τα semantic trajectories ως ατομική μονάδα, όπως έχει αναφερθεί σε προηγούμενη ενότητα.

Με αυτόν τον τρόπο δημιουργείται ένα Hybrid 3D R-Tree που έχει σημαντικά μικρότερο μέγεθος σε σύγκριση με τον πρώτο ευρετήριο. Η διάσχιση των επιμέρους κόμβων που αποτελείται ένα trajectory πραγματοποιείται μόνο εφόσον

ικανοποιούνται πρώτα οι χωροχρονικές και λεκτικές απαιτήσεις των προηγούμενων επιπέδων του δέντρου και του αρχικού κόμβου του trajectory, επιτυγχάνοντας έτσι τη διάσχιση μικρότερου αριθμού κόμβων κατά την πραγματοποίηση αναζήτησης στο δέντρο.

Με την ανωτέρω μορφή του δέντρου είναι εφικτός ο σταδιακός έλεγχος του συνολικού pattern query από την αρχή εκτέλεσης του αλγόριθμου. Στη συνέχεια ακολουθούν τα βήματα του αλγόριθμου σε ψευδογλώσσα.

4.3.1 Περιγραφή του Semantic-Trajectory-based Hybrid 3D R-Tree Search Algorithm

Το συγκεκριμένο ευρετήριο δίνει τη δυνατότητα ελέγχου των χωροχρονικών και λεκτικών απαιτήσεων του συνολικού pattern query από την αρχή της διάσχισης του δέντρου, δηλαδή από τη ρίζα του.

Οι λεκτικές απαιτήσεις μπορούν να ελεγχθούν ανεξάρτητα από τη μορφή που έχει το λεκτικό regular expression του query (είτε περιλαμβάνει δηλαδή το wildchar (*) είτε όχι). Ειδικότερα, αρχικά σχηματίζεται η λεκτική μορφή του συνολικού query, δηλαδή αν το query περιέχει στη λίστα των λεκτικών απαιτήσεων τις τιμές «A», «*», «B» και «C», τότε το συνολικό λεκτικό query που θα σχηματιστεί θα έχει τη μορφή «A*BC». Χρησιμοποιώντας την παραπάνω μορφή λεκτικού query (A*BC) κατά τη διάσχιση των επιπέδων του δέντρου (εκτός του επιπέδου των φύλλων) γίνεται έλεγχος στα λεκτικά ευρετήρια κάθε επιπέδου αν επαληθεύεται το ανωτέρω συνολικό λεκτικό query. Όσον αφορά τις χωροχρονικές απαιτήσεις προκειμένου να γίνει έλεγχος κατά την διάσχιση όλων των επιπέδων του δέντρου πρέπει να είναι δυνατός ο σχηματισμός ενός συνολικού MBB που περιβάλλει το regular expression. Προκειμένου να είναι αυτό δυνατό, πρέπει το query pattern που έχει δοθεί ως όρισμα περιλαμβάνει για όλα τα episodes spatiotemporal constraints. Σε περίπτωση που δεν περιλαμβάνονται σε όλα τα atomic pattern elements spatiotemporal constraints, τότε οι χωροχρονικές απαιτήσεις ελέγχονται στα φύλλα του δέντρου.

Για το παράδειγμα που αναφέρθηκε παραπάνω που το query έχει τη μορφή «A*BC» αν τα A,*,B και C έχουν και τα τέσσερα spatiotemporal constraints τότε μπορεί να σχηματιστεί συνολικό MBB για το pattern query. Σε περίπτωση για παράδειγμα που το «B» δεν έχει spatiotemporal constraints δεν μπορεί να σχηματιστεί ένα συνολικό MBB με τα υπόλοιπα spatiotemporal constraints που έχουν δοθεί ως όρισμα γιατί μπορεί να υπάρχει μία σημασιολογική τροχιά που να ικανοποιεί την λεκτική απαίτηση του «B» αλλά να έχει MBB στο pattern «B» που δεν επικαλύπτεται από το

συνολικό MBB που έχει σχηματιστεί με τα λιγότερα constraints. Δεν καλύπτονται δηλαδή όλες οι πιθανές περιπτώσεις σημασιολογικών τροχιών με διαφορετικά μεγέθη MBB στα επιμέρους episodes τους.

Διακρίνουμε δύο διαφορετικές περιπτώσεις διάσχισης του ευρετηρίου, αναλόγως αν το query pattern ανήκει στη συγκεκριμένη υποκατηγορία ή έχει τη γενικότερη μορφή.

4.3.1.1 Διάσχιση με σχηματισμό MBB για το regular expression του pattern query

-Πραγματοποίησε έλεγχο για τα χωροχρονικά και λεκτικά κριτήρια του ερωτήματος με τα χωροχρονικά και λεκτικά δεδομένα της ρίζας του δέντρου. Αν δεν ικανοποιούνται τα κριτήρια αναζήτησης ο αλγόριθμος σταματάει.

-Αν ικανοποιούνται τα κριτήρια αναζήτησης με τα δεδομένα της ρίζας του δέντρου, τότε γίνεται σταδιακά έλεγχος για ποια παιδιά της ρίζας του δέντρου ικανοποιούν τα ίδια κριτήρια. Αν δεν βρεθούν παιδιά της ρίζας που να τα ικανοποιούν ο αλγόριθμος σταματάει.

-Για τα παιδιά της ρίζας του δέντρου που ικανοποιούν τα κριτήρια της αναζήτησης, επιστρέφονται επαναληπτικά τα παιδιά τους. Πραγματοποιείται δηλαδή σταδιακά έλεγχος για όλα τα επίπεδα του δέντρου μέχρι το επίπεδο των φύλλων. Στο κατώτερο επίπεδο γίνεται έλεγχος αν οι συγκεκριμένοι κόμβοι (geometry nodes) ικανοποιούν τα χωροχρονικά και λεκτικά κριτήρια της αναζήτησης.

-Επιστρέφονται ως αποτελέσματα της αναζήτησης τα semantic trajectories που ικανοποιούν τα κριτήρια του query.

4.3.1.2 Διάσχιση χωρίς σχηματισμό MBB για το regular expression του pattern query

Η συγκεκριμένη διάσχιση στο ευρετήριο αποτελεί τη γενικότερη μορφή του αλγόριθμου και μπορεί να χρησιμοποιηθεί για έλεγχο ορθότητας των queries ανεξάρτητα από την ύπαρξη ή όχι συνολικού MBB για το regular expression του pattern query.

-Πραγματοποίησε έλεγχο για τα λεκτικά κριτήρια του ερωτήματος με τα λεκτικά δεδομένα της ρίζας του δέντρου. Αν δεν ικανοποιούνται τα κριτήρια αναζήτησης ο αλγόριθμος σταματάει.

-Αν ικανοποιούνται τα λεκτικά κριτήρια αναζήτησης με τα δεδομένα της ρίζας του δέντρου, τότε γίνεται σταδιακά έλεγχος για ποια παιδιά της ρίζας του δέντρου ικανοποιούν τα ίδια κριτήρια. Αν δεν βρεθούν παιδιά της ρίζας που να τα ικανοποιούν ο αλγόριθμος σταματάει.

-Για τα παιδιά της ρίζας του δέντρου που ικανοποιούν τα κριτήρια της αναζήτησης, επιστρέφονται επαναληπτικά τα παιδιά τους. Πραγματοποιείται δηλαδή σταδιακά έλεγχος για όλα τα επίπεδα του δέντρου μέχρι το επίπεδο των φύλλων. Στο κατώτερο επίπεδο γίνεται έλεγχος αν οι συγκεκριμένοι κόμβοι (geometry nodes) ικανοποιούν τα χωροχρονικά και λεκτικά κριτήρια της αναζήτησης.

-Επιστρέφονται ως αποτελέσματα της αναζήτησης τα semantic trajectories που ικανοποιούν τα κριτήρια του query.

4.3.2 Υλοποίηση αλγόριθμου STB

Στη συνέχεια ακολουθεί αναλυτική περιγραφή αλγοριθμικής υλοποίησης που αναλόγως της ύπαρξης ή όχι συνολικού MBB εκτελεί αντίστοιχα τη γενική ή την ειδικότερη περίπτωση του αλγόριθμου.

Input: Λίστα με Minimum Bounding Box αντικείμενα για τους χωροχρονικούς περιορισμούς (envelopes), Λίστα με String για τους λεκτικούς περιορισμούς (tags), Βάση δεδομένων Neo4j προς αναζήτηση (layer)

Output: Λίστα με τα Semantic Trajectories που ικανοποιούν τις απαιτήσεις του ερωτήματος (results)

Η λίστα tags μπορεί να περιλαμβάνει και τα “*” wildchars των regular expressions. Αντίστοιχα, η λίστα envelopes μπορεί να περιλαμβάνει και «κενά» αντικείμενα MBB που υποδηλώνουν ότι στη συγκεκριμένη περίπτωση δεν υπάρχει χωροχρονικός περιορισμός.

Αλγόριθμος STB - Συνδυαστική μορφή

1. begin
2. results=Ø , depthNodes=Ø

```
3.  boolean trajectoryMBB=checkTrajectoryMBB(layer, envelopes)
4.  depthNodes(0)=getRootConstraints(layer,envelopes,tags)
5.  treeDepth=getTreeDepth(layer)
6.  int i=0
7.  while (i<treeDepth)
8.      for each node in depthNodes(i)
9.          if (i=treeDepth-1) then
10.             if checkSpatiotemporalKeywordConstraints(node) then
11.               results.add(getSemanticTrajectory(node)
12.             end if
13.         else
14.             if (trajectoryMBB) then
15.                 if (checkSpatiotemporalKeywordConstraints(node) then
    depthNodes(i+1).add(node)
16.                 end if
17.             else
18.                 if checkKeywordConstraints(node) then depthNodes(i+1).add(node)
19.                 end if
20.           end if
21.         end if
22.     end for
23.     i=i+1
24. end while
25. return results
26. end
```

Στη συνέχεια ακολουθεί περιγραφή των βημάτων του αλγόριθμου.

Γραμμή 2: Δημιουργία και αρχικοποίηση λίστας από trajectories που θα ικανοποιούν τις απαιτήσεις του ερωτήματος και λίστας από λίστες (μία για κάθε επίπεδο) που ικανοποιούν τις απαιτήσεις για κάθε επίπεδο του δέντρου.
Γραμμή 3: Έλεγχος αν σχηματίζεται συνολικό MBB για το query.
Γραμμή 5: Εύρεση του ύψους του δέντρου.
Γραμμή 7: Έναρξη διάσχισης (breadth-first) του δέντρου ανά επίπεδο.
Γραμμή 8-23: Για κάθε επίπεδο του δέντρου κάνε έλεγχο ποιοι κόμβοι ικανοποιούν τις απαιτήσεις του ερωτήματος. Για τα παιδιά των συγκεκριμένων κόμβων στη συνέχεια πραγματοποίησε τον ίδιο έλεγχο.
Γραμμή 10: Όταν η επανάληψη είναι στο επίπεδο των φύλλων του δέντρου τότε κάνε έλεγχο ποια semantic trajectories ικανοποιούν τις απαιτήσεις του ερωτήματος.
Γραμμή 11: Αν οι απαιτήσεις ικανοποιούνται, τότε πρόσθεσε το συγκεκριμένο semantic trajectory στη λίστα των σημασιολογικών τροχιών (results) που θα επιστραφούν ως αποτελέσματα.
Γραμμή 24: Επέστρεψε τη συγκεκριμένη λίστα (results) σημασιολογικών τροχιών.

Στη συνέχεια ακολουθεί η αλγοριθμική υλοποίηση της γενικότερης μορφής του αλγόριθμου.

Αλγόριθμος STB - Γενική Μορφή

```
1. begin
2. results=Ø , depthNodes=Ø
3. depthNodes(0)=getRootConstraints(layer,envelopes,tags)
4. treeDepth=getTreeDepth(layer)
5. int i=0
6. while (i<treeDepth)
7.     for each node in depthNodes(i)
8.         if (i=treeDepth-1) then
9.             if checkSpatiotemporalKeywordConstraints(node) then
10.              results.add(getSemanticTrajectory(node)
11.            end if
12.        else
13.            if checkKeywordConstraints(node) then depthNodes(i+1).add(node)
14.            end if
15.        end if
16.    end for
17.    i=i+1
18. end while
19. return results
20. end
```

4.3.3 Παράδειγμα εκτέλεσης του αλγόριθμου STB

Στη συνέχεια ακολουθεί περιγραφή εκτέλεσης του αλγόριθμου για το παράδειγμα από το Hermes Attica data set. Υποθέτουμε ότι ως κριτήριο αναζήτησης έχει δοθεί το « TRANSPORTATION, SCHOOL, TRANSPORTATION » μαζί με τις ανάλογες χωροχρονικές συντεταγμένες για τα «TRANSPORTATION» και «SCHOOL» που αντιστοιχούν στα bounding boxes του συγκεκριμένου παραδείγματος.

Αρχικά ο αλγόριθμος πραγματοποιεί έλεγχο αν τα χωροχρονικά και λεκτικά κριτήρια του « TRANSPORTATION, SCHOOL, TRANSPORTATION » επαληθεύονται από τα δεδομένα της ρίζας του δέντρου. Στη συνέχεια η επαλήθευση συνεχίζεται για όλα τα επίπεδα του δέντρου μέχρι το επίπεδο των φύλλων του δέντρου. Η αναζήτηση στο επίπεδο των φύλλων θα βρει κατάλληλο κόμβο που αντιστοιχεί σε συγκεκριμένο semantic trajectory και επαληθεύει τις χωροχρονικές και λεκτικές απαιτήσεις του

ερωτήματος. Κατά συνέπεια το συγκεκριμένο trajectory που αυτός αναφέρεται ικανοποιεί τις συνολικές απαιτήσεις του ανωτέρω query.

Υποθέτουμε ως δεύτερο παράδειγμα στη συνέχεια ότι ως κριτήριο αναζήτησης έχει δοθεί το « HOME, * , SCHOOL, * , HOME » μαζί με τις ανάλογες χωροχρονικές συντεταγμένες μόνο για την περιοχή «HOME».

Αρχικά ο αλγόριθμος πραγματοποιεί έλεγχο αν τα λεκτικά κριτήρια του «HOME, * , SCHOOL, * , HOME» επαληθεύονται από τα δεδομένα της ρίζας του δέντρου. Στη συνέχεια η λεκτική επαλήθευση συνεχίζεται για όλα τα επίπεδα του δέντρου μέχρι το επίπεδο των φύλλων του δέντρου. Στο επίπεδο των φύλλων του δέντρου θα βρεθεί κατάλληλος κόμβος που αντιστοιχεί σε συγκεκριμένο semantic trajectory και επαληθεύει επίσης τις χωροχρονικές και λεκτικές απαιτήσεις του ερωτήματος. Κατά συνέπεια το συγκεκριμένο trajectory που αυτός αναφέρεται ικανοποιεί τις συνολικές απαιτήσεις του ανωτέρω query.

4.4 Enhanced Semantic-Trajectory-based Hybrid 3D R-Tree Search Algorithm (ESTB)

Ο συγκεκριμένος αλγόριθμος ακολουθεί τη φιλοσοφία και τη δομή του δεύτερου αλγόριθμου και χρησιμοποιεί το Enhanced Semantic Trajectory-based Hybrid 3D R-Tree. Όπως αναφέρθηκε δηλαδή και σε προηγούμενη ενότητα κάθε κόμβος του δέντρου περιλαμβάνει το σύνολο των tags που υπάρχουν στα παιδιά του αποθηκευμένα σε ξεχωριστά node properties για κάθε tag. Σε κάθε property-tag ξεκινώντας από τη ρίζα του δέντρου υπάρχει ένα tag-MBB που αντιστοιχεί στο χωροχρονικό union των παιδιών του κόμβου που περιέχουν το συγκεκριμένο tag. Αντίστοιχα υπάρχουν tag-properties και για τα υπόλοιπα επίπεδα του δέντρου. Στη συνέχεια ακολουθούν τα βήματα του αλγόριθμου σε ψευδογλώσσα.

4.4.1 Περιγραφή του Enhanced Semantic-Trajectory-based Hybrid 3D R-Tree Search Algorithm

-Πραγματοποίησε για το query pattern που έχει δοθεί ως όρισμα τους ελέγχους του αλγόριθμου STB και στη συνέχεια έλεγξε αν το σύνολο των tags που έχουν δοθεί ως όρισμα υπάρχουν στη ρίζα του δέντρου ως ξεχωριστά properties και ικανοποιούν τους χωροχρονικούς περιορισμούς που υπάρχουν στη ρίζα για κάθε ένα από αυτά

τα tags-properties. Ειδικότερα για κάθε MBB-tag που υπάρχει γίνεται έλεγχος αν ικανοποιείται χωροχρονικά από τα αντίστοιχα tag που υπάρχουν στο query. Αν υπάρχει κάποιο tag-property που δεν ικανοποιείται ή αν δεν ικανοποιούνται τα υπόλοιπα κριτήρια αναζήτησης ο αλγόριθμος σταματάει.

-Αν ικανοποιούνται τα κριτήρια αναζήτησης με τα δεδομένα της ρίζας του δέντρου, τότε γίνεται σταδιακά ο ίδιος έλεγχος για ποια παιδιά της ρίζας του δέντρου ικανοποιούν τα ίδια κριτήρια. Αν δεν βρεθούν παιδιά της ρίζας που να τα ικανοποιούν ο αλγόριθμος σταματάει.

-Για τα παιδιά της ρίζας του δέντρου που ικανοποιούν τα κριτήρια της αναζήτησης, επιστρέφονται επαναληπτικά τα παιδιά τους. Πραγματοποιείται επίσης έλεγχος αν οι συγκεκριμένοι κόμβοι ικανοποιούν τα ίδια κριτήρια αναζήτησης όπως στα προηγούμενα βήματα με τα tag-properties. Το συγκεκριμένο βήμα, επαναλαμβάνεται για όλα τα επίπεδα του δέντρου, μέχρι τα φύλλα του δέντρου. Στα φύλλα του δέντρου γίνεται στη συνέχεια έλεγχος αν υπάρχουν semantic trajectories που ικανοποιούν το query pattern.

-Επιστρέφονται ως αποτελέσματα της αναζήτησης τα semantic trajectories που ικανοποιούν τα κριτήρια του query.

4.4.2 Υλοποίηση αλγόριθμου ESTB

Στη συνέχεια ακολουθεί αναλυτική περιγραφή της υλοποίησης.

Input: Λίστα με Minimum Bounding Box αντικείμενα για τους χωροχρονικούς περιορισμούς (envelopes), Λίστα με String για τους λεκτικούς περιορισμούς (tags), Βάση δεδομένων Neo4j προς αναζήτηση (layer)

Output: Λίστα με τα Semantic Trajectories που ικανοποιούν τις απαιτήσεις του ερωτήματος (results)

Η λίστα tags μπορεί να περιλαμβάνει και τα “*” wildchars των regular expressions. Αντίστοιχα, η λίστα envelopes μπορεί να περιλαμβάνει και «κενά» αντικείμενα MBB που υποδηλώνουν ότι στη συγκεκριμένη περίπτωση δεν υπάρχει χωροχρονικός περιορισμός.

Αλγόριθμος ESTB

1. begin

```
2. results=Ø , depthNodes=Ø
3. boolean trajectoryMBB=checkTrajectoryMBB(layer, envelopes)
4. depthNodes(0)=getRootTagConstraints(layer,envelopes,tags)
5. treeDepth=getTreeDepth(layer)
6. int i=0
7. while (i<treeDepth)
8.     for each node in depthNodes(i)
9.         if (i=treeDepth-1) then
10.            if checkSpatiotemporalKeywordConstraints(node) &&
    checkTagConstraints(node)  then
11.                results.add(getSemanticTrajectory(node)
12.            end if
13.        else
14.            if (trajectoryMBB) then
15.                if (checkSpatiotemporalKeywordConstraints(node) &&
    checkTagConstraints(node)   then depthNodes(i+1).add(node)
16.                end if
17.            else
18.                if checkKeywordConstraints(node) && checkTagConstraints(node)  then
    depthNodes(i+1).add(node)
19.                end if
20.            end if
21.        end if
22.    end for
23. i=i+1
24. end while
25. return results
26. end
```

Ο αλγόριθμος έχει την ίδια μορφή με τον προηγούμενο αλγόριθμο (STB) αναζήτησης με τη διαφορά ότι κατά την διάσχιση των επιπέδων του δέντρου και τον έλεγχο ικανοποίησης των κριτηρίων του ερωτήματος ελέγχεται επιπρόσθετα αν υπάρχει το σύνολο των tags (λεκτικά atomic pattern elements) του ερωτήματος στο συγκεκριμένο επίπεδο και αν κάθε tag-property στο εκάστοτε επίπεδο του δέντρου ικανοποιεί τις χωροχρονικές απαιτήσεις που υπάρχουν στο ερώτημα για αυτό το tag (γραμμές 4, 10, 15, 18).

4.4.3 Παράδειγμα εκτέλεσης του αλγόριθμου ESTB

Στη συνέχεια ακολουθεί περιγραφή εκτέλεσης του αλγόριθμου για το παράδειγμα από το Hermes Attica data set. Υποθέτουμε ότι ως κριτήριο αναζήτησης έχει δοθεί το « TRANSPORTATION, SCHOOL, TRANSPORTATION » μαζί με τις ανάλογες χωροχρονικές συντεταγμένες για τα «TRANSPORTATION» και «SCHOOL» που αντιστοιχούν στα bounding boxes του συγκεκριμένου παραδείγματος.

Αρχικά ο αλγόριθμος πραγματοποιεί έλεγχο αν τα χωροχρονικά και λεκτικά κριτήρια του « TRANSPORTATION, SCHOOL, TRANSPORTATION » επαληθεύονται από τα δεδομένα της ρίζας του δέντρου. Ελέγχεται δηλαδή αν τα συγκεκριμένα tags υπάρχουν στη ρίζα του δέντρου και ικανοποιούνται οι χωροχρονικές απαιτήσεις που αυτά έχουν, μέσω των tag-properties. Ειδικότερα, γίνεται έλεγχος αν υπάρχουν στη ρίζα του δέντρου τα tag properties με ονομασία «SCHOOL» και «TRANSPORTATION» και αν το MBB που αντιστοιχεί στα συγκεκριμένα node properties επαληθεύεται χωροχρονικά με τα MBB των αντίστοιχων tags του ερωτήματος. Στη συνέχεια η επαλήθευση συνεχίζεται με τον ίδιο τρόπο για όλα τα επίπεδα του δέντρου μέχρι το επίπεδο των φύλλων του δέντρου. Στο επίπεδο των φύλλων του δέντρου θα βρεθεί κατάλληλος κόμβος που αντιστοιχεί σε semantic trajectory που επαληθεύει τις απαιτήσεις και κατά συνέπεια το συγκεκριμένο trajectory ικανοποιεί τις συνολικές απαιτήσεις του ανωτέρω query.

4.5 Συγκριτικά απόδοσης των αλγόριθμων

Στη συνέχεια ακολουθούν υποενότητες που παρουσιάζουν συγκριτικά απόδοσης των τριών αλγόριθμων και των αντίστοιχων ευρετηρίων. Χρησιμοποιήθηκαν 2 διαφορετικά hardware setups για την πραγματοποίηση των συγκριτικών απόδοσης. Τα hardware χαρακτηριστικά του πρώτου υπολογιστή (Hardware Set A) που χρησιμοποιήθηκε στα συγκριτικά απόδοσης είναι «Intel Core 2 Duo CPU P7350 2 GHz, 2 GB RAM, 60 GB Hard disk, Windows Vista Business 32 bit» (Macbook 2008 model), ενώ τα hardware χαρακτηριστικά του δεύτερου υπολογιστή (Hardware Set B) είναι «Intel Core i5-2320 CPU 3.00 GHz, 8 GB RAM, 400 GB Hard disk, Windows 7 64 bit».

Για τις ανάγκες των παρακάτω μετρήσεων απόδοσης χρησιμοποιήθηκε το dataset που έχει αναφερθεί σε προηγούμενη ενότητα. Για τον έλεγχο του χρόνου απόκρισης των ερωτημάτων χρησιμοποιήθηκαν 4 διαφορετικές μορφές από query patterns που η κάθε μία περιέχει 2, 3, 5 και 7 pattern elements. Για κάθε μία από αυτές τις μορφές

επιλέχθηκαν 5 διαφορετικά pattern queries και στη συνέχεια εξάχθηκε ο μέσος όρος του χρόνου εκτέλεσης του ερωτήματος για κάθε μία μορφή, ανά αλγόριθμο και ευρετήριο.

Ο χρόνος εκτέλεσης των ερωτημάτων αφορά τον χρόνο εκτέλεσης της κατάλληλης μεθόδου αναζήτησης μετά τη διασύνδεση στη Neo4j Spatiotemporal βάση δεδομένων, δηλαδή μετά το start-up της βάσης. Ειδικότερα, δημιουργήθηκαν 6 διαφορετικά σύνολα εγγραφών από το dataset που περιέχουν τον παρακάτω αριθμό από records:

Dataset	Αριθμός εγγραφών
Dataset 1	250.305
Dataset 2	500.913
Dataset 3	750.167
Dataset 4	1.000.038
Dataset 5	1.250.133
Dataset 6	1.450.739

Σχήμα 29: Δοκιμαστικά datasets για τα συγκριτικά απόδοσης των αλγόριθμων

Ο διαχωρισμός των datasets πραγματοποιήθηκε ανά 250.000 περίπου εγγραφές με βάση τα διαφορετικά trajectories που ολοκληρώνονταν στο συγκεκριμένο αριθμό εγγραφών.

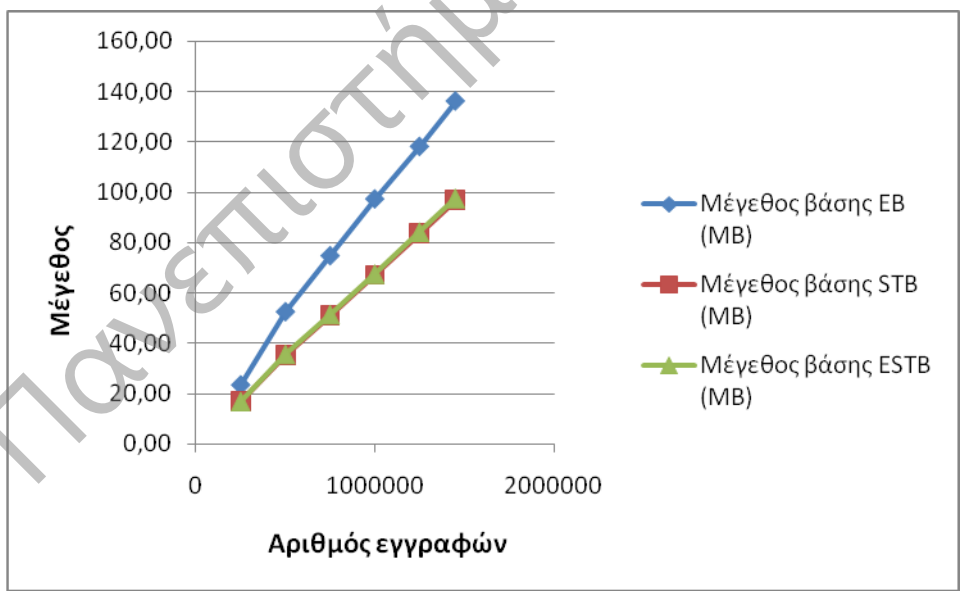
4.5.1 Σύγκριση μεγέθους βάσης

Στη συνέχεια ακολουθεί κατάλληλος πίνακας που παρουσιάζει για κάθε ένα από τα τρία ευρετήρια το μέγεθος της βάσης που έχει δημιουργηθεί ανά dataset.

Αριθμός εγγραφών	Μέγεθος βάσης EB (MB)	Μέγεθος βάσης STB (MB)	Μέγεθος βάσης ESTB (MB)
Dataset 1 (250.305)	23,50	16,80	16,90
Dataset 2 (500.913)	52,50	35,40	35,70
Dataset 3 (750.167)	74,70	51,10	51,50
Dataset 4 (1.000.038)	97,20	67,20	67,60
Dataset 5 (1.250.133)	118,00	83,80	84,30
Dataset 6 (1.450.739)	136,00	97,10	97,70

Σχήμα 30: Συγκριτικός πίνακας αριθμού εγγραφών και μεγέθους βάσης ανά ευρετήριο

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 30 και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 31: Διάγραμμα αριθμού εγγραφών και μεγέθους βάσης ανά ευρετήριο

Από τα δεδομένα του πίνακα του Σχήματος 30 και του αντίστοιχου διαγράμματος του Σχήματος 31 μπορούμε να παρατηρήσουμε ότι το ευρετήριο EB δημιουργεί συγκριτικά με τα άλλα 2 ευρετήρια για κάθε dataset μεγαλύτερο μέγεθος στην αντίστοιχη βάση δεδομένων. Το γεγονός αυτό οφείλεται στον μεγαλύτερο αριθμό από κόμβους και κορυφές που δημιουργούνται στο συγκεκριμένο ευρετήριο σε σύγκριση με τα άλλα 2 ευρετήρια. Τα ευρετήρια STB και ESTB παρουσιάζουν σχεδόν ίδιο μέγεθος καθώς η μορφολογία του υβριδικού R-Δέντρου που δημιουργείται έχει παραπλήσια μορφή.

4.5.2 Σύγκριση συνολικού χρόνου δημιουργίας βάσης δεδομένων

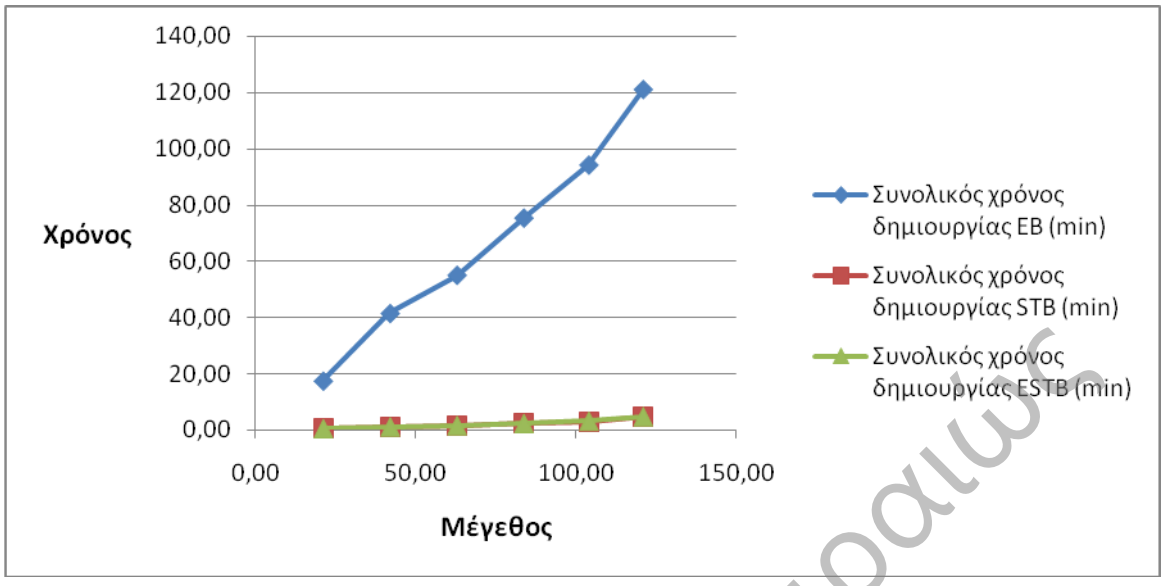
Μέγεθος αρχείου dataset (MB)	Συνολικός χρόνος δημιουργίας EB (min)	Συνολικός χρόνος δημιουργίας STB (min)	Συνολικός χρόνος δημιουργίας ESTB (min)
21,00	17,46	0,44	0,46
41,90	41,43	1,04	1,11
62,80	54,98	1,57	1,56
83,70	75,29	2,26	2,32
104,00	94,07	2,90	3,11
121,00	120,85	4,69	4,78

Σχήμα 32: Συγκριτικός πίνακας μεγέθους αρχείου βάσης και συνολικού χρόνου δημιουργίας βάσης ανά ευρετήριο (HS A)

Μέγεθος αρχείου dataset (MB)	Συνολικός χρόνος δημιουργίας EB (min)	Συνολικός χρόνος δημιουργίας STB (min)	Συνολικός χρόνος δημιουργίας ESTB (min)
21,00	17,32	0,38	0,40
41,90	41,87	0,92	0,86
62,80	55,19	1,20	1,26
83,70	72,44	1,87	1,92
104,00	88,01	2,37	2,43
121,00	100,98	3,27	3,24

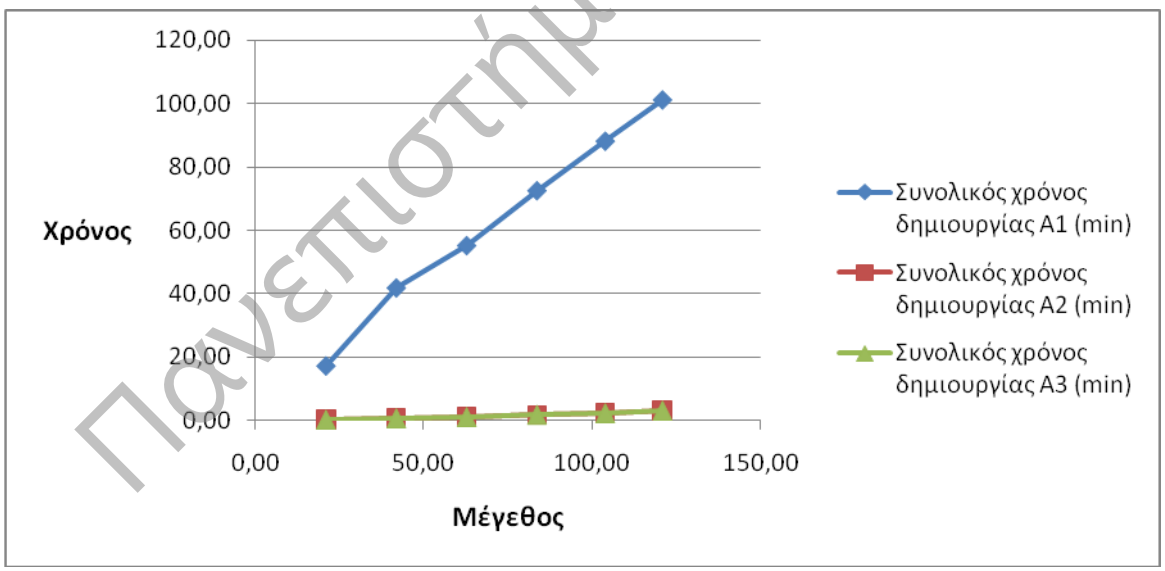
Σχήμα 33: Συγκριτικός πίνακας μεγέθους αρχείου βάσης και συνολικού χρόνου δημιουργίας βάσης ανά ευρετήριο (HS B)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 32 και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 34: Διάγραμμα μεγέθους αρχείου βάσης και συνολικού χρόνου δημιουργίας βάσης ανά ευρετήριο (Hardware Set A)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 33 και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 35: Διάγραμμα μεγέθους αρχείου βάσης και συνολικού χρόνου δημιουργίας βάσης ανά ευρετήριο (Hardware Set B)

Από τα δεδομένα των πινάκων των Σχημάτων 32 και 33 και των αντίστοιχων διαγραμμάτων των Σχημάτων 34 και 35 μπορούμε να παρατηρήσουμε ότι το

ευρετήριο EB χρειάζεται συγκριτικά με τα άλλα 2 ευρετήρια για κάθε dataset μεγαλύτερο χρονικό διάστημα για τη δημιουργία της αντίστοιχης βάσης δεδομένων. Το γεγονός αυτό οφείλεται στον μεγαλύτερο αριθμό από κόμβους και κορυφές που δημιουργούνται στο συγκεκριμένο ευρετήριο σε σύγκριση με τα άλλα 2 ευρετήρια. Τα ευρετήρια STB και ESTB παρουσιάζουν σχεδόν ίδιο μέγεθος καθώς η μορφολογία του υβριδικού R-Δέντρου που δημιουργείται έχει παραπλήσια μορφή.

4.5.3 Χρόνοι εκτέλεσης των spatiotemporal keyword pattern queries ανά αλγόριθμο

Στη συνέχεια ακολουθούν αναλυτικοί πίνακες για κάθε αλγόριθμο που παρουσιάζουν τον χρόνο εκτέλεσης των spatiotemporal keyword pattern queries ανά αριθμό spatiotemporal keyword ορισμάτων και ανά Hardware Set (HS).

4.5.3.1 Αλγόριθμος EB

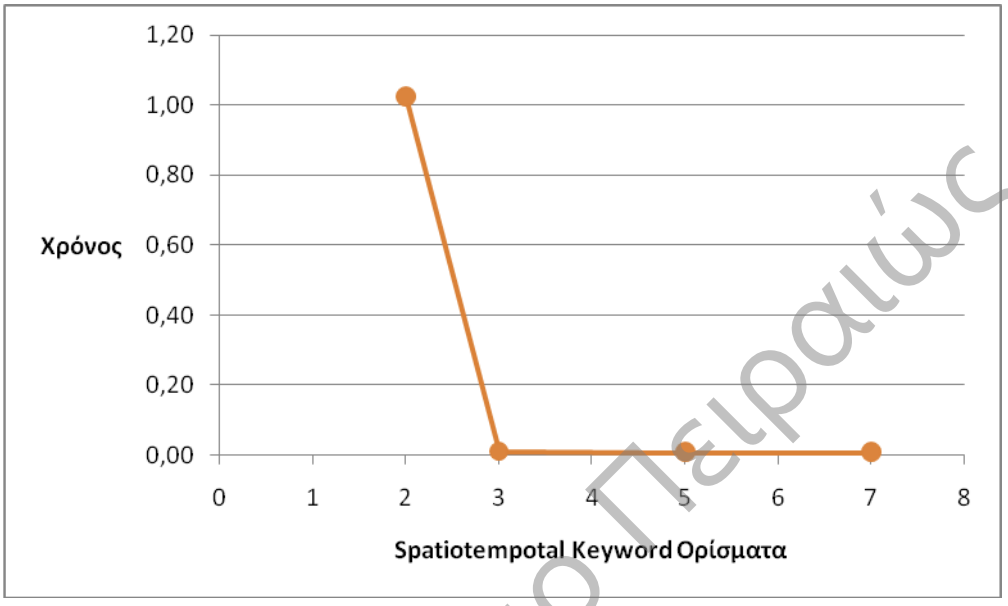
Αριθμός spatiotemporal keyword ορισμάτων	Χρόνος εκτέλεσης μεθόδου αναζήτησης (sec)
2	1,03
3	0,01
5	0,01
7	0,01

Σχήμα 36: Πίνακας αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης της μεθόδου αναζήτησης για 1.450.739 εγγραφές για τον Αλγόριθμο EB (Hardware Set A)

Αριθμός spatiotemporal keyword ορισμάτων	Χρόνος εκτέλεσης μεθόδου αναζήτησης (sec)
2	0,56
3	0,01
5	0,01
7	0,01

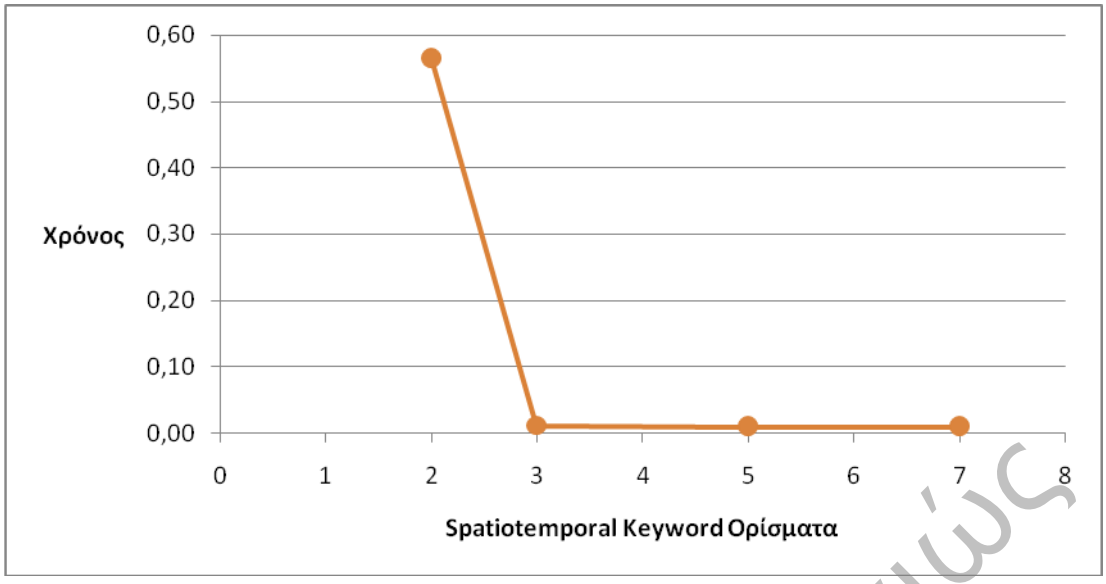
Σχήμα 37: Πίνακας αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης της μεθόδου αναζήτησης για 1.450.739 εγγραφές για τον Αλγόριθμο EB (Hardware Set B)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 36 (Hardware Set A) και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 38: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τον Αλγόριθμο EB (Hardware Set A)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 37 (Hardware Set A) και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 39: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τον Αλγόριθμο EB (Hardware Set B)

Από τα δεδομένα των πινάκων των Σχημάτων 36 και 37 και των αντίστοιχων διαγραμμάτων των Σχημάτων 38 και 39 μπορούμε να παρατηρήσουμε ότι ο Αλγόριθμος EB έχει συγκριτικά καλύτερη απόδοση όταν το query αποτελείται από μεγάλο αριθμό από spatiotemporal keyword ορίσματα. Η σταδιακή αύξηση του αριθμού των ορισμάτων έχει ως αποτέλεσμα την μείωση του χρόνου εκτέλεσης του ερωτήματος.

4.5.3.2 Αλγόριθμος STB

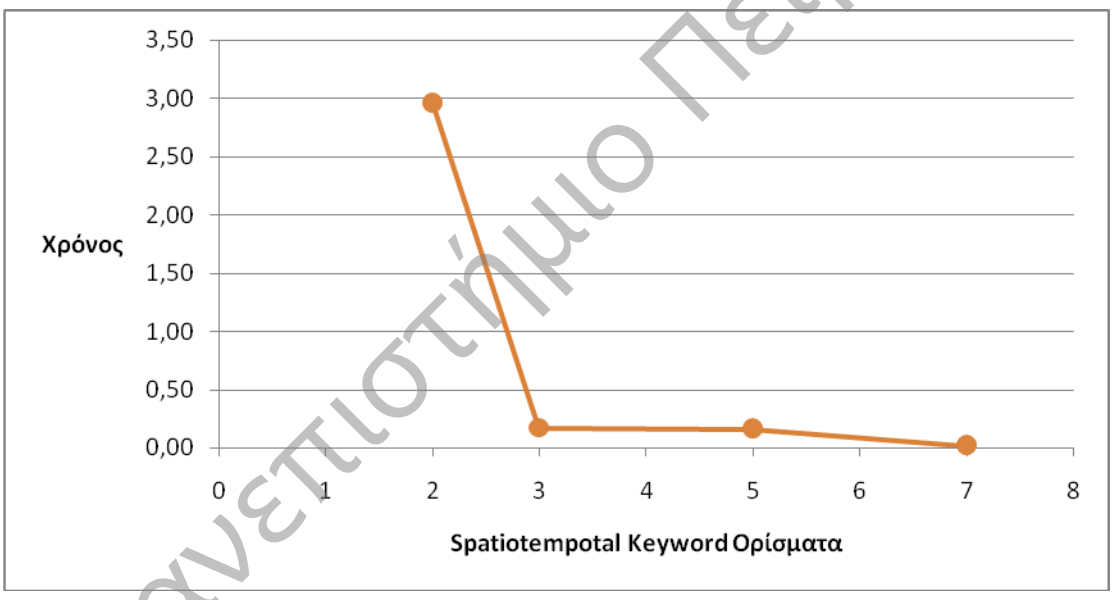
Αριθμός spatiotemporal keyword ορισμάτων	Χρόνος εκτέλεσης μεθόδου αναζήτησης (sec)
2	2,96
3	0,17
5	0,17
7	0,02

Σχήμα 40: Πίνακας αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης της μεθόδου αναζήτησης για 1.450.739 εγγραφές για τον Αλγόριθμο STB (Hardware Set A)

Αριθμός spatiotemporal keyword ορισμάτων	Χρόνος εκτέλεσης μεθόδου αναζήτησης (sec)
2	0,48
3	0,08
5	0,08
7	0,01

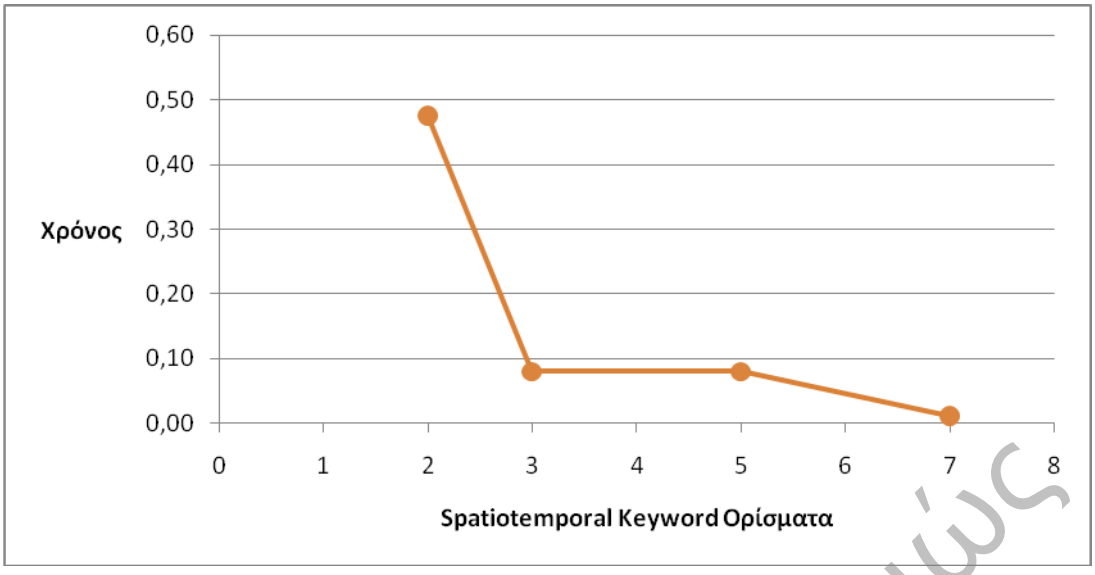
Σχήμα 41: Πίνακας αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης της μεθόδου αναζήτησης για 1.450.739 εγγραφές για τον Αλγόριθμο STB (Hardware Set B)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 40 (Hardware Set A) και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα..



Σχήμα 42: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τον Αλγόριθμο STB (Hardware Set A)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 41 (Hardware Set A) και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 43: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τον Αλγόριθμο STB (Hardware Set B)

Παρατηρούμε ότι και στις δύο περιπτώσεις των Hardware Sets, η σταδιακή αύξηση του αριθμού των ορισμάτων έχει ως αποτέλεσμα την μείωση του χρόνου εκτέλεσης του ερωτήματος. Όσο μεγαλώνει ο αριθμός από ορίσματα η συμπεριφορά του αλγόριθμου είναι σχετικά σταθερή όσον αφορά τον χρόνο εκτέλεσης του ερωτήματος.

Από τα δεδομένα των πινάκων του Σχημάτων 40 και 41 και των αντίστοιχων διαγραμμάτων των Σχημάτων 42 και 43 μπορούμε να παρατηρήσουμε ότι ο Αλγόριθμος STB έχει συγκριτικά καλύτερη απόδοση όταν το query αποτελείται από μεγάλο αριθμό από spatiotemporal keyword ορίσματα. Η σταδιακή αύξηση του αριθμού των ορισμάτων έχει ως αποτέλεσμα τη μείωση του χρόνου εκτέλεσης του ερωτήματος.

4.5.3.3 Αλγόριθμος ESTB

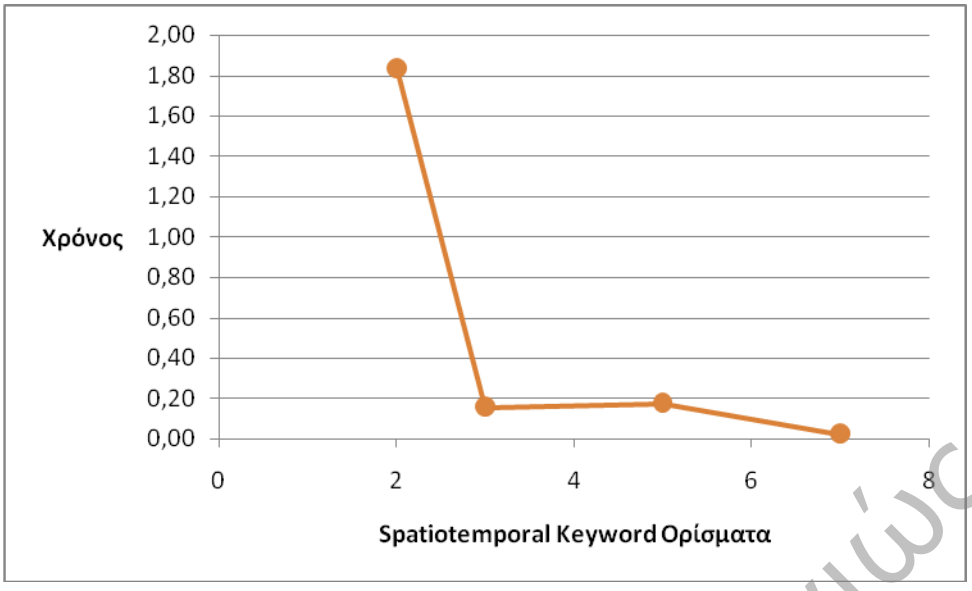
Αριθμός spatiotemporal keyword ορισμάτων	Χρόνος εκτέλεσης μεθόδου αναζήτησης (sec)
2	1,84
3	0,16
5	0,17
7	0,02

Σχήμα 44: Πίνακας αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης της μεθόδου αναζήτησης για 1.450.739 εγγραφές για τον Αλγόριθμο ESTB (Hardware Set A)

Αριθμός spatiotemporal keyword ορισμάτων	Χρόνος εκτέλεσης μεθόδου αναζήτησης (sec)
2	0,30
3	0,08
5	0,08
7	0,01

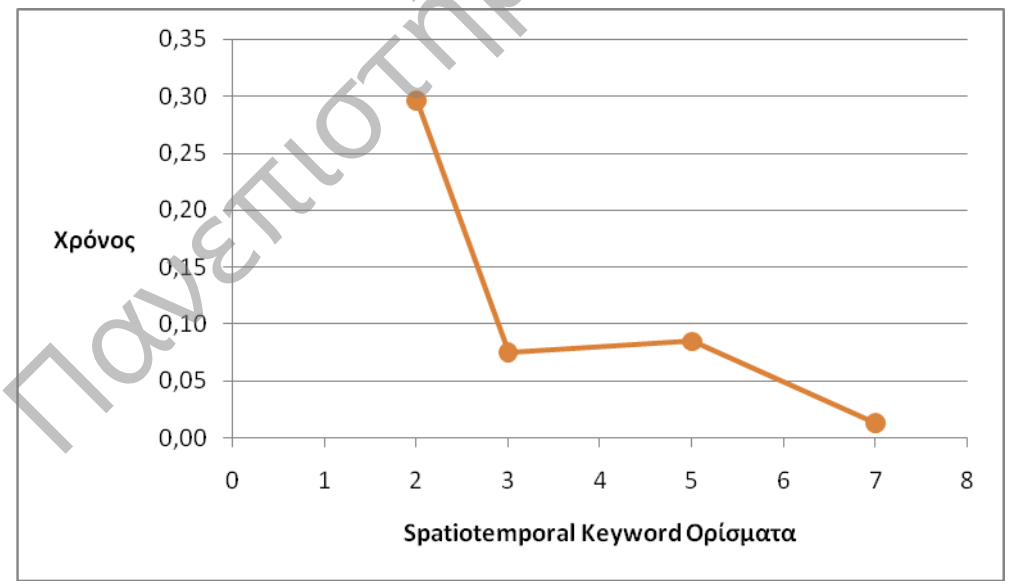
Σχήμα 45: Πίνακας αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης της μεθόδου αναζήτησης για 1.450.739 εγγραφές για τον Αλγόριθμο ESTB (Hardware Set B)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 44 (Hardware Set A) και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα.



Σχήμα 46: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τον Αλγόριθμο ESTB (Hardware Set A)

Το παρακάτω Σχήμα έχει δημιουργηθεί από τον πίνακα του Σχήματος 45 (Hardware Set A) και παρουσιάζει με τη μορφή διαγράμματος τα δεδομένα του συγκεκριμένου πίνακα..

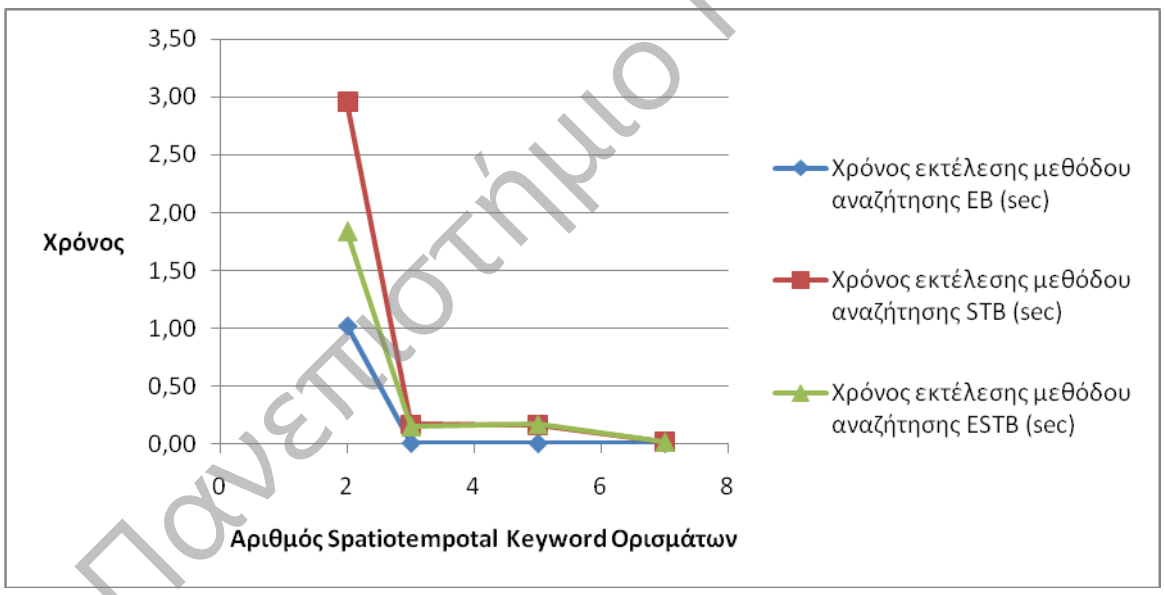


Σχήμα 47: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τον Αλγόριθμο ESTB (Hardware Set B)

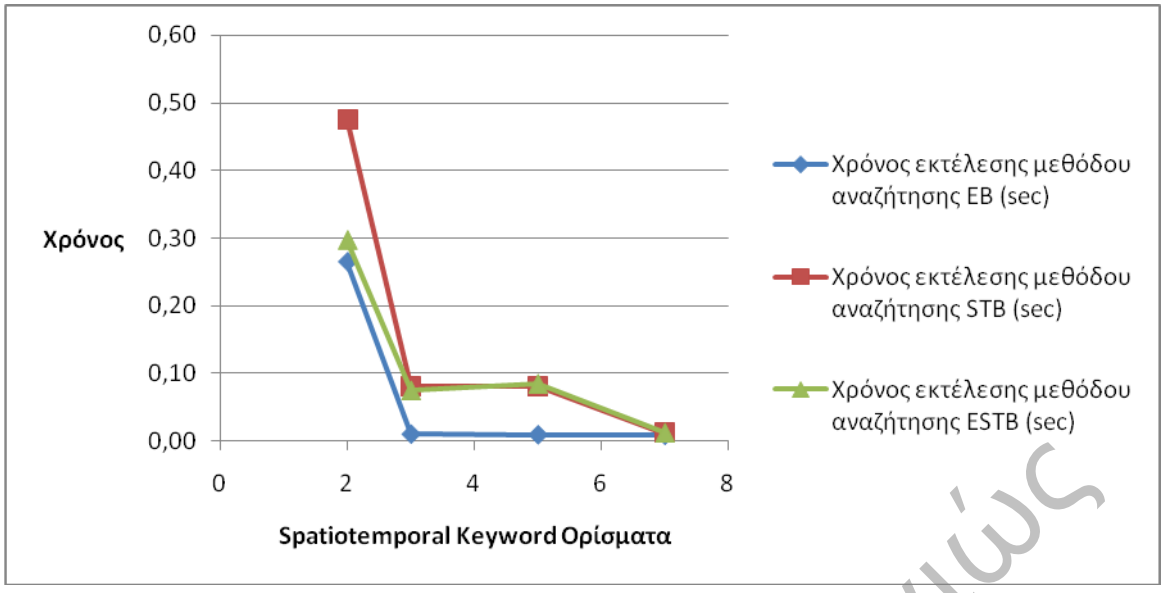
Από τα δεδομένα των πινάκων του Σχημάτων 44 και 45 και των αντίστοιχων διαγραμμάτων των Σχημάτων 46 και 47 μπορούμε να παρατηρήσουμε ότι ο Αλγόριθμος ESTB έχει παρόμοια συμπεριφορά με τον Αλγόριθμο STB. Ειδικότερα, όπως φαίνεται στα δύο παραπάνω διαγράμματα, ο Αλγόριθμος ESTB έχει συγκριτικά καλύτερη απόδοση όταν το query αποτελείται από μεγάλο αριθμό από spatiotemporal keyword ορίσματα. Η σταδιακή αύξηση του αριθμού των ορισμάτων έχει ως αποτέλεσμα και σε αυτήν την περίπτωση τη μείωση του χρόνου εκτέλεσης του ερωτήματος.

4.5.4 Συγκριτικά διαγράμματα απόδοσης των αλγόριθμων

Στη συνέχεια χρησιμοποιώντας τα δεδομένα της προηγούμενης υποενότητας ακολουθούν διαγράμματα που παρουσιάζουν τη συγκριτική απόδοση των αλγόριθμων στην εκτέλεση των ερωτημάτων.



Σχήμα 48: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τους τρεις αλγόριθμους για 1.450.739 εγγραφές (Hardware Set A)



Σχήμα 49: Συγκριτικό διάγραμμα αριθμού spatiotemporal keyword ορισμάτων και χρόνου εκτέλεσης των μεθόδων αναζήτησης για τους τρεις αλγόριθμους για 1.450.739 εγγραφές (Hardware Set B)

Από τα δεδομένα των διαγραμμάτων των Σχημάτων 48 και 49 μπορούμε να παρατηρήσουμε ότι υπάρχει μία σχετική ομοιογένεια στη διαγραμματική μορφή κάθε αλγόριθμου. Αρχικά παρατηρούμε ότι για μικρό αριθμό ορισμάτων καλύτερη απόδοση παρουσιάζει ο Αλγόριθμος EB και στη συνέχεια ακολουθεί ο Αλγόριθμος ESTB. Ο Αλγόριθμος STB έχει παρόμοιες επιδόσεις με τον Αλγόριθμο ESTB, παρουσιάζοντας μικρή διαφορά στον χρόνο εκτέλεσης των ερωτημάτων. Η απόδοση και των τριών αλγόριθμων βελτιώνεται με την σταδιακή αύξηση του αριθμού των ορισμάτων. Παρουσιάζουν δηλαδή την καλύτερη δυνατή απόδοση στον μεγαλύτερο αριθμό ορισμάτων. Με την σταδιακή αύξηση του αριθμού των ορισμάτων παρατηρούμε ότι και οι τρεις αλγόριθμοι παρουσιάζουν σχετικά ίδια συμπεριφορά όσον αφορά το χρόνο εκτέλεσης των ερωτημάτων.

Εν κατακλείδι, μπορούμε να παρατηρήσουμε ότι ανεξάρτητα από τον αριθμό ορισμάτων ο Αλγόριθμος EB παρουσιάζει γενικότερα καλύτερους χρόνους εκτέλεσης των ερωτημάτων σε σύγκριση με τους άλλους δύο αλγόριθμους. Με την αύξηση του αριθμού των ορισμάτων στα ερωτήματα οι άλλοι δύο αλγόριθμοι έχουν σταδιακά βελτίωση και αυτοί στους χρόνους εκτέλεσης. Ο Αλγόριθμος EB μειονεκτεί στο γεγονός ότι η βάση δεδομένων που δημιουργείται έχει μεγαλύτερο μέγεθος σε σχέση με τους άλλους δύο, ανεξάρτητα από τον αριθμό των εγγραφών που βρίσκονται στη βάση και παρουσιάζει επίσης μεγαλύτερο χρόνο δημιουργίας της βάσης δεδομένων.

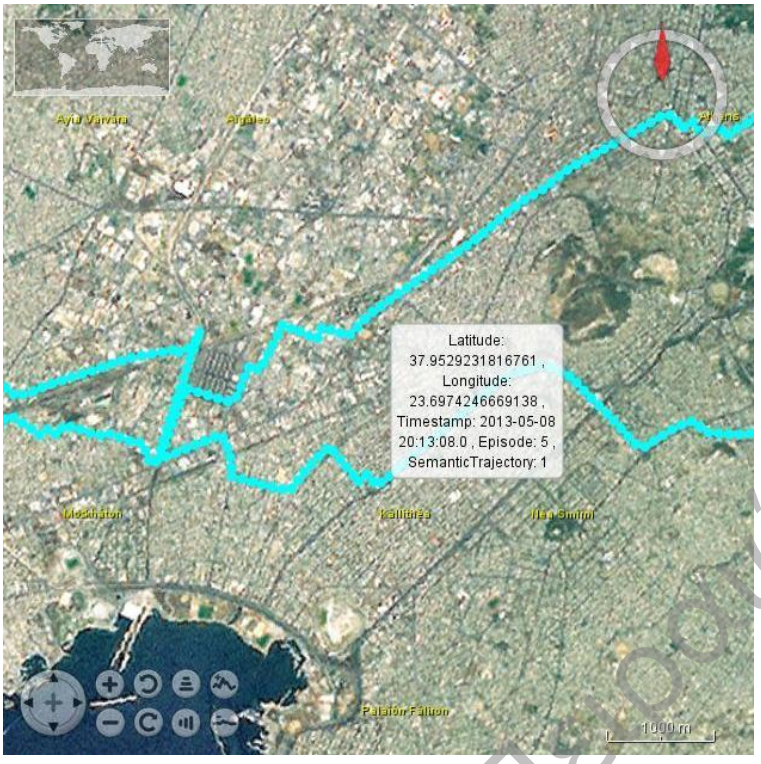
5 Γραφική Διεπαφή

5.1 NASA World Wind API

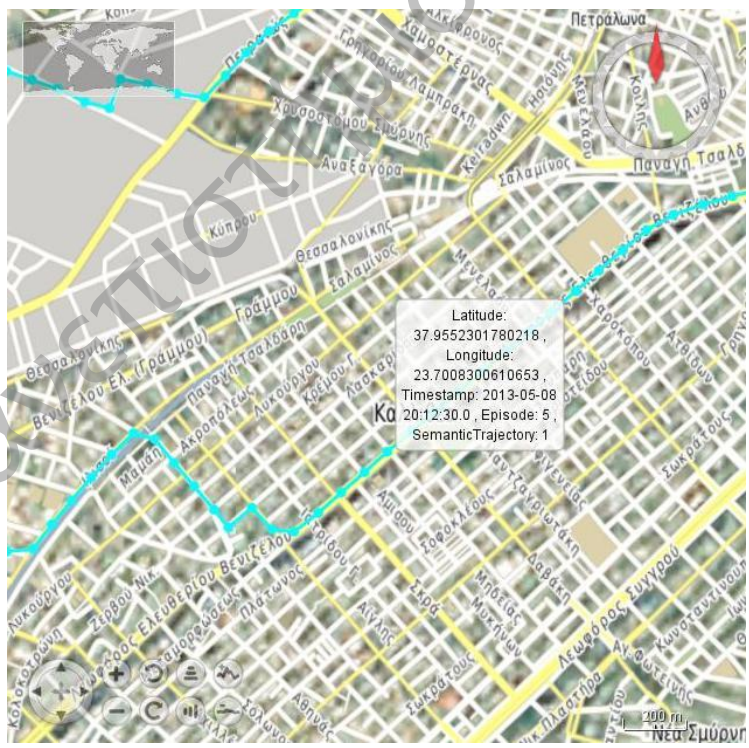
Η οπτική αναπαράσταση των αποτελεσμάτων της αλγοριθμικής αναζήτησης πραγματοποιείται μέσω του NASA World Wind API [37], που αποτελεί μία Java open-source 3D/2D library που χρησιμοποιείται για την γραφική απεικόνιση σε χάρτη χωρικών αντικειμένων. Με το συγκεκριμένο API υπάρχει δυνατότητα χρησιμοποίησης γεωγραφικών δεδομένων (χάρτες) όπως Open Street Map, Bing, MS Virtual Earth, NASA Blue Marble και i-cubed Landsat.

5.2 Γραφική αναπαράσταση των Semantic Trajectories

Για την γραφική αναπαράσταση των Semantic Trajectories δημιουργήθηκε η κλάση «TemporalPosition» που αποτελεί επέκταση της κλάσης «Position» του NASA World Wind API και η κλάση «SemanticGraphQueryDisplay» που αποτελεί η βασική κλάση της γραφικής διεπαφής. Στη συνέχεια ακολουθούν εικόνες από τη γραφική διεπαφή που απεικονίζουν την πληροφορία (Episode ID, SemanticTrajectory ID, Latitude, Longitude, Timestamp) που υπάρχει σε ένα συγκεκριμένο σημείο ενός SemanticTrajectory.



Σχήμα 50: Γραφική απεικόνιση της πληροφορίας ενός σημείου από ένα SemanticTrajectory



Σχήμα 51: Γραφική απεικόνιση της πληροφορίας ενός σημείου από ένα SemanticTrajectory με χρήση OpenStreepMap data

Στη συνέχεια ακολουθεί συνολική εικόνα από τη γραφική διεπαφή. Στο αριστερό μέρος της διεπαφής μπορούμε να διακρίνουμε επιλογές χρήσης διαφορετικών χαρτών όπως Open Street Map, Bing ή MS Virtual Earth. Στο κεντρικό μέρος της διεπαφής υπάρχει η απεικόνιση των σημασιολογικών τροχιών στο χάρτη με δυνατότητα εστίασης (zoom) σε μικρότερη ή μεγαλύτερη κλίμακα, εμφάνισης του σημείου που βρισκόμαστε στο σύνολο του χάρτη καθώς και πυξίδας για έλεγχο ορθότητας του προσανατολισμού. Στο δεξιό μέρος της διεπαφής υπάρχον επιλογές για πραγματοποίηση επερωτήσεων σε Neo4j βάση που θα αναλυθούν στην επόμενη ενότητα.

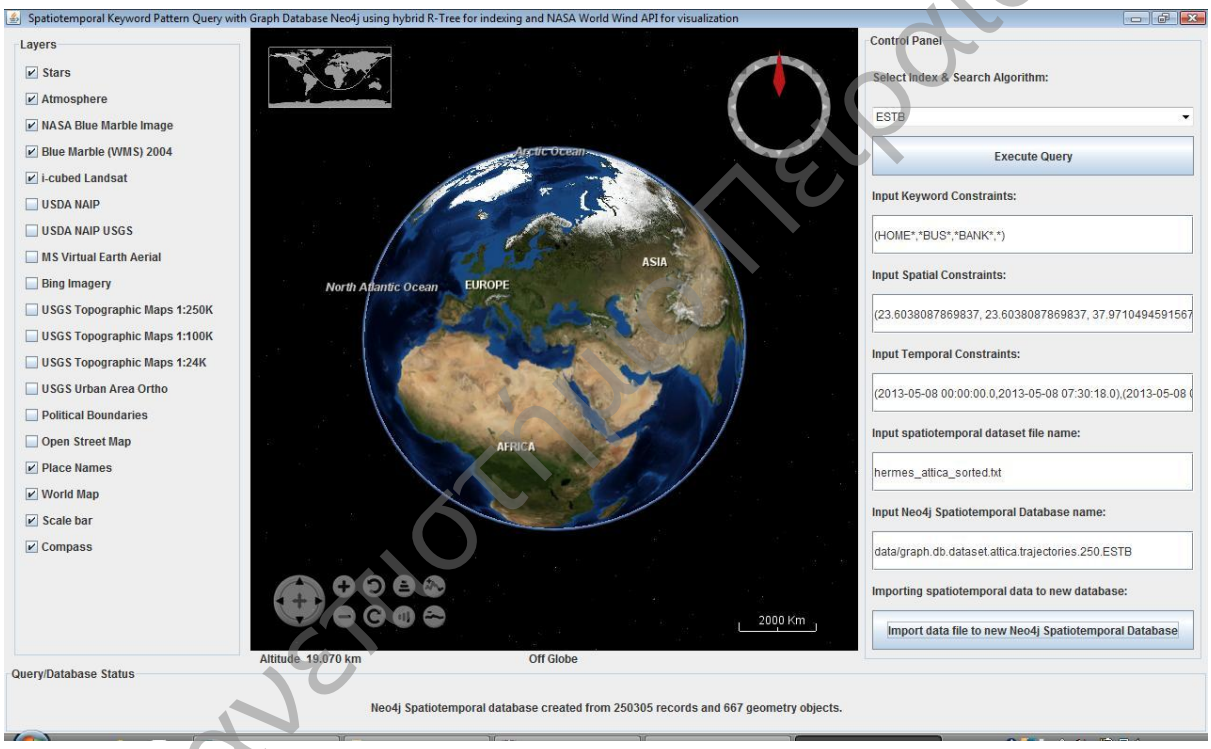
Σχήμα 52: Γραφική διεπαφή αναζήτησης

5.3 Πραγματοποίηση επερωτήσεων στη βάση μέσω της γραφικής διεπαφής

Για την πραγματοποίηση επερωτήσεων χρησιμοποιώντας τους αλγόριθμους που αναλύθηκαν στις προηγούμενες ενότητες, η διεπαφή δίνει τη δυνατότητα εισαγωγής των κατάλληλων χωροχρονικών και λεκτικών περιορισμών ως ορίσματα του spatiotemporal keyword pattern query. Η διεπαφή περιλαμβάνει επίσης λειτουργίες για την ανάγνωση ενός αρχείου που περιέχει χωροχρονικά και σημασιολογικά δεδομένα και την αντίστοιχη δημιουργία της κατάλληλης Neo4j βάσης δεδομένων.

5.3.1 Ανάγνωση του αρχείου δεδομένων και δημιουργία της βάσης

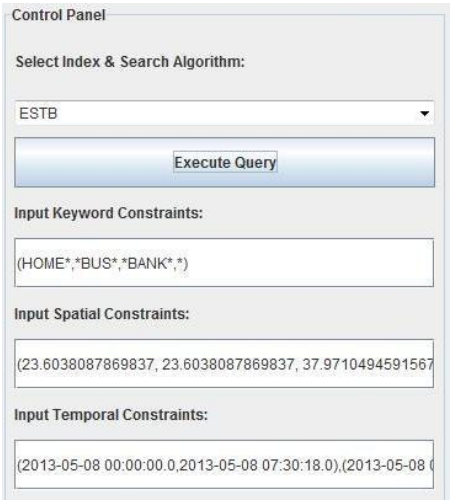
Για τη δημιουργία της Neo4j βάσης δεδομένων χρειάζεται να συμπληρωθούν στα κατάλληλα πεδία, το όνομα του αρχείου που περιέχει τα δεδομένα και η ονομασία της βάσης. Το αρχείο πρέπει να περιέχει δεδομένα στη μορφή που έχει αναφερθεί στην ενότητα 4.1. Στη συνέχεια ακολουθεί κατάλληλο screenshot που δείχνει την εισαγωγή των στοιχείων ενός αρχείου στη βάση και την εμφάνιση κατάλληλου μηνύματος που δείχνει τον αριθμό των εγγραφών που υπήρχαν στο αρχείο και έγιναν εισαγωγή στη βάση καθώς και τον αριθμό των γεωμετρικών αντικειμένων που δημιουργήθηκαν.



Σχήμα 53: Δημιουργία Neo4j βάσης δεδομένων με 250.305 εγγραφές (Dataset 1)

5.3.2 Spatiotemporal Keyword Pattern Query

Για την πραγματοποίηση επερωτήσεων στη βάση, χρειάζεται να συμπληρωθούν τα κατάλληλα πεδία στη διεπαφή που αφορούν την μέθοδο δημιουργίας του ευρετηρίου, τον αλγόριθμο αναζήτησης καθώς και χωροχρονικούς και λεκτικούς περιορισμούς για τα προς αναζήτηση σημασιολογικά αντικείμενα.



Σχήμα 54: Spatiotemporal Keyword Pattern Query παράμετροι αναζήτησης

Ειδικότερα, υπάρχουν 3 κατάλληλα πεδία για την επιλογή μέθοδος δημιουργίας του ευρετηρίου και του αλγόριθμου αναζήτησης:

-EB
-STB
-ESTB

Αντίστοιχα, υπάρχουν 3 κατάλληλα πεδία για τη συμπλήρωση των περιορισμών του query:

-Λεκτικοί περιορισμοί
-Χωρικοί περιορισμοί
-Χρονικοί περιορισμοί

Στη συνέχεια ακολουθούν παραδείγματα συμπλήρωσης των περιορισμών ανά αλγόριθμο.

5.3.2.1 Παράδειγμα αναζήτησης για τους αλγόριθμους STB και ESTB

-Λεκτικός περιορισμός:
(HOME*,*BUS*,*BANK*,*)

Ο συγκεκριμένος περιορισμός αφορά αντικείμενα SemanticTrajectory που να έχουν episode που να περιέχει τον πρώτο λεκτικό περιορισμό, στη συνέχεια να ακολουθεί ένα episode που να περιέχει το δεύτερο λεκτικό περιορισμό, ένα episode που να περιέχει το τρίτο λεκτικό περιορισμό και στη συνέχεια οποιοσδήποτε αριθμός από episodes χωρίς λεκτικούς περιορισμούς.

-Χωρικός περιορισμός:
(23.6038087869837, 23.6038087869837, 37.9710494591567, 37.9710494591567), (23.6046204134222, 23.7481714095042, 37.953217164859, 37.9885534794328) ,(23.7485917480656, 23.7485917480656, 37.9888518026881, 37.9888518026881), (*,*,*,*)

Ο συγκεκριμένος περιορισμός αφορά αντικείμενα SemanticTrajectory που να έχουν episode που να περιέχει τον πρώτο χωρικό περιορισμό, στη συνέχεια να ακολουθεί ένα episode που να περιέχει το δεύτερο χωρικό περιορισμό, ένα episode που να περιέχει το τρίτο χωρικό περιορισμό και στη συνέχεια οποιοσδήποτε αριθμός από episodes χωρίς χωρικούς περιορισμούς.

-Χρονικός περιορισμός:
(2013-05-08 00:00:00.0,2013-05-08 07:30:18.0),(2013-05-08 08:20:14.0,2013-05-08 08:59:46.0),(2013-05-08 09:00:00.0,2013-05-08 17:00:00.0),(*,*)

Ο συγκεκριμένος περιορισμός αφορά αντικείμενα SemanticTrajectory που να έχουν episode που να περιέχει τον πρώτο χρονικό περιορισμό, στη συνέχεια να ακολουθεί ένα episode που να περιέχει το δεύτερο χρονικό περιορισμό, ένα episode που να περιέχει το τρίτο χρονικό περιορισμό και στη συνέχεια οποιοσδήποτε αριθμός από episodes χωρίς χρονικούς περιορισμούς.

5.3.2.2 Παράδειγμα αναζήτησης για τον αλγόριθμο EB

-Λεκτικός περιορισμός:
(HOME,BUS,BANK)

Ο συγκεκριμένος περιορισμός αφορά αντικείμενα SemanticTrajectory που να έχουν episode που να περιέχει τον πρώτο λεκτικό περιορισμό, στη συνέχεια να ακολουθεί ένα episode που να περιέχει το δεύτερο λεκτικό περιορισμό, ένα episode που να περιέχει το τρίτο λεκτικό περιορισμό και στη συνέχεια ή πριν οποιοσδήποτε αριθμός από episodes χωρίς λεκτικούς περιορισμούς.

-Χωρικός περιορισμός:
(23.6038087869837, 23.6038087869837, 37.9710494591567, 37.9710494591567), (23.6046204134222, 23.7481714095042, 37.953217164859, 37.9885534794328) ,(23.7485917480656, 23.7485917480656, 37.9888518026881, 37.9888518026881)

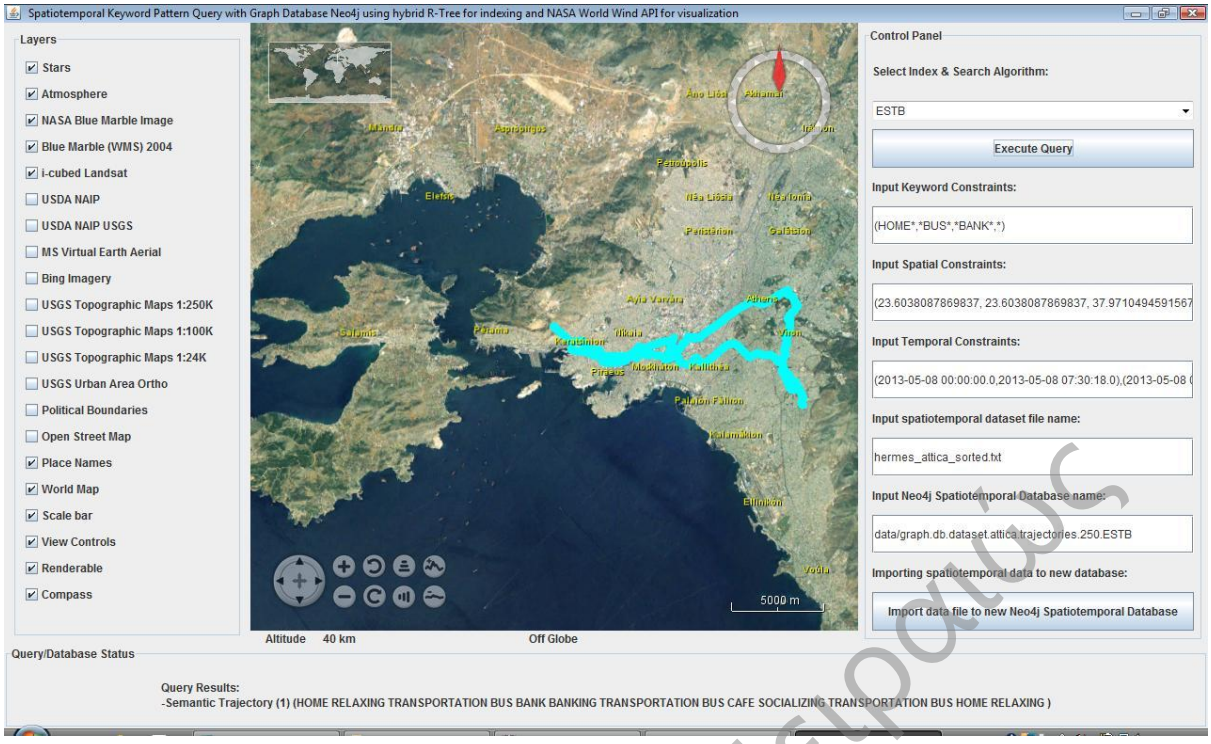
Ο συγκεκριμένος περιορισμός αφορά αντικείμενα SemanticTrajectory που να έχουν episode που να περιέχει τον πρώτο χωρικό περιορισμό, στη συνέχεια να ακολουθεί ένα episode που να περιέχει το δεύτερο χωρικό περιορισμό, ένα episode που να περιέχει το τρίτο χωρικό περιορισμό και στη συνέχεια ή πριν οποιοσδήποτε αριθμός από episodes χωρίς χωρικούς περιορισμούς.

-Χρονικός περιορισμός:
(2013-05-08 00:00:00.0,2013-05-08 07:30:18.0),(2013-05-08 08:20:14.0,2013-05-08 08:59:46.0),(2013-05-08 09:00:00.0,2013-05-08 17:00:00.0)

Ο συγκεκριμένος περιορισμός αφορά αντικείμενα SemanticTrajectory που να έχουν episode που να περιέχει τον πρώτο χρονικό περιορισμό, στη συνέχεια να ακολουθεί ένα episode που να περιέχει το δεύτερο χρονικό περιορισμό, ένα episode που να περιέχει το τρίτο χρονικό περιορισμό και στη συνέχεια ή πριν οποιοσδήποτε αριθμός από episodes χωρίς χρονικούς περιορισμούς.

Η συμπλήρωση των 3 πεδίων έχει ως αποτέλεσμα την πραγματοποίηση ενός spatiotemporal keyword pattern query χρησιμοποιώντας τους αλγόριθμους που αναλύθηκαν σε προηγούμενη ενότητα.

Αν η αναζήτηση έχει ως αποτέλεσμα την εύρεση ενός Semantic Trajectory που ικανοποιεί τα κριτήρια που έχουν δοθεί, τότε στο σχετικό πλαίσιο της γραφικής διεπαφής που υπάρχει ο παγκόσμιος χάρτης σε 3D μορφή, γίνεται με κατάλληλο animation zoom σε 2D χάρτη εμφανίζοντας την γεωγραφική περιοχή που υπάρχει το συγκεκριμένο Semantic Trajectory. Κάθε Semantic Trajectory που ανήκει στα αποτελέσματα της αναζήτησης απεικονίζεται με διαφορετικό χρώμα στο χάρτη. Αντίστοιχα, εμφανίζονται στο πλαίσιο αποτελεσμάτων πληροφορίες για τα αποτελέσματα της αναζήτησης και τον αριθμό των trajectories που ικανοποιούν τα κριτήρια της αναζήτησης. Στη συνέχεια ακολουθεί screenshot που παρουσιάζει το αποτέλεσμα αναζήτησης από μία αναζήτηση.



Σχήμα 55: Αποτέλεσμα αναζήτησης

6 Συμπεράσματα

Οι αυξανόμενες ανάγκες διαχείρισης μεγάλου όγκου δεδομένων σχετικών με χωροχρονικές και λεκτικές πληροφορίες λόγω της ραγδαίας χρήσης συσκευών με GPS δυνατότητες, όπως κινητά τηλέφωνα ή tablets, έχει οδηγήσει στην ανάγκη δημιουργίας αλγοριθμικών τρόπων εξαγωγής από αυτά χρησίμων πληροφοριών και συμπερασμάτων. Οι αλγόριθμοι και τα συστήματα και υπηρεσίες που διαχειρίζονται τα ανωτέρα δεδομένα πρέπει να ανταποκρίνονται στη μορφή των συγκεκριμένων δεδομένων αλλά και στον ταχύτατα αυξανόμενο όγκο τους.

Τα NoSQL συστήματα και βάσεις δεδομένων χρησιμοποιούν, όπως αναφέρθηκε και νωρίτερα, μεθόδους και μοντέλα αποθήκευσης, ανάκτησης και διαχείρισης δεδομένων που δεν βασίζονται στις παραδοσιακές αρχιτεκτονικές των σχεσιακών συστημάτων βάσεων δεδομένων (RDBMS) και χρησιμοποιούνται σε μεγάλη έκταση στη διαχείριση μεγάλου όγκου δεδομένων (Big Data) και σε real-time web εφαρμογές.

Η βάση δεδομένων που χρησιμοποιήθηκε για την αποθήκευση και διαχείριση των δεδομένων των Semantic Trajectories (Neo4j) αποτελεί μία NoSQL Graph database που μπορεί να χρησιμοποιηθεί ιδιαίτερα αποδοτικά σε δεδομένα που μπορούν να αναπαρασταθούν σε εξατομικευμένες μορφές γράφων ανάλογα με τη υπηρεσία ή τα δεδομένα.

Χαρακτηριστικό παράδειγμα συγκεκριμένων δεδομένων είναι οι σημασιολογικές τροχιές και τα χωροχρονικά και λεκτικά δεδομένα που αυτές περιέχουν. Η χρησιμοποίηση της συγκεκριμένης βάσης δεδομένων για την αντιμετώπιση των spatiotemporal keyword pattern queries δείχνει ότι τα graph συστήματα βάσεων δεδομένων και ειδικότερα το Neo4j μπορεί να ανταπεξέλθει ικανοποιητικά στη διαχείριση τους και στην πραγματοποίηση επερωτήσεων σε αυτά.

Το γεγονός αυτό υποδεικνύει ότι είναι δυνατή η χρησιμοποίηση της ανωτέρω βάσης δεδομένων και για την πραγματοποίηση διαφορετικών ερωτημάτων εκτός από το τη συγκεκριμένη κατηγορία ερωτημάτων. Με άλλα λόγια, μία μελλοντική εργασία θα μπορούσε να ασχοληθεί με την πραγματοποίηση data mining επερωτήσεων άλλων μορφών (όπως τα boolean range queries, τα boolean kNN queries ή τα top-k kNN queries) σε μία βάση δεδομένων από γράφους.

7 Βιβλιογραφία

[1] N. Pelekis, Y. Theodoridis, D.Janssens: “On the management and analysis of our LifeSteps”, SIGKDD Explorations, 15(1):23-32, ACM, 2013

[2] Antonin Guttman: “R-trees: a dynamic index structure for spatial searching”, SIGMOD '84, Proceedings of the ACM SIGMOD international conference on Management of data, pages 47 – 57, 1984

[3] Pfoser D., Jensen C.S., Theodoridis Y. : “Novel Approaches to the Indexing of Moving Object Trajectories”, Proceedings of the VLDB, 2000

[4] Ralf Hartmut Guting, Fabio Valdes, Maria Luisa Damiamni: “Symbolic Trajectories”, Informatik Berichte, 369- 2013

[5] R. H. Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, M. Vazirgiannis: “A foundation for representing and querying moving objects”, ACM Trans. Database Syst., 25(1):1–42, 2000

[6] S Dieker, RH Guting: “Plug and play with query algebras: SECONDO, a generic DBMS development environment”, Database Engineering and Applications Symposium, 2000

[7] Harry R. Lewis, Χρίστος Χ. Παπαδημητρίου: “Στοιχεία Θεωρίας Υπολογισμού”, Εκδόσεις Κριτική ΑΕ, 2005

[8] A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman: “Compilers: Principles, Techniques, and Tools (2nd Edition)”, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006

[9] Lisi Chen, Gao Cong, Christian S. Jensen, Dingming Wu: “Spatial Keyword Query Processing: An Exprerimental Evaluation”, Proceedings of the VLDB Endowment, 2013

[10] Panagiotis Bouros, Shen Ge, Nikos Mamoulis: “Spatio-Textual Similarity Joins”, Proceedings of the VLDB Endowment, Vol. 6, No. 1, 2012

[11] Y.Zhou, X. Xie, C. Wang, Y. Gong, W.-Y. Ma: “Hybrid index structures for location-based web search”, In CIKM, pages 155–162, 2005

[12] R.Hariharan, B. Hore, C. Li, S. Mehrotra: “Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems”, In SSDBM, page 16, 2007

[13] I. D. Felipe, V. Hristidis, N. Rishe: “Keyword search on spatial databases”, In ICDE, pages 656–665, 2008

[14] C. Faloutsos, S. Christodoulakis: “Signature files: An access method for documents and its analytical performance evaluation”, ACM Trans. Inf. Syst., 2(4):267–288, 1984

[15] A. Cary, O. Wolfson, N. Rishe: “Efficient and scalable method for processing top-k spatial boolean queries”, In SSDBM, pages 87–95, 2010

[16] G. Cong, C. S. Jensen, D. Wu, “Efficient retrieval of the top-k most relevant spatial web objects”, PVLDB, 2(1):337–348, 2009

[17] D. Wu, G. Cong, C. S. Jensen: “A framework for efficient spatial web object retrieval”, VLDBJ, 21(6):797–822, 2012

[18] Xin Cao, Gao Cong, Christian S. Jensen, Jun Jie Ng, Beng Chin Oooi, Nhan-Tue Phan, Dingming Wu: “SWORS: A system for the Efficient Retrieval of Relevant Spatial Web Objects”, Proceedings of the VLDB Endowment, Vol. 5, No. 12, 2012

[19] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang: “Ir-tree: An efficient index for geographic document search”, IEEE TKDE, 23(4):585–599, 2011

[20] D. Wu, M. L. Yiu, G. Cong, C. S. Jensen: “Joint top-k spatial keyword query processing”, IEEE TKDE, 24(10):1889–1903, 2012

[21] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, K. Norvag: “Efficient processing of top-k spatial keyword queries”, In SSTD, pages 205–222, 2011

[22] S. Vaid, C. B. Jones, H. Joho, M. Sanderson: “Spatio-textual indexing for geographical search on the web”, In SSTD, pages 218–235, 2005

[23] A. Khodaei, C. Shahabi, C. Li.: “Hybrid indexing and seamless ranking of spatial and textual features of web documents”, In DEXA, pages 450–466, 2010

[24] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, T. Suel: “Text vs. space: efficient geo-search query processing”, In CIKM, pages 423–432, 2011

[25] Rick Cattell: “Scalable SQL and NoSQL Data Stores”, SIGMOD Record, 2010 (Vol. 39, No. 4).

[26] Christof Strauch: “NoSQL Databases”, Selected Topics on Software-Technology Ultra-Large Scale Sites, Stuttgart Media University, 2011.

[27] Neo4j Documentation - http://docs.neo4j.org/

[28] The Neo Database – A Technology Introduction, http://dist.neo4j.org/neo-technology-introduction.pdf, 2006

[29] Neo4j Spatial - http://www.neo4j.org/develop/spatial

[30] JTS Topology Suite - http://www.vividsolutions.com/jts/JTSHome.htm

[31] N. Pelekis, C. Ntrigkogias, P. Tampakis, S. Sideridis, Y. Theodoridis: “Hermoupolis: A Trajectory Generator for Simulating Generalized Mobility Patterns”, demo paper, Proceedings of the European Conference on Machine Learning / Principles and Practice of Knowledge Discovery in Databases, ECML/PKDD'13, Prague, Czech Republic, Springer, 2013

[32] Apache Lucene - http://lucene.apache.org/core/

[33] Νικηφοράκη Ελένη: “Ανάκτηση κειμένων από μεγάλες συλλογές εγγράφων με χρήση προηγμένων τελεστών επερώτησης”, Μεταπτυχιακή Διατριβή, Πανεπιστήμιο Πειραιώς, 2012

[34] What is an intuitive description of how Lucene works? - http://www.quora.com/What-is-an-intuitive-description-of-how-Lucene-works

[35] Carpenter, Bob, Mitzi Morris, Breck Baldwin: “Text Processing with Java 6”, 2012

[36] Θεόδωρος Καλαμπούκης, Αντωνία Κυριακοπούλου: “Σημειώσεις Μαθήματος Συστήματα Ανάκτησης Πληροφοριών”, Οικονομικό Πανεπιστήμιο Αθηνών, 2010

[37] NASA Word Wind - http://worldwind.arc.nasa.gov/features.html

8 Παραρτήματα

8.1 Δομή προγραμματιστικής υλοποίησης

Τα σημαντικότερα πακέτα κλάσεων της υλοποίησης (ονομασία project STKPGraphQuery) είναι τα παρακάτω:

-Neo4jSpatiotemporal που αφορά τη επέκταση της Neo4j Spatial library για τη διαχείριση σημασιολογικών τροχιών.

-SemanticJTS που αφορά τη επέκταση της JTS library για τη διαχείριση σημασιολογικών τροχιών.

-SemanticGraphQuery που αφορά το πακέτο κλάσεων για εισαγωγή δεδομένων στη βάση δεδομένων, δημιουργία καινούριας βάσης δεδομένων και την πραγματοποίηση επερωτήσεων στη βάση. Μέσω της κλάσης Main του συγκεκριμένου πακέτου είναι εφικτή η πραγματοποίηση επερωτήσεων στο σύνολο των datasets που χρησιμοποιήθηκαν για τις συγκριτικές δοκιμές των αλγόριθμων καθώς και η δημιουργία καινούριας βάσης δεδομένων. Στη συνέχεια ακολουθεί screenshot από τη χρησιμοποίηση της συγκεκριμένης κλάσης.

```
Choose index and search algorithm:
1. Episode-based Hybrid 3D R-Tree Search
2. Semantic-trajectory-based Hybrid 3D R-Tree Search
3. Enhanced Semantic-trajectory-based Hybrid 3D R-Tree Search

3
Choose dataset:
1. Dataset 1 (250.305 records)
2. Dataset 2 (500.913 records)
3. Dataset 3 (750.167 records)
4. Dataset 4 (1.000.038 records)
5. Dataset 5 (1.250.133 records)
6. Dataset 6 (1.450.739 records)
7. Create Dataset from file
6
Choose search:
1. Example search with 1 spatiotemporal keyword pattern
2. Example search with 2 spatiotemporal keyword patterns
3. Example search with 3 spatiotemporal keyword patterns
4. Example search with 5 spatiotemporal keyword patterns
5. Example search with 7 spatiotemporal keyword patterns
6. Search with custom spatiotemporal keyword pattern
5

Duration: 1.00468428 seconds.
Results size: 1

1000  HOME RELAXING(1) TRANSPORTATION BUS(2) BANK BANKING(3) TRANSPORTATION BUS(4) CAFE SOCIALIZING(5)
```

Σχήμα 56: Εκτέλεση επερώτησης στη βάση δεδομένων μέσω της κλάσης Main του πακέτου SemanticGraphQuery

-Visualization που αφορά την υλοποίηση της γραφικής διεπαφής για την πραγματοποίηση επερωτήσεων σε βάση δεδομένων. Μέσω της κλάσης Main του συγκεκριμένου πακέτου είναι δυνατή η χρησιμοποίηση της γραφικής διεπαφής για την πραγματοποίηση επερωτήσεων σε βάση δεδομένων καθώς και η δημιουργία καινούριας βάσης δεδομένων, όπως αναλύθηκε σε προηγούμενη ενότητα.

8.2 Αναλυτική περιγραφή κλάσεων προγραμματιστικής υλοποίησης

8.2.1 Υλοποίηση βασικών κλάσεων προβλήματος

Στη συνέχεια ακολουθεί σύντομη περιγραφή της υλοποίησης των βασικότερων κλάσεων και μεθόδων των ορισμών του προβλήματος. Η υλοποίηση βασίστηκε σε επέκταση βασικών κλάσεων της JTS library.

8.2.1.1 Κλάση RawSubTrajectory

Η κλάση RawSubTrajectory αποτελεί κλάση για την αναπαράσταση ενός linestring στο χωροχρόνο και αποτελεί επέκταση της κλάσης LineString της JTS. Περιέχει τις παρακάτω βασικές μεθόδους:

- String getTrajectoryID(): Επιστρέφει το ID του RawSubTrajectory.
- void setTrajectoryID(String ID): Καθορίζει το ID του RawSubTrajectory.
- String getObjectID(): Επιστρέφει το ID του αντικειμένου που πραγματοποίησε το RawSubTrajectory.
- void setObjectID(String ID): Καθορίζει το ID του αντικειμένου που πραγματοποίησε το RawSubTrajectory.
- String toString(): Επιστρέφει σε κατάλληλη συμβολοσειρά το σύνολο των συντεταγμένων του RawSubTrajectory.
- TemporalCoordinate[] getTemporalCoordinates(): Επιστρέφει σε πίνακα το σύνολο των συντεταγμένων του RawSubTrajectory.

8.2.1.2 Κλάση RawTrajectory

Η κλάση RawTrajectory αποτελεί κλάση για την αναπαράσταση ενός linestring στο χωροχρόνο και αναπαριστά ένα σύνολο από αντικείμενα RawSubTrajectory. Περιέχει τις παρακάτω βασικές μεθόδους:

- RawSubTrajectory[] getRawSubTrajectories (): Επιστρέφει σε πίνακα το σύνολο των αντικειμένων RawSubTrajectory.
- int getTrajectoriesLength(): Επιστρέφει τον αριθμό των αντικειμένων RawSubTrajectory που περιέχει το αντικείμενο RawTrajectory.

8.2.1.3 Κλάση MBB

Η επέκταση της κλάσης Envelope της JTS με ονομασία MBB αποτελεί κλάση για την αναπαράσταση ενός bounding box στο χώρο. Περιέχει τις παρακάτω βασικές μεθόδους:

- void temporalExpandToInclude(MBB other): Αυξάνει τις διαστάσεις του MBB προκειμένου να περιέχει το MBB other.
- MBB getEnvelopeInterval(): Επιστρέφει το MBB που περιβάλλει το συγκεκριμένο αντικείμενο episode.
- double[] getBbox(): Επιστρέφει σε πίνακα τις συντεταγμένες του MBB.
- String toString(): Επιστρέφει σε κατάλληλη συμβολοσειρά το σύνολο των συντεταγμένων του MBB.

8.2.1.4 Κλάση Episode

Η επέκταση της κλάσης RawSubTrajectory με ονομασία Episode περιέχει κατάλληλους κατασκευαστές και μεθόδους για τη δημιουργία του αντικειμένου με επιπρόσθετη σημασιολογική πληροφορία. Περιέχει τις παρακάτω βασικές μεθόδους:

- MBB computeEnvelopeInterval(): Δημιουργεί το MBB που περιβάλλει το συγκεκριμένο αντικείμενο episode.
- MBB getEnvelopeInterval(): Επιστρέφει το MBB που περιβάλλει το συγκεκριμένο αντικείμενο episode.
- double[] getXArray(): Επιστρέφει σε πίνακα τις (x) συντεταγμένες.
- double[] getYArray(): Επιστρέφει σε πίνακα τις (y) συντεταγμένες.
- long[] getTimestampArray: Επιστρέφει σε πίνακα τα timestamps που περιέχει το episode.
- String getID(): Επιστρέφει το ID του episode.
- void setID(String ID): Καθορίζει το ID του episode.
- String getNextID(): Επιστρέφει το ID του επόμενου episode.

- void setNextID(String ID): Καθορίζει το ID του επόμενου episode.
- String getTrajectoryID(): Επιστρέφει το trajectory ID του episode.
- void setTrajectoryID(String ID): Καθορίζει το trajectory ID του episode.
- String getObjectID(): Επιστρέφει το Object ID του episode.
- void setObject ID(String ID): Καθορίζει το Object ID του episode.
- String getFlag(): Επιστρέφει το flag του episode.
- void setFlag(String flag): Καθορίζει το flag του episode.
- String getTags(): Επιστρέφει τα tags του episode.
- void setTags(String tags): Καθορίζει τα tags του episode.
- String toString(): Επιστρέφει σε κατάλληλη συμβολοσειρά το σύνολο των συντεταγμένων του Episode.

8.2.1.5 Κλάση SemanticTrajectory

Η συγκεκριμένη κλάση χρησιμοποιείται για την αναπαράσταση των σημασιολογικών τροχιών και περιέχει τις παρακάτω βασικές μεθόδους:

- Episode[] getEpisodes(): Επιστρέφει σε πίνακα το σύνολο των episodes που αποτελείται ένα αντικείμενο Semantic Trajectory.
- String getTrajectoryID(): Επιστρέφει το trajectory ID.
- void setTrajectoryID(String ID): Καθορίζει το trajectory ID.
- MBB computeEnvelopeInterval(): Δημιουργεί το MBB που περιβάλλει το συγκεκριμένο αντικείμενο SemanticTrajectory.
- MBB getEnvelopeInterval(): Επιστρέφει το MBB που περιβάλλει το συγκεκριμένο αντικείμενο SemanticTrajectory.
- double[] getMBB(): Επιστρέφει σε πίνακα τις συντεταγμένες του MBB που περιβάλλει το συγκεκριμένο αντικείμενο SemanticTrajectory.
- int getTrajectoryLength(): Επιστρέφει τον αριθμό των episodes που αποτελείται το SemanticTrajectory.
- String toString(): Επιστρέφει σε κατάλληλη συμβολοσειρά το σύνολο των episodes (tags) της σημασιολογικής τροχιάς.
- String getObjectID(): Επιστρέφει το Object ID του trajectory.
- void setObject ID(String ID): Καθορίζει το Object ID του trajectory.
- String[] getTags(): Επιστρέφει σε πίνακα τα tags των episodes που αποτελείται το SemanticTrajectory.
- ArrayList<TemporalPosition> getTemporalPositions(): Επιστρέφει σε ArrayList τις συντεταγμένες του SemanticTrajectory για την γραφική αναπαράσταση μέσω του NASA World Wind API.

8.2.2 Ανάλυση κλάσεων των Hybrid 3D R-Trees

Η υλοποίηση των ευρετηρίων στη βάση δεδομένων Neo4j βασίστηκε στο πακέτο κλάσεων «org.neo4j.collections.rtree» του Neo4j και περιλαμβάνει τις παρακάτω βασικές κλάσεις:

- IndexEnvelope
- IndexEnvelopeDecoder
- Listener
- NullListener
- Hybrid3DRTreeIndex
- RTreeRelationshipTypes
- SpatialTemporalIndexReader
- SpatialTemporalIndexRecordCounter
- SpatialTemporalIndexVisitor
- SpatialTemporalIndexWriter
- TemporalSearchFilter

Για την αποθήκευση χωροχρονικής και λεκτικής πληροφορίας το πακέτο των κλάσεων που αφορά την προγραμματιστική υλοποίηση των R-Δέντρων τροποποιήθηκε προκειμένου να περιλαμβάνει και την αποθήκευση και διαχείριση χρονικής και λεκτικής πληροφορίας.

Η βασική κλάση του πακέτου στην οποία υλοποιείται η δημιουργία του R-δέντρου είναι η Hybrid3DRTreeIndex. Η δημιουργία του δέντρου περιλαμβάνει 2 στάδια. Αρχικά το δέντρο δημιουργείται λαμβάνοντας υπόψη τα χωροχρονικά δεδομένα των σημασιολογικών αντικειμένων. Η δημιουργία δηλαδή γίνεται με την σταδιακή εισαγωγή στο δέντρο των σημασιολογικών αντικειμένων, όπως και σε ένα «κανονικό» R-Tree.

Μετά την ολοκλήρωση της δημιουργίας του δέντρου σύμφωνα με τα χωροχρονικά κριτήρια, για κάθε κόμβο του δέντρου δημιουργείται το αντίστοιχο λεκτικό ευρετήριο που περιλαμβάνει την λεκτική πληροφορία των παιδιών του. Αρχικά ξεκινώντας από την κορυφή του δέντρου συγκεντρώνονται οι κόμβοι που δείχνουν οι ακμές με type «RTREE_CHILD». Για τις συγκεκριμένες ακμές στη συνέχεια, μέσω των relationships «RTREE_REFERENCE» (ή «RTREE_CHILD» αν το δέντρο είναι πολυεπίπεδο») συγκεντρώνονται τα λεκτικά στοιχεία των «παιδιών» των κόμβων για τη δημιουργία του ανεστραμμένου αρχείου του εκάστοτε node που καταλήγει η κάθε ακμή «RTREE_CHILD». Μετά τη δημιουργία των ανεστραμμένων αρχείων για το σύνολο των συγκεκριμένων κόμβων που αποτελούν τα «παιδιά» της ρίζας του

δέντρου επαναλαμβάνεται η ίδια διαδικασία για τη δημιουργία του λεκτικού ευρετηρίου για τη ρίζα του δέντρου.

Στη συνέχεια ακολουθεί αναλυτική περιγραφή των βασικών αντικειμένων και μεθόδων που αποτελούν το πακέτο των κλάσεων του δέντρου.

8.2.2.1 Κλάση IndexEnvelope

Η κλάση IndexEnvelope αποτελεί το Minimum Bounding Box (MBB) που χρησιμοποιείται στο ευρετήριο. Δεν έχει την ονομασία MBB, προς αποφυγή ταύτισης με την κλάση MBB της JTS library που αναλύθηκε σε προηγούμενη ενότητα. Περιλαμβάνει κατασκευαστές για τη δημιουργία του αντικειμένου με ορίσματα πίνακες double με ονομασία min και max, με σκοπό την αποθήκευση σε αυτούς τις τιμές των συντεταγμένων των δύο σημείων που δημιουργούν το bounding box καθώς και έναν πίνακα long[] με ονομασία timestamp με σκοπό την αποθήκευση σε αυτόν των δύο τιμών timestamp των δύο σημείων που προσδιορίζουν το bounding box. Περιλαμβάνονται επίσης και κατάλληλες setters και getters μέθοδοι. Δημιουργήθηκαν επίσης οι παρακάτω μέθοδοι που αποτελούν επεκτάσεις των μεθόδων της αρχικής κλάσης Envelope, με σκοπό να λαμβάνεται υπόψη και η χρονική διάσταση του αντικειμένου. Κάθε μέθοδος αφού πρώτα υλοποιηθεί η χωρική διάσταση της πληροφορίας, στη συνέχεια με επιπρόσθετο κώδικα ελέγχεται η χρονική διάσταση της πληροφορίας. Για παράδειγμα, στη Boolean μέθοδο temporalCovers που ελέγχει αν ένα αντικείμενο Envelope βρίσκεται μέσα σε ένα άλλο αντικείμενο Envelope, αφού πρώτα ελεγχθεί αν το αντικείμενο επικαλύπτεται χωρικά με το άλλο αντικείμενο γίνεται μετά έλεγχος για την χρονική επικάλυψη τους.

-boolean temporalContains(IndexEnvelope other): Ελέγχει αν το συγκεκριμένο αντικείμενο IndexEnvelope περιέχει ένα άλλο αντικείμενο IndexEnvelope. Με τον όρο contains ορίζεται ότι κάθε σημείο των αντικειμένου είναι σημείο του άλλου αντικειμένου και έχουν τουλάχιστον ένα εσωτερικό σημείο κοινό.
-boolean temporalCovers(IndexEnvelope other): Ελέγχει αν το συγκεκριμένο αντικείμενο IndexEnvelope «καλύπτει» ένα άλλο αντικείμενο IndexEnvelope. Με τον όρο covers ορίζεται ότι κάθε σημείο του εξεταζόμενου αντικειμένου είναι κοινό μεταξύ των δύο αντικειμένων. Σε αντίθεση με τη μέθοδο contains πρόκειται για πιο «περιεκτική» μέθοδο καθώς δεν κάνει διαχωρισμό μεταξύ σημείων πάνω στο χωρικό αντικείμενο και στο εσωτερικό του. Πρόκειται δηλαδή για πιο γενική μέθοδο.
-boolean temporalDisjoint(IndexEnvelope other): Ελέγχει αν το συγκεκριμένο αντικείμενο IndexEnvelope είναι disjoint από ένα άλλο αντικείμενο IndexEnvelope. Με τον όρο disjoint ορίζεται ότι τα δύο αντικείμενα δεν έχουν κανένα κοινό σημείο μεταξύ τους. Η μέθοδος disjoint είναι η αντίθετη της μεθόδου intersects που θα αναλυθεί στη συνέχεια.

-boolean temporalIntersects(IndexEnvelope other): Ελέγχει αν το συγκεκριμένο αντικείμενο IndexEnvelope «τέμνεται» με ένα άλλο αντικείμενο IndexEnvelope. Με τον όρο intersect ορίζεται ότι τα δύο αντικείμενα έχουν τουλάχιστον ένα κοινό σημείο. Η μέθοδος intersects είναι η αντίθετη της μεθόδου disjoint.
-void temporalExpandToInclude(IndexEnvelope other): Το αντικείμενο επεκτείνεται προκειμένου να περιλαμβάνει το αντικείμενο που δίνεται ως όρισμα.
-void temporalExpandBy(double[] padding, long[] timestamp) : Το αντικείμενο επεκτείνεται προκειμένου να περιλαμβάνει τα στοιχεία που δίνονται ως όρισμα.
-void temporalExpandToInclude(double x, double y, long timestamp) : Το αντικείμενο επεκτείνεται προκειμένου να περιλαμβάνει τα στοιχεία που δίνονται ως όρισμα.
double getTimeDiff(): Επιστρέφει τη χρονική διαφορά μεταξύ δύο αντικειμένων IndexEnvelope.

8.2.2.2 Κλάση IndexEnvelopeDecoder

Η κλάση IndexEnvelopeDecoder είναι ένα interface που περιλαμβάνει την παρακάτω μέθοδο που επιστρέφει αντικείμενα IndexEnvelope προς υλοποίηση:

- IndexEnvelope decodeEnvelope(PropertyContainer container)

Οι κλάσεις που υλοποιούν το συγκεκριμένο interface θα πρέπει να περιέχουν μία μέθοδο με την παραπάνω ονομασία που μετά την ανάγνωση των ιδιοτήτων ενός συγκεκριμένου property container (για παράδειγμα αντικείμενο Node) θα πρέπει να δημιουργούν ένα αντικείμενο IndexEnvelope. Το συγκεκριμένο interface υλοποιείται στην κλάση Hybrid3DRTreeIndex και χρησιμοποιείται για την ανάκτηση των σημασιολογικών πληροφοριών από έναν κόμβο και τη δημιουργία του αντίστοιχου αντικειμένου IndexEnvelope.

8.2.2.3 Κλάση SpatialTemporalIndexReader

Η κλάση SpatialTemporalIndexReader είναι ένα interface που αφορά την ανάγνωση του ευρετηρίου και περιλαμβάνει τις παρακάτω μεθόδους προς υλοποίηση:

- IndexEnvelopeDecoder getEnvelopeDecoder(): Ανάκτηση από τη βάση της κλάσης που υλοποιεί το interface IndexEnvelopeDecoder και αφορά την ανάκτηση δεδομένων για τη δημιουργία ενός αντικειμένου IndexEnvelope.
- boolean isEmpty(): Έλεγχος αν ο root κόμβος του index περιέχει ή όχι την ιδιότητα «bbox», δηλαδή αν είναι γεωμετρικός κόμβος ή όχι.
- int count(): Απαρίθμηση των γεωγραφικών αντικειμένων (κόμβων) στο ευρετήριο.
- IndexEnvelope getBoundingBox(): Επιστροφή του αντικειμένου IndexEnvelope που περιβάλλει το ευρετήριο.
- boolean isNodeIndexed(Long nodeId): Έλεγχος αν ο κόμβος με id nodeId υπάρχει στο ευρετήριο.

- Iterable<Node> getAllIndexedNodes(): Επιστροφή όλων των κόμβων που περιέχουν γεωγραφικά αντικείμενα και υπάρχουν στο ευρετήριο.
- SearchResults searchIndex(TemporalSearchFilter filter): Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter.

8.2.2.4 Κλάση SpatialTemporalIndexWriter

Η κλάση SpatialTemporalIndexWriter είναι ένα interface που αφορά την εγγραφή στο ευρετήριο, αποτελεί επέκταση του interface SpatialTemporalIndexReader και περιλαμβάνει τις παρακάτω μεθόδους προς υλοποίηση:

- void add(Node geomNode): Προσθήκη ενός geometry node στο ευρετήριο.
- void remove(long geomNodeId, boolean deleteGeomNode): Διαγραφή ενός geometry node από το ευρετήριο με id «geomNodeId».
- void removeAll(boolean deleteGeomNodes, Listener monitor): Διαγραφή όλων των κόμβων από το ευρετήριο.
- void clear(Listener monitor): Εκκαθάριση των περιεχόμενων του ευρετηρίου.

Το συγκεκριμένο interface υλοποιείται στην κλάση Hybrid3DRTreeIndex.

8.2.2.5 Κλάση SpatialTemporalIndexVisitor

Η κλάση SpatialTemporalIndexVisitor είναι ένα interface που αφορά την προσπέλαση του ευρετηρίου και περιλαμβάνει τις παρακάτω μεθόδους προς υλοποίηση:

- boolean needsToVisit(IndexEnvelope indexNodeEnvelope)
- void onIndexReference(Node geomNode)

8.2.2.6 Κλάση SpatialTemporalIndexRecordCounter

Η κλάση SpatialTemporalIndexRecordCounter είναι υλοποίηση του interface SpatialTemporalIndexVisitor και περιλαμβάνει τις παρακάτω μεθόδους:

- boolean needsToVisit(IndexEnvelope indexNodeEnvelope)
- void onIndexReference(Node geomNode)
- int getResult()

Η ανωτέρω κλάση χρησιμοποιείται για την απαρίθμηση του συνολικού αριθμού των geometry nodes που υπάρχουν στο ευρετήριο. Το συγκεκριμένο interface

υλοποιείται στην κλάση Hybrid3DRTreeIndex και χρησιμοποιείται για την εύρεση του αριθμού των κόμβων που περιλαμβάνονται στο ευρετήριο.

8.2.2.7 Κλάση Hybrid3DRTreeIndex

Η κλάση Hybrid3DRTreeIndex είναι υλοποίηση του interface SpatialTemporalIndexWriter. Αποτελεί την βασική κλάση του πακέτου που δημιουργεί το Hybrid 3D R-Tree ευρετήριο και δίνει την δυνατότητα αναζήτησης στο index με τους υλοποιημένους αλγόριθμους που θα αναλυθούν σε επόμενη ενότητα.

Η κλάση Hybrid3DRTreeIndex αποτελεί τροποποίηση της αρχικής κλάσης του ευρετηρίου R-Tree του Neo4j με τις παρακάτω προσθήκες προκειμένου να λαμβάνεται υπόψη η σημασιολογική διάσταση της πληροφορίας:

- Προσθήκη του αντικειμένου IndexEnvelopeDecoder στους κατασκευαστές της κλάσης και χρησιμοποίηση του στις μεθόδους της κλάσης προκειμένου να λαμβάνεται υπόψη και η χρονική διάσταση της πληροφορίας.
- Χρησιμοποίηση στην κλάση κατάλληλων μεθόδων που εκτός από τον χωρικό έλεγχο για την επικάλυψη αντικειμένων λαμβάνουν υπόψη και τη χρονική διάσταση των αντικειμένων.
- Δημιουργία των μεθόδων προσθήκης των λεκτικών ιδιοτήτων των γεωμετρικών στους κόμβους του δέντρου.
- Δημιουργία επιπρόσθετων μεθόδων αναζήτησης (pattern queries) και διάσχισης του δέντρου που λαμβάνουν υπόψη χωροχρονικά και λεκτικά κριτήρια.
- Αντικατάσταση της χρησιμοποίησης των αντικειμένων Envelope (αρχικό αντικείμενο του πακέτου) στις μεθόδους της κλάσης με αντικείμενα IndexEnvelope και τροποποίηση των αντίστοιχων bounding box έτσι ώστε να αποθηκεύονται σε αυτά και η σημασιολογική πληροφορία.
- Υλοποίηση του interface SpatialTemporalIndexWriter.
- Αντικατάσταση των μεθόδων του αντικειμένου Envelope για την χωρική σύγκριση μεταξύ δύο αντικειμένων Envelope, με τις αντίστοιχες μεθόδους του αντικειμένου IndexEnvelope προκειμένου να λαμβάνεται υπόψη και η χρονική πληροφορία κατά την τοποθέτηση των αντικειμένων στο R-Δέντρο.
- Τροποποίηση της μεθόδου getAllIndexInternalNodes() που επιστρέφει σε Iterable<Node> όλα τα αντικείμενα Node που περιέχουν geometry objects έτσι ώστε να μην χρησιμοποιεί deprecated μεθόδους και κλάσεις (μέθοδο traverse, αντικείμενα StopEvaluator, Order και ReturnableEvaluator) για την εύρεση των κόμβων, αλλά να χρησιμοποιεί κλάσεις και αντικείμενα που χρησιμοποιούνται στις τελευταίες εκδόσεις του Neo4j για την διάσχιση της δομής του γράφου (αντικείμενο Traversal και αντίστοιχες μεθόδους).

- Τροποποίηση της private κλάσης GeometryNodeIterator με σκοπό την διάσχιση των γεωμετρικών κόμβων της βάσης έτσι ώστε να υπάρχει κατασκευαστής που αρχικοποιεί τον Iterator<Node> με τους indexed κόμβους του γράφου.

Περιλαμβάνει τις παρακάτω μεθόδους:

- void addLexicalRTreeEpisodePropertiesChild(): Δημιουργία λεκτικού ευρετηρίου για τον αλγόριθμο EB Hybrid R-Tree Search.
- void createNodeLexicalIndex(Node geomNode): Δημιουργία λεκτικού ευρετηρίου για τον αλγόριθμο EB Hybrid R-Tree Search.
- void createEpisodesNextRelationships(Node geomNode): Δημιουργία των «NEXT» relationships για τον αλγόριθμο EB Hybrid R-Tree Search.
- Node getRightRootNode(): Επιστρέφει τον κόμβο της ρίζας του δέντρου.
- boolean rootNodeHasRelationshipChild(): Έλεγχος αν η ρίζα έχει relationship με type «RTREE_CHILD».
- void addLexicalRTreeTrajectoryPropertiesChild(): Προσθήκη λεκτικών ιδιοτήτων στους κόμβους του δέντρου για τον αλγόριθμο STB Hybrid R-Tree Search.
- void updateMBBTag(Node upNode, Node downNode, String tag): Ενημέρωση του MBB του κόμβου upNode λαμβάνοντας υπόψη το MBB του κόμβου downNode για τα φύλλα του δέντρου. Χρησιμοποιείται στον αλγόριθμο ESTB Hybrid R-Tree Search.
- void updateMBBTagChild(Node rootNode, Node downNode, String tag): Ενημέρωση του MBB του κόμβου upNode λαμβάνοντας υπόψη το MBB του κόμβου downNode για κόμβους που δεν είναι φύλλα του δέντρου. Χρησιμοποιείται στον αλγόριθμο ESTB Hybrid R-Tree Search.
- void addLexicalRTreeTrajectoryPropertiesChildA3(): Προσθήκη λεκτικών ιδιοτήτων στους κόμβους του δέντρου για τον αλγόριθμο ESTB Hybrid R-Tree Search.
- String getChildIndexTrajectory(Node geomNode): Επιστρέφει το όνομα του λεκτικού ευρετηρίου για τον κόμβο geomNode για τον αλγόριθμο STB Hybrid R-Tree Search.
- String getChildIndex(Node geomNode): Επιστρέφει το όνομα του λεκτικού ευρετηρίου για τον κόμβο geomNode για τον αλγόριθμο EB Hybrid R-Tree Search.
- List<SpatialTemporalDatabaseRecord> searchTreeEpisodeChild(LayerST layer,IndexEnvelope envelope, String tag): Αναζήτηση episode με συγκεκριμένα κριτήρια για τον αλγόριθμο EB Hybrid R-Tree Search.
- IndexEnvelope sumIndexEnvelopes(List<IndexEnvelope> envelopes): Επιστρέφει αντικείμενο IndexEnvelope που το MBB του είναι το άθροισμα των MBB των αντικειμένων IndexEnvelope που δίνονται ως όρισμα.
- boolean checkEmptyEnvelope(List<IndexEnvelope> envelopes): Ελέγχει εάν υπάρχει στην συγκεκριμένη λίστα αντικείμενο IndexEnvelope που δεν έχει δεδομένα.
- int getTrueSizeSpatialTemporalCheck(List<IndexEnvelope> envelopes): Επιστρέφει τον αριθμό των αντικείμενων IndexEnvelope που έχουν δεδομένα.

- SemanticTrajectory getTrajectoryFromEpisode (Node node): Επιστρέφει το SemanticTrajectory που ανήκει ένας συγκεκριμένος κόμβος στο EB Hybrid R-Tree.
- int getTagsRealSize (List<String> tags): Επιστρέφει τον αριθμό των λεκτικών ορισμάτων που δεν έχουν την τιμή «*».
- boolean checkTrajectorySTL(Node node, List<IndexEnvelope> envelopes, List<String> tags): Έλεγχος του αλγόριθμου STB Hybrid R-Tree Search και ESTB Hybrid R-Tree Search για την ικανοποίηση των χωροχρονικών και λεκτικών απαιτήσεων που έχουν δοθεί ως όρισμα για τον κόμβο node.
- boolean checkTrajectoryTags_A3(Node node, List<IndexEnvelope> envelopes, List<String> tags): Έλεγχος του αλγόριθμου ESTB Hybrid R-Tree Search για την ικανοποίηση των λεκτικών απαιτήσεων που έχουν δοθεί ως όρισμα για τον κόμβο node.
- List<SpatialTemporalDatabaseRecord> searchTreeTrajectoryChild(LayerST layer, List<IndexEnvelope> envelopes, List<String> tags):): Αναζήτηση trajectory με συγκεκριμένα κριτήρια για τον αλγόριθμο STB Hybrid R-Tree Search.
- List<SpatialTemporalDatabaseRecord> searchTreeTrajectoryChildA3(LayerST layer, List<IndexEnvelope> envelopes, List<String> tags): Αναζήτηση trajectory με συγκεκριμένα κριτήρια για τον αλγόριθμο ESTB Hybrid R-Tree Search.
- SpatialTemporalDatabaseRecord searchTreeObjectTrajectoryEpisodeChild(LayerST layer,IndexEnvelope envelope, String tag, String objectID, String trajectoryID, String previousID): : Αναζήτηση episode με συγκεκριμένα κριτήρια για τον αλγόριθμο EB Hybrid R-Tree Search.
- Node getFirstEpisode(Node node): Επιστρέφει το πρώτο episode για το trajectory που ανήκει το συγκεκριμένο episode που υπάρχει στον geometry κόμβο node.
- int getKleeneStar(List<String> tags, List<IndexEnvelope> envelopes): Επιστρέφει τον αριθμό των περιπτώσεων που ως λεκτικό κριτήριο είναι το «*» και το αντικείμενο IndexEnvelope είναι κενό.
- List<SpatialTemporalDatabaseRecord> searchTreeTrajectoryEpisodeNext(LayerST layer, List<IndexEnvelope> envelopes, List<String> tags): Επιστρέφει τα αποτελέσματα του αλγόριθμου EB Hybrid R-Tree Search.
- IndexEnvelopeDecoder getEnvelopeDecoder(): Ανάκτηση από τη βάση της κλάσης που υλοποιεί το interface IndexEnvelopeDecoder και αφορά την ανάκτηση δεδομένων για τη δημιουργία ενός αντικειμένου IndexEnvelope.
- void add(Node geomNode): Προσθήκη ενός geometry node στο ευρετήριο.
- void remove(long geomNodeId, boolean deleteGeomNode) : Διαγραφή ενός node από το ευρετήριο.
- void remove(long geomNodeId, boolean deleteGeomNode, boolean throwExceptionIfNotFound) : Διαγραφή ενός geometry node από το ευρετήριο.
- Node deleteEmptyTreeNodes(Node indexNode, RelationshipType relType): Διαγραφή κόμβων που δεν δείχνουν σε geometry nodes μετά από αναδιοργάνωση της δομής του δέντρου.

- void removeAll(final boolean deleteGeomNodes, final Listener monitor) : Διαγραφή όλων των κόμβων από το ευρετήριο.
- IndexEnvelope getBoundingBox(): Επιστροφή του αντικειμένου IndexEnvelope που περιβάλλει το ευρετήριο.
- int count(): Απαρίθμηση των γεωγραφικών αντικειμένων στο ευρετήριο.
- boolean isEmpty(): Έλεγχος αν ο root κόμβος περιέχει ή όχι την ιδιότητα «bbox», δηλαδή αν είναι γεωμετρικός κόμβος ή όχι. Πραγματοποιείται δηλαδή έλεγχος αν έχει δημιουργηθεί το ευρετήριο.
- boolean isNodeIndexed(Long geomNodeId) : Έλεγχος αν ο κόμβος με id geomNodeId υπάρχει στο ευρετήριο.
- void warmUp(): Αναδρομική διάσχιση του δέντρου του ευρετηρίου.
- Iterable<Node> getAllIndexInternalNodes()/getAllIndexedNodes(): Επιστροφή όλων των κόμβων που περιέχουν γεωγραφικά αντικείμενα και υπάρχουν στο ευρετήριο σε Iterable <Node>.
- checkPosition(TraversalPosition position): Έλεγχος της σχέσης (RTREE_CHILD/RTREE_REFERENCE) σε μία συγκεκριμένη διάσχιση του δέντρου μέσω των παρακάτω δύο μεθόδων.
- boolean isReturnableNode(TraversalPosition position): Έλεγχος αν η θέση έχει περαιτέρω relationships για διάσχιση.
- boolean isStopNode(TraversalPosition position): Έλεγχος αν η θέση δεν έχει περαιτέρω relationships για διάσχιση.
- SearchResults searchIndex(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter.
- void visit(SpatialTemporalIndexVisitor visitor, Node indexNode): Διάσχιση του δέντρου με σημείο εκκίνησης τον κόμβο indexNode.
- Node getIndexRoot(): Επιστροφή του root Node (ρίζα του δέντρου).
- IndexEnvelope getChildNodeEnvelope(Node child, RelationshipType relType): Έλεγχος αν ο κόμβος είναι φύλλο ή όχι και χρησιμοποίηση των δύο παρακάτω μεθόδων αντίστοιχα για επιστροφή του αντικειμένου IndexEnvelope του γεωμετρικού κόμβου child.
- IndexEnvelope getLeafNodeEnvelope(Node geomNode): Επιστροφή του IndexEnvelope του συγκεκριμένου geometry node με χρήση της κλάσης IndexEnvelopeDecoder.
- IndexEnvelope getIndexNodeEnvelope(Node indexNode): Επιστροφή του αντικειμένου IndexEnvelope που σχηματίζεται από την ιδιότητα «bbox» του indexNode.
- void visitInTx(SpatialTemporalIndexVisitor visitor, Long indexNodeId): Διάσχιση του δέντρου με σημείο εκκίνησης τον node με id indexNodeId.
- void initIndexMetadata(): Δημιουργία κόμβου για καταγραφή των metadata δεδομένων.

- void initIndexRoot(): Δημιουργία root κόμβου για κατασκευή ευρετηρίου.
- Node getMetadataNode(): Επιστροφή του κόμβου που διατηρούνται τα metadata.
- void saveCount(): Απαρίθμηση των γεωγραφικών αντικειμένων στο ευρετήριο.
- boolean nodeIsLeaf(Node node): Έλεγχος αν ο κόμβος είναι φύλλο στο δέντρο.
- Node chooseSubTree(Node parentIndexNode, Node geomRootNode): Επιστροφή συγκεκριμένου υποδέντρου που θα τοποθετηθεί ο geometry node geomRootNode.
- double getAreaEnlargement(Node indexNode, Node geomRootNode): Επιστροφή της ποσοτικής αύξησης που θα υπάρξει στο εμβαδό αν προστεθεί ο geometry node geomRootNode στο υποδέντρο του κόμβου indexNode.
- Node chooseIndexNodeWithSmallestArea(List<Node> indexNodes): Επιστροφή του κόμβου με το μικρότερο εμβαδό στο αντικείμενο IndexEnvelope.
- int countChildren(Node indexNode, RelationshipType relationshipType): Επιστροφή του αριθμού των παιδιών ενός κόμβου για ένα συγκεκριμένο relationship.
- boolean insertInLeaf(Node indexNode, Node geomRootNode): εισαγωγή κόμβου στο δέντρο σε συγκεκριμένο σημείο (φύλλο).
- void splitAndAdjustPathBoundingBox(Node indexNode): Δημιουργία καινούριου κόμβου και επαναδιανομή των παιδιών του κόμβου indexNode μεταξύ των δύο κόμβων.
- Node quadraticSplit(Node indexNode): Δημιουργία 2 καινούριων υποδέντρων με διαχωρισμό των κόμβων του υποδέντρου του κόμβου που δίνεται ως όρισμα.
- Node quadraticSplit(Node indexNode, RelationshipType relationshipType): Δημιουργία και διαχωρισμός καινούριων υποδέντρων και διαχωρισμός των παιδιών που έχουν με χρησιμοποίηση της προηγούμενης μεθόδου και επιπλέον κριτήριο αν ο κόμβος είναι φύλλο ή όχι (RTREE_CHILD/RTREE_REFERENCE).
- void createNewRoot(Node oldRoot, Node newIndexNode): Δημιουργία καινούριας ρίζας που έχει Relationship RTREE_CHILD προς τους δύο κόμβους του ορίσματος.
- boolean addChild(Node parent, RelationshipType type, Node newChild): Προσθήκη συγκεκριμένου κόμβου και σχέσης σε έναν αρχικό κόμβο.
- void adjustPathBoundingBox(Node indexNode): Ενημέρωση του bounding box του πατέρα του συγκεκριμένου κόμβου με relationship «RTREE_CHILD».
- boolean adjustParentBoundingBox(Node indexNode, RelationshipType relationshipType) : Αναπροσαρμογή του bounding box του συγκεκριμένου κόμβου για τη συγκεκριμένη σχέση που δίνεται ως όρισμα.
- boolean expandParentBoundingBoxAfterNewChild(Node parent, double[] childBBox): Αναπροσαρμογή του bounding box του συγκεκριμένου κόμβου με τις τιμές που δίνονται ως όρισμα.

- boolean setMin(double[] parent, double[] child, int index): Έλεγχος αν ο πίνακας parent στη θέση index έχει μεγαλύτερη τιμή από τον πίνακα child.
- boolean setMax(double[] parent, double[] child, int index): Έλεγχος αν ο πίνακας parent στη θέση index έχει μικρότερη τιμή από τον πίνακα child.
- Node getIndexNodeParent(Node indexNode): Έλεγχος αν ο κόμβος είναι η κατάληξη μίας relationship «RTREE_CHILD» και επιστροφή του κόμβου που αρχίζει η σχέση. Αν δεν βρεθεί η συγκεκριμένη relationship επιστρέφει τιμή «NULL».
- void deleteRecursivelySubtree(Node indexNode): Διαγραφή του υποδέντρου ενός συγκεκριμένου κόμβου.
- Node findLeafContainingGeometryNode(Node geomNode, boolean throwExceptionIfNotFound): Εύρεση συγκεκριμένου κόμβου με χωρικό κριτήριο έναν geometry node.
- void deleteNode(Node node): Διαγραφή ενός συγκεκριμένου κόμβου από το ευρετήριο.
- static IndexEnvelope createEnvelope(IndexEnvelope e, IndexEnvelope e1): «Επέκταση» του IndexEnvelope e με «προσθήκη» σε αυτόν του IndexEnvelope e1.
- GraphDatabaseService getDatabase(): Επιστροφή του GraphDatabaseService που χρησιμοποιείται στο ευρετήριο.
- void removeAll(final boolean deleteGeomNodes, final Listener monitor): Διαγραφή των περιεχομένων του ευρετηρίου.

8.2.2.8 Κλάση TemporalSearchFilter

Η κλάση TemporalSearchFilter είναι ένα interface που περιλαμβάνει τις παρακάτω μεθόδους προς υλοποίηση:

- boolean needsToVisit(IndexEnvelope envelope)
- boolean geometryMatches (Node geomNode)

Οι κλάσεις που υλοποιούν το συγκεκριμένο interface θα πρέπει να περιέχουν υλοποίηση μεθόδου που να προσδιορίζει τα κριτήρια με τα οποία θα πραγματοποιείται η αναζήτηση στο δέντρο. Το συγκεκριμένο interface υλοποιείται στην κλάση Hybrid3DRTreeIndex.

8.2.3 Υλοποίηση των αλγόριθμων σε Java

Στη συνέχεια ακολουθεί η υλοποίηση των αλγόριθμων σε Java καθώς και αναλυτική περιγραφή των υλοποιήσεων.

8.2.3.1 Αλγόριθμος EB

```
1 .  List<SpatialTemporalDatabaseRecord> searchTreeTrajectoryEpisodeNext(LayerST layer, List<IndexEnvelope>
envelopes, List<String> tags)
2 . {
3 .         List<SpatialTemporalDatabaseRecord> records= new ArrayList<SpatialTemporalDatabaseRecord>();
4 .         List<SpatialTemporalDatabaseRecord> firstNodes= new ArrayList<SpatialTemporalDatabaseRecord>();
5 .
6 .         firstNodes.addAll(searchTreeEpisodeChild(layer,envelopes.get(0),tags.get(0)));
7 .
8 .         if (envelopes.size()==1) {
9 .                 for(int i = 0; i<firstNodes.size(); i++)
10.                 {
11.                         records.add(new
SpatialTemporalDatabaseRecord(layer,getFirstEpisode(firstNodes.get(i).getGeomNode())));
12.                 }
13.                 return records;
14.         }
15.         for(int i = 0; i<firstNodes.size(); i++)
16.         {
17.                 if (checkEpisodeSTL(firstNodes.get(i).getGeomNode(), envelopes, tags))
18.                 {
19.                         SpatialTemporalDatabaseRecord stdr=new
SpatialTemporalDatabaseRecord(layer,getFirstEpisode(firstNodes.get(i).getGeomNode()));
20.                         if (!records.contains(stdr)) records.add(stdr);
21.                 }
22.         }
23.
24.         return records;
25. }
```

Στον αλγόριθμο δίνονται ως ορίσματα σε μορφή λιστών τα δεδομένα του regular expression. Μία λίστα αφορά τα χωροχρονικά κριτήρια και μία λίστα τα λεκτικά κριτήρια (γραμμή 1). Στη συνέχεια, γίνεται διάσχιση του δέντρου προκειμένου να γίνει εύρεση αν υπάρχει episode ή episodes που ικανοποιούν τις χωροχρονικές και λεκτικές απαιτήσεις του πρώτου pattern element με κατάλληλη μέθοδο αναζήτησης (γραμμή 6).

Αρχικά η μέθοδος αναζήτησης πραγματοποιεί έλεγχο στη ρίζα του δέντρου αν ικανοποιούνται τα λεκτικά και τα χωροχρονικά κριτήρια. Αν επαληθευτούν τα

κριτήρια του ερωτήματος στη ρίζα, τότε ελέγχεται αν το δέντρο έχει μόνο ένα επίπεδο δηλαδή μόνο φύλλα ή έχει περισσότερα επίπεδα. Αν είναι μονοεπίπεδο τότε στη συνέχεια γίνεται έλεγχος των φύλλων του δέντρου αν ικανοποιούν τις χωροχρονικές και λεκτικές απαιτήσεις του ερωτήματος.

Αν το δέντρο έχει περισσότερα επίπεδα τότε χρησιμοποιώντας το Breadth First Search αλγόριθμο αναζήτησης γίνεται διάσχιση του δέντρου ανά επίπεδο ελέγχοντας σε κάθε επίπεδο αν υπάρχουν κόμβοι που ικανοποιούν τις απαιτήσεις του ερωτήματος.

Ειδικότερα, αρχικά γίνεται διάσχιση του πρώτου επιπέδου του δέντρου μετά τη ρίζα και ελέγχεται αν στο συγκεκριμένο επίπεδο υπάρχουν κόμβοι που ικανοποιούν τις απαιτήσεις του ερωτήματος. Για τους κόμβους που ικανοποιούν τα κριτήρια του ερωτήματος εφαρμόζεται επαναληπτικά η ίδια αναζήτηση προκειμένου να βρεθούν αν υπάρχουν παιδιά που τα ικανοποιούν.

Μετά την διάσχιση όλων των επιπέδων του δέντρου πραγματοποιείται στη συνέχεια έλεγχος στα φύλλα του δέντρου αν υπάρχουν episodes που ικανοποιούν τις απαιτήσεις.

Στη συνέχεια για τα episodes που έχουν βρεθεί γίνεται έλεγχος μέσω κατάλληλης μεθόδου (γραμμή 17) για τα υπόλοιπα pattern elements του pattern query αν ικανοποιούνται από τα υπόλοιπα episodes των semantic trajectories που τα episodes ανήκουν.

Μέσω της relationship «NEXT» που συνδέονται τα episodes που ανήκουν στο ίδιο semantic trajectory, γίνεται σταδιακά έλεγχος αν ικανοποιούνται οι χωροχρονικές και λεκτικές απαιτήσεις. Τα semantic trajectories που έχουν βρεθεί ότι ικανοποιούν τις απαιτήσεις του pattern query επιστρέφονται στη συνέχεια ως αποτελέσματα αναζήτησης (γραμμή 24).

8.2.3.2 Αλγόριθμος STB

```
1 . List<SpatialTemporalDatabaseRecord> searchTreeTrajectoryChild(LayerST layer, List<IndexEnvelope> envelopes,
List<String> tags)
2 . {
3 .         List<SpatialTemporalDatabaseRecord> records= new ArrayList<SpatialTemporalDatabaseRecord>();
4 .         List<ArrayList<Node>> childNodes= new ArrayList<ArrayList<Node>>();
5 .         IndexManager indexManager = getDatabase().index();
6 .         String trajectoryTags=tags.toString().replace("[", "").replace("]", ""). replace(" ,", "#").replace(" ", "");
7 .         trajectoryTags=("*"+trajectoryTags+"*").replace("**", "*");
8 .         IndexEnvelope trajectoryEnvelope=null;
9 .         boolean emptyEnvelope=checkEmptyEnvelope(envelopes);
10 .        if (!emptyEnvelope) {
```

```
11 .        trajectoryEnvelope=sumIndexEnvelopes(envelopes);
12 .        }
13 .        boolean rootcheck;
14 .        if (emptyEnvelope) rootcheck=indexManager.forNodes("root").query("tags",trajectoryTags).size()==1;
15 .        else rootcheck=(getIndexNodeTemporalEnvelope(getRightRootNode()).temporalCovers(trajectoryEnvelope) &&
indexManager.forNodes("root").query("tags",trajectoryTags).size()==1);
16 .        if (rootcheck)
17 .        {
18 .                    if (rootNodeHasRelationshipChild())
19 .                    {
20 .                    childNodes.add(new ArrayList<Node>());
21 .                    for (Node node: new Iterable<Node>() {
22 .                       public Iterator<Node> iterator() {
23 .                          return (( (Traversal.description()
24 .                          .breadthFirst()
25 .                          .relationships( RTreeRelationshipTypes.RTREE_CHILD, Direction.OUTGOING )
26 .                          .evaluator( Evaluators.atDepth(1))
27 .                          .evaluator( Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_CHILD)
)
28 .                          .evaluator( Evaluators.excludeStartPosition() )) .traverse(getRightRootNode())).nodes().iterator() );
29 .                       }}  )
30 .                    {
31 .                                if (emptyEnvelope) {
32 .                                        if
(indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1)
33 .                                                childNodes.get(0).add(node);
34 .                                }
35 .                                else {
36 .                                        if (getIndexNodeTemporalEnvelope(node).temporalCovers(trajectoryEnvelope) &&
indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1)
37 .                                                childNodes.get(0).add(node);
38 .                                }
39 .                    }
40 .                    childEpisodeDepth=getChildDepth(getRightRootNode());
41.                     int k=0;
42 .                    while (k<=childEpisodeDepth-1)
43 .                    {
44 .                       childNodes.add(new ArrayList<Node>());
45 .                               for (Node node2: childNodes.get(k)) {
46 .                                       treeNode=node2;
47 .                               for (Node node: new Iterable<Node>() {
48 .                                  public Iterator<Node> iterator() {
49 .                                     return (( (Traversal.description()
50 .                                     .breadthFirst()
51 .                                     .relationships( RTreeRelationshipTypes.RTREE_CHILD, Direction.OUTGOING )
52 .                                     .evaluator( Evaluators.atDepth(1))
53 .                                     .evaluator(
Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_CHILD) )
54 .                                     .evaluator( Evaluators.excludeStartPosition() )) .traverse(treeNode)).nodes().iterator() );
55 .                                  }}  )
56 .                               {
57 .                                  if (emptyEnvelope) {
58 .                                        if
(indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1)

59 .                                                childNodes.get(k+1).add(node);
60 .                                }
61 .                                else {
```

```
62 .                                    if (getIndexNodeTemporalEnvelope(node).temporalCovers(trajectoryEnvelope) &&
indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1)
63 .                                        childNodes.get(k+1).add(node);
64 .                                }
65 .                            }
66 .                        }
67 .                            k++;
68 .                    }
69 .                    if (childNodes.get(childNodes.size()-1).size()==0)  childNodes.remove(childNodes.size()-1);
70 .                    for (Node node2: childNodes.get(k-1)) {
71 .                            treeNode=node2;
72 .
73 .                            for (Node node3: new Iterable<Node>() {
74 .                               public Iterator<Node> iterator() {
75 .                                  return (( (Traversal.description()
76 .                                  .breadthFirst()
77 .                                  .relationships( RTreeRelationshipTypes.RTREE_REFERENCE, Direction.OUTGOING )
78 .                                  .evaluator(
Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_REFERENCE) )
79 .                                  .evaluator( Evaluators.excludeStartPosition() )) .traverse(treeNode)).nodes().iterator() );
80 .                               }} )
81 .                            {
82 .                                    if ((Arrays.toString((String[])node3.getProperty("tagsArray")).replace("[",
"").replace("]", ""). replace(" ,", "#").replace(" ", "")  ).matches(trajectoryTags.replace("*", ".*"))
83 .                                                    &&checkTrajectorySTL(node3, envelopes,tags))
84 .                                        records.add(new SpatialTemporalDatabaseRecord(layer,node3));
85 .                                }
86 .                            }
87 .          }
88 .         else
89 .         {
90 .                    for (Node node4: new Iterable<Node>() {
91 .                       public Iterator<Node> iterator() {
92 .                          return (( (Traversal.description()
93.                           .breadthFirst()
94.                           .relationships( RTreeRelationshipTypes.RTREE_REFERENCE, Direction.OUTGOING )
95.                           .evaluator(
Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_REFERENCE) )
96.                           .evaluator( Evaluators.excludeStartPosition() )) .traverse(getRightRootNode())).nodes().iterator() );
97.                       }} )
98.                    {
99.                           if (Arrays.toString((String[])node4.getProperty("tagsArray")).replace("[", "").replace("]", "").
replace(" ,", "#").replace(" ", "").matches(trajectoryTags.replace("*", ".*"))
100.                                   &&checkTrajectorySTL(node4, envelopes,tags))
101.                                   records.add(new SpatialTemporalDatabaseRecord(layer,node4));
102.                           }
103.        }
104.
105.        }
106.        return records;
107. }
```

Στον αλγόριθμο δίνονται ως ορίσματα σε μορφή λιστών τα δεδομένα του regular expression. Μία λίστα αφορά τα χωροχρονικά κριτήρια και μία λίστα τα λεκτικά κριτήρια (γραμμή 1). Στη συνέχεια, δημιουργείται σε μορφή String το συνολικό

λεκτικό pattern του query προκειμένου να χρησιμοποιηθεί κατά την διάσχιση του δέντρου (γραμμή 6).

Σε κατάλληλη boolean μεταβλητή γίνεται έλεγχος αν μπορεί να σχηματιστεί συνολικό MBB για το query ή όχι (γραμμή 9). Αν μπορεί να σχηματιστεί τότε δημιουργείται το κατάλληλο MBB του ερωτήματος (γραμμή 11).

Στη συνέχεια πραγματοποιείται έλεγχος στη ρίζα του δέντρου αν ικανοποιούνται τα χωροχρονικά κριτήρια αν έχει σχηματιστεί συνολικό MBB ή μόνο τα λεκτικά αν δεν έχει σχηματιστεί (γραμμές 14 και 15).

Αν επαληθευτούν τα κριτήρια του ερωτήματος, τότε ελέγχεται αν το δέντρο έχει μόνο ένα επίπεδο δηλαδή μόνο φύλλα ή έχει περισσότερα επίπεδα (γραμμή 18). Αν είναι μονοεπίπεδο τότε στη συνέχεια γίνεται έλεγχος των φύλλων του δέντρου αν ικανοποιούν τις χωροχρονικές και λεκτικές απαιτήσεις του ερωτήματος (γραμμές 90-105).

Αν το δέντρο έχει περισσότερα επίπεδα τότε χρησιμοποιώντας το Breadth First Search αλγόριθμο αναζήτησης γίνεται διάσχιση του δέντρου ανά επίπεδο ελέγχοντας σε κάθε επίπεδο αν υπάρχουν κόμβοι που ικανοποιούν τις απαιτήσεις του ερωτήματος.

Ειδικότερα, αρχικά γίνεται διάσχιση του πρώτου επιπέδου του δέντρου μετά τη ρίζα και ελέγχεται αν στο συγκεκριμένο επίπεδο υπάρχουν κόμβοι που ικανοποιούν τις απαιτήσεις του ερωτήματος (γραμμές 21-39). Για τους κόμβους που ικανοποιούν τα κριτήρια του ερωτήματος εφαρμόζεται επαναληπτικά η ίδια αναζήτηση προκειμένου να βρεθούν αν υπάρχουν παιδιά που τα ικανοποιούν (γραμμές 42-68).

Μετά την διάσχιση όλων των επιπέδων του δέντρου πραγματοποιείται στη συνέχεια έλεγχος στα φύλλα του δέντρου αν υπάρχουν πλέον semantic trajectories που ικανοποιούν τις απαιτήσεις (γραμμές 73-86).

8.2.3.3 Αλγόριθμος ESTB

```
1 .  List<SpatialTemporalDatabaseRecord> searchTreeTrajectoryChildA3(LayerST layer, List<IndexEnvelope> envelopes,
List<String> tags) {
2 .      List<SpatialTemporalDatabaseRecord> records= new ArrayList<SpatialTemporalDatabaseRecord>();
3 .      List<ArrayList<Node>> childNodes= new ArrayList<ArrayList<Node>>();
4 .      IndexManager indexManager = getDatabase().index();
5 .      String trajectoryTags=tags.toString().replace("[", "").replace("]", ""). replace(" ,", "#").replace(" ", "");
6 .      trajectoryTags=("*"+trajectoryTags+"*").replace("**", "*");
7 .      IndexEnvelope trajectoryEnvelope=null;
8 .      boolean emptyEnvelope=checkEmptyEnvelope(envelopes);
9 .      if (!emptyEnvelope) {
```

```
10 .        trajectoryEnvelope=sumIndexEnvelopes(envelopes);
11 .        }
12 .        boolean rootcheck;
13 .        if (emptyEnvelope) rootcheck=indexManager.forNodes("root").query("tags",trajectoryTags).size()==1 &&
checkTrajectoryTags_A3(getRightRootNode(), envelopes, tags);
14 .        else rootcheck=(getIndexNodeTemporalEnvelope(getRightRootNode()).temporalCovers(trajectoryEnvelope)
15 .                        && indexManager.forNodes("root").query("tags",trajectoryTags).size()==1
16 .                        && checkTrajectoryTags_A3(getRightRootNode(), envelopes, tags));
17 .        if (rootcheck)
18 .        {
19 .                if (rootNodeHasRelationshipChild())
20 .                {
21 .                childNodes.add(new ArrayList<Node>());
22 .                for (Node node: new Iterable<Node>() {
23 .                   public Iterator<Node> iterator() {
24 .                      return (( (Traversal.description()
25 .                      .breadthFirst()
26 .                      .relationships( RTreeRelationshipTypes.RTREE_CHILD, Direction.OUTGOING )
27 .                      .evaluator( Evaluators.atDepth(1))
28 .                      .evaluator( Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_CHILD)
)
29 .                      .evaluator( Evaluators.excludeStartPosition() )) .traverse(getRightRootNode())).nodes().iterator() );
30 .                   }}  )
31 .                {
32 .                        if (emptyEnvelope) {
33 .                                if
(indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1
34 .                                                && checkTrajectoryTags_A3(node, envelopes, tags))
35 .                                        childNodes.get(0).add(node);
36 .                        }
37 .                        else {
38 .                                if (getIndexNodeTemporalEnvelope(node).temporalCovers(trajectoryEnvelope) &&
indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1
39 .                                                && checkTrajectoryTags_A3(node, envelopes, tags))
40 .                                        childNodes.get(0).add(node);
41 .                        }
42 .                }
43 .                childEpisodeDepth=getChildDepth(getRightRootNode());
44 .                int k=0;
45 .                while (k<=childEpisodeDepth-1)
46 .                {
47 .                        childNodes.add(new ArrayList<Node>());
48 .                        for (Node node2: childNodes.get(k)) {
49 .                                treeNode=node2;
50 .                        for (Node node: new Iterable<Node>() {
51 .                           public Iterator<Node> iterator() {
52 .                              return (( (Traversal.description()
53 .                              .breadthFirst()
54 .                              .relationships( RTreeRelationshipTypes.RTREE_CHILD, Direction.OUTGOING )
55 .                              .evaluator( Evaluators.atDepth(1))
56 .                              .evaluator(
Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_CHILD) )
57 .                              .evaluator( Evaluators.excludeStartPosition() )) .traverse(treeNode)).nodes().iterator() );
58 .                           }}  )
59 .                        {
60 .                           if (emptyEnvelope) {
61 .                                   if
((indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1)&&
checkTrajectoryTags_A3(node2, envelopes, tags))
```

```
62 .                                                    childNodes.get(k+1).add(node);
63 .                                 }
64 .                                 else {
65 .                                         if (getIndexNodeTemporalEnvelope(node).temporalCovers(trajectoryEnvelope) &&
indexManager.forNodes(getChildIndexTrajectory(node)).query("tags",trajectoryTags).size()==1 &&
checkTrajectoryTags_A3(node2, envelopes, tags))
66 .                                                    childNodes.get(k+1).add(node);
67 .                                 }

68 .                                 }
69 .                  }
70 .                                 k++;
71 .                  }
72 .                  if (childNodes.get(childNodes.size()-1).size()==0)  childNodes.remove(childNodes.size()-1);
73 .                  for (Node node2: childNodes.get(k-1)) {
74 .                                 treeNode=node2;
75 .                                 for (Node node3: new Iterable<Node>() {
76 .                                    public Iterator<Node> iterator() {
77 .                                       return (( (Traversal.description()
78 .                                       .breadthFirst()
79 .                                       .relationships( RTreeRelationshipTypes.RTREE_REFERENCE, Direction.OUTGOING )
80 .                                       .evaluator(
Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_REFERENCE) )
81 .                                       .evaluator( Evaluators.excludeStartPosition() )) .traverse(treeNode)).nodes().iterator() );
82 .                                    }} )
83 .                                 {
84 .                                         if (Arrays.toString((String[])node3.getProperty("tagsArray")).replace("[",
"").replace("]", ""). replace(" ,", "#").replace(" ", "")  ).matches(trajectoryTags.replace("*", ".*"))
85 .                                                                 &&checkTrajectorySTL(node3, envelopes,tags)&&
checkTrajectoryTags_A3(node3, envelopes, tags))
86 .                                                    records.add(new SpatialTemporalDatabaseRecord(layer,node3));
87 .                                 }
88 .                  }
89 .          }
90 .          else
91 .          {
92 .                  for (Node node4: new Iterable<Node>() {
93.                     public Iterator<Node> iterator() {
94.                        return (( (Traversal.description()
95.                        .breadthFirst()
96.                        .relationships( RTreeRelationshipTypes.RTREE_REFERENCE, Direction.OUTGOING )
97.                        .evaluator(
Evaluators.includeWhereLastRelationshipTypeIs(RTreeRelationshipTypes.RTREE_REFERENCE) )
98.                        .evaluator( Evaluators.excludeStartPosition() )) .traverse(getRightRootNode())).nodes().iterator() );
99.                     }} )
100.                 {
101.                                                   if (Arrays.toString((String[])node4.getProperty("tagsArray")).replace("[",
"").replace("]", "").replace(" ", ""). replace(" ,", "#").matches(trajectoryTags.replace("*", ".*"))
102.                                                     &&checkTrajectorySTL(node4, envelopes, tags)&&
checkTrajectoryTags_A3(node4, envelopes, tags))
103.                                                   records.add(new SpatialTemporalDatabaseRecord(layer,node4));
                    }
104.                 }
105.         }
106.         }
107.         return records;
108.         }
```

Ο αλγόριθμος έχει την ίδια μορφή με τον προηγούμενο αλγόριθμο (STB) αναζήτησης με τη διαφορά ότι κατά την διάσχιση των επιπέδων του δέντρου και τον έλεγχο ικανοποίησης των κριτηρίων του ερωτήματος ελέγχεται επιπρόσθετα αν υπάρχει το σύνολο των tags (λεκτικά atomic pattern elements) του ερωτήματος στο συγκεκριμένο επίπεδο και αν κάθε tag στο εκάστοτε επίπεδο του δέντρου ικανοποιεί τις χωροχρονικές απαιτήσεις που υπάρχουν στο ερώτημα για αυτό το tag.

Ειδικότερα, ξεκινώντας τη διάσχιση από τη ρίζα του δέντρου εκτός των ελέγχων που γίνονται με τον STB αλγόριθμο γίνεται επιπρόσθετα έλεγχος αν υπάρχει το σύνολο των tags του query στη ρίζα και αν το MBB του κάθε tag που υπάρχει στη ρίζα ικανοποιεί τις χωροχρονικές απαιτήσεις που υπάρχουν στο ερώτημα για αυτό το tag (γραμμή 13-16).

Στη συνέχεια οι ίδιοι έλεγχοι πραγματοποιούνται στο πρώτο επίπεδο του δέντρου (γραμμές 22-89) και στη συνέχεια με επαναληπτική λούπα γίνεται διάσχιση του συνόλου των επιπέδων του δέντρου (γραμμές 45-88). Αν το δέντρο είναι μονοεπίπεδο, μετά τη ρίζα γίνεται έλεγχος των φύλλων του δέντρου (γραμμές 92-103).

8.2.4 Χρονική επέκταση των κλάσεων της Neo4j Spatial Library

Η library Neo4j Spatial για την αναπαράσταση και αποθήκευση των γεωγραφικών αντικειμένων στη Neo4j database περιέχει κατάλληλες κλάσεις αναπαράστασης των χωρικών αντικειμένων. Για την διαχείριση χωροχρονικής και λεκτικής πληροφορίας πραγματοποιήθηκε τροποποίηση των βασικών κλάσεων της library προκειμένου να είναι εφικτή η αναπαράσταση των σημασιολογικών τροχιών και η πραγματοποίηση επερωτήσεων σε αυτές, όπως αναφέρθηκε σε προηγούμενες ενότητες. Στη συνέχεια ακολουθούν οι βασικότερες κλάσεις της Neo4j Spatial Library που επεκτάθηκαν για να λαμβάνεται υπόψη η χρονική και λεκτική διάσταση της πληροφορίας.

8.2.4.1 Κλάση LayerST

Η κλάση LayerST (Layer Spatiotemporal) είναι ένα interface που οι κλάσεις που το υλοποιούν χρησιμοποιούνται για την αποθήκευση και ανάκτηση γεωγραφικών αντικειμένων (JTS αντικείμενα) που βρίσκονται αποθηκευμένα στη Neo4j Database.

Τα γεωγραφικά αντικείμενα αυτά μπορεί να βρίσκονται σε ένα αντικείμενο Dataset, που θα αναλυθεί στη συνέχεια. Περιέχει τις παρακάτω μεθόδους προς υλοποίηση:

- void initialize(SpatialTemporalDatabaseService spatialDatabase, String name, Node layerNode): Μέθοδος αρχικοποίησης του LayerST που καλείται από τον κατασκευαστή της κλάσης. Ο Node layerNode περιέχει πληροφορίες για την αρχικοποίηση του LayerST και η String μεταβλητή name την ονομασία του layer.
- LayerIndexReaderTemporal getIndex(): Επιστροφή του index του layer. Κάθε αντικείμενο layer χρησιμοποιεί ένα συγκεκριμένο implementation των interfaces SpatialTemporalIndexReader και SpatialTemporalIndexWriter.
- SpatialTemporalDatabaseRecord add(Node geomNode): Προσθήκη ενός node με γεωμετρικό αντικείμενο στο LayerST.
- TemporalGeometryFactory getGeometryFactory(): Επιστρέφει το αντικείμενο TemporalGeometryFactory που χρησιμοποιείται για την κατασκευή χωροχρονικών αντικειμένων στο LayerST.
- Node getLayerNode(): Επιστροφή του βασικού Node του LayerST που περιέχει πληροφορίες για το LayerST.
- void delete(Listener monitor): Διαγραφή των στοιχείων του layer μαζί με το ευρετήριο.
- String getName(): Επιστρέφει την μοναδική ονομασία του LayerST.
- TemporalGeometryEncoder getGeometryEncoder(): Επιστρέφει την συγκεκριμένη υλοποίηση Encoder που έχει χρησιμοποιηθεί σε αυτό το LayerST για την αποθήκευση και ανάκτηση χωροχρονικών αντικειμένων.
- CoordinateReferenceSystem getCoordinateReferenceSystem(): Επιστρέφει το συγκεκριμένο CoordinateReferenceSystem που χρησιμοποιείται στο LayerST.
- String[] getExtraPropertyNames(): Επιστρέφει τυχόν extra geometry properties που υπάρχουν στο LayerST.
- Integer getGeometryType(): Επιστρέφει την μοναδική κατηγορία χωροχρονικών αντικειμένων που αποθηκεύονται στο LayerST.
- SpatialTemporalDatabaseService getSpatialDatabase(): Επιστρέφει το instance του SpatialTemporalDatabaseService που έχει χρησιμοποιηθεί για την κατασκευή του LayerST.
- SpatialTemporalDataset getDataset(): Επιστρέφει το instance του SpatialTemporalDataset που έχει χρησιμοποιηθεί για την κατασκευή του LayerST.
- Object getStyle(): Επιστρέφει το συγκεκριμένο style (SLD) για την οπτική απεικόνιση του περιεχομένου του LayerST.
- PropertyMappingManagerTemporal getPropertyMappingManager():Επιστρέφει το instance του PropertyMappingManagerTemporal που έχει χρησιμοποιηθεί για την κατασκευή του LayerST.

8.2.4.2 Κλάση SpatialTemporalDataset

Η κλάση SpatialTemporalDataset είναι ένα interface που χρησιμοποιείται για την αναπαράσταση ενός συγκεκριμένου Dataset που περιέχει ομοειδή γεωγραφικά αντικείμενα. Ένα συγκεκριμένο Dataset μπορεί να περιέχει ένα η περισσότερα layers και συγκεκριμένες μεθόδους για decoding και encoding για την ανάκτηση των δεδομένων που καθορίζονται από το αντικείμενο TemporalGeometryEncoder. Περιέχει τις παρακάτω μεθόδους προς υλοποίηση:

- Iterable<Node> getAllGeometryNodes(): Επιστρέφει όλους τους κόμβους που περιέχον χωρικά αντικείμενα.
- Iterable< ? extends Geometry> getAllGeometries(): Επιστρέφει όλα τα χωρικά αντικείμενα του Dataset.
- TemporalGeometryEncoder getGeometryEncoder(): Επιστρέφει το instance του TemporalGeometryEncoder που έχει χρησιμοποιηθεί για την αποθήκευση και ανάκτηση των χωροχρονικών αντικειμένων.
- Iterable< ? extends LayerST> getLayers(): Επιστρέφει ένα ή περισσότερα Layers που έχουν χρησιμοποιηθεί στο συγκεκριμένο Dataset για την αποθήκευση των αντικειμένων.
- boolean containsGeometryNode(Node node): Έλεγχος αν ο geometry node που δίνεται ως όρισμα υπάρχει στο συγκεκριμένο Dataset.

8.2.4.3 Κλάση TemporalConstants

Η κλάση TemporalConstants είναι ένα interface που περιέχει τιμές σταθερών που χρησιμοποιούνται σε όλες τις κλάσεις του Neo4jSpatioTemporal και ειδικότερα στην κλάση Hybrid3DRTreeIndex που αποτελεί την υλοποίηση των υβριδικών ευρετηρίων και των αλγοριθμικών μεθόδων αναζήτησης, όπως έχει περιγραφεί σε προηγούμενη ενότητα.

8.2.4.4 Κλάση SpatialTemporalRecord

Η κλάση SpatialTemporalRecord είναι ένα interface που χρησιμοποιείται για την αναπαράσταση μίας συγκεκριμένης εγγραφής στη βάση που αντιστοιχεί σε ένα χωροχρονικό αντικείμενο. Περιέχει τις παρακάτω μεθόδους προς υλοποίηση:

- String getId(): Επιστρέφει το συγκεκριμένο id της εγγραφής.
- Geometry getGeometry():Επιστρέφει το γεωμετρικό αντικείμενο της εγγραφής.
- boolean hasProperty(String name): Έλεγχος αν η εγγραφή έχει τη συγκεκριμένη ιδιότητα.
- String[] getPropertyNames(): Επιστροφή σε String[] των ονομάτων των ιδιοτήτων της εγγραφής.
- Object getProperty(String name): Επιστρέφει το property με τη συγκεκριμένη ονομασία.
- Map<String, Object> getProperties(): Επιστρέφει ζευγάρια ιδιοτήτων με τις τιμές τους.

8.2.4.5 Κλάση SpatialTemporalDatabaseRecord

Η κλάση SpatialTemporalDatabaseRecord αποτελεί υλοποίηση των interfaces TemporalConstants και SpatialTemporalRecord που αναφέρθηκαν προηγουμένως. Περιέχει δηλαδή τις παρακάτω μεθόδους από τα interfaces που υλοποιεί:

- String getId(): Επιστρέφει το συγκεκριμένο id της εγγραφής.
- long getNodeId(): Επιστρέφει το συγκεκριμένο node id.
- Node getGeomNode(): Επιστρέφει τον κόμβο που περιέχει το χωρικό αντικείμενο.
- int getType(): Επιστρέφει τον γεωμετρικό τύπο της εγγραφής.
- Geometry getGeometry(): Επιστρέφει το χωροχρονικό αντικείμενο της εγγραφής.
- CoordinateReferenceSystem getCoordinateReferenceSystem(): Επιστρέφει το CoordinateReferenceSystem της εγγραφής.
- String getLayerName(): Επιστρέφει την ονομασία του LayerST της εγγραφής.
- boolean hasProperty(String name): Έλεγχος αν η εγγραφή έχει τη συγκεκριμένη ιδιότητα.
- boolean hasGeometryProperty(String name): Έλεγχος αν η εγγραφή έχει τη συγκεκριμένη χωροχρονική ιδιότητα με χρήση του TemporalGeometryEncoder.
- String[] getPropertyNames(): Επιστροφή σε String[] των ονομάτων των ιδιοτήτων της εγγραφής.
- Object[] getPropertyValues(): Επιστροφή σε Object[] τις τιμές των ιδιοτήτων της εγγραφής.
- Map<String,Object> getProperties(): Επιστρέφει ζευγάρια ιδιοτήτων με τις τιμές τους.
- Object getProperty(String name): Επιστρέφει την εγγραφή με τη συγκεκριμένη ιδιότητα.

- Object getGeometryProperty(String name): Επιστρέφει την εγγραφή με τη συγκεκριμένη χωροχρονική ιδιότητα με χρήση του TemporalGeometryEncoder.
- void setProperty(String name, Object value): Εισαγωγή συγκεκριμένης ιδιότητας.
- int hashcode(): Επιστροφή του hashcode της εγγραφής.
- boolean equals(Object anotherObject): Έλεγχος αν δύο εγγραφές είναι ίσες με έλεγχο του node id τους.
- String toString(): Επιστρέφει σε μεταβλητή String πληροφορίες για τη συγκεκριμένη εγγραφή.
- void checkIsNotReservedProperty(String name): Έλεγχος αν μπορεί να τροποποιηθεί μία συγκεκριμένη property (ιδιότητα RESERVE_PROPS).
- String getPropString():Επιστρέφει σε μεταβλητή String πληροφορίες για τη συγκεκριμένη property.

8.2.4.6 Κλάση SpatialTemporalDatabaseService

Η κλάση SpatialTemporalDatabaseService είναι μία κλάση που υλοποιεί το interface TemporalConstants. Αποτελεί μία από τις βασικές κλάσεις του Neo4jSpatiotemporal καθώς με αυτήν την κλάση είναι δυνατή η δημιουργία και ανάκτηση ενός αντικειμένου LayerST που περιέχει τα χωροχρονικά αντικείμενα. Περιέχει τις παρακάτω μεθόδους:

- Node getOrCreateRootFrom(Node ref, RelationshipType relType): Δημιουργία root node με property «type» με single Relationship relType ή επιστροφή αυτού αν υπάρχει ήδη.
- Node getSpatialRoot(): Επιστροφή του root Node.
- String[] getLayerNames(): Επιστρέφει το όνομα του ενός ή περισσοτέρων LayerST που υπάρχουν.
- LayerST getLayer(String name): Επιστρέφει το αντικείμενο LayerST με το συγκεκριμένο όνομα.
- LayerST getDynamicLayer(String name): Επιστρέφει το DynamicLayerTemporal με το συγκεκριμένο όνομα.
- DynamicLayerTemporal asDynamicLayer(LayerST layer): Μετατροπή ενός LayerST σε DynamicLayerTemporal, δίνοντας έτσι τη δυνατότητα δημιουργία «όψεων» (dynamic layers).
- DefaultLayerTemporal getOrCreateDefaultLayer(String name): Επιστρέφει ή δημιουργεί ένα αντικείμενο DefaultLayerTemporal με το συγκεκριμένο όνομα.
- EditableLayerTemporal getOrCreateEditableLayer(String name, String format, String propertyNameConfig) : Επιστρέφει ή δημιουργεί ένα αντικείμενο EditableLayerTemporal με το συγκεκριμένο όνομα και παραμέτρους.

- EditableLayerTemporal getOrCreateEditableLayer(String name) : Επιστρέφει ή δημιουργεί ένα αντικείμενο DefaultLayerTemporal με το συγκεκριμένο όνομα.
- EditableLayerTemporal getOrCreateEditableLayer(String name, String wktProperty) : Επιστρέφει ή δημιουργεί ένα αντικείμενο DefaultLayerTemporal με το συγκεκριμένο όνομα και WKT property.
- EditableLayerTemporal getOrCreatePointLayer(String name, String xProperty, String yProperty, String timestamp) : Επιστρέφει ή δημιουργεί ένα αντικείμενο PointLayer με το συγκεκριμένο όνομα και παραμέτρους.
- LayerST getOrCreateLayer(String name, Class< ? extends TemporalGeometryEncoder> geometryEncoder, Class< ? extends Layer> layerClass, String config) : Επιστρέφει ή δημιουργεί ένα αντικείμενο LayerST με συγκεκριμένο TemporalGeometryEncoder και παραμέτρους.
- LayerST getOrCreateLayer(String name, Class< ? extends TemporalGeometryEncoder> geometryEncoder, Class< ? extends LayerST> layerClass) : Επιστρέφει ή δημιουργεί ένα αντικείμενο LayerST με συγκεκριμένο TemporalGeometryEncoder.
- LayerST findLayerContainingGeometryNode(Node geometryNode) : Επιστρέφει το αντικείμενο LayerST με το συγκεκριμένο Geometry Node.
- LayerST getLayerFromChild(Node child, RelationshipType relType): Δημιουργία LayerST με root node το child.
- boolean containsLayer(String name): Έλεγχος αν υπάρχει LayerST με συγκεκριμένη ονομασία.
- LayerST createWKBLayer(String name): Δημιουργεί ένα αντικείμενο WKB Layer με συγκεκριμένη ονομασία.
- SimplePointLayerST createSimplePointLayer(String name) : Δημιουργεί ένα αντικείμενο SimplePointLayerST με συγκεκριμένη ονομασία.
- SimplePointLayerST createSimplePointLayer(String name, String xProperty, String yProperty, String timestamp) : Δημιουργεί ένα αντικείμενο SimplePointLayerST με συγκεκριμένη ονομασία και παραμέτρους.
- LayerST createLayer(String name, Class<? extends Geometry> geometryEncoderClass, Class<? extends LayerST> layerClass) : Επιστρέφει ή δημιουργεί ένα αντικείμενο LayerST με συγκεκριμένο TemporalGeometryEncoder.
- LayerST createLayer(String name, Class<? extends Geometry> geometryEncoderClass, Class<? extends LayerST> layerClass, String encoderConfig) : Επιστρέφει ή δημιουργεί ένα αντικείμενο LayerST με συγκεκριμένο TemporalGeometryEncoder και παραμέτρους.
- LayerST createLayer(String name, Class<? extends TemporalGeometryEncoder> geometryEncoderClass, Class<? extends LayerST> layerClass, String encoderConfig, CoordinateReferenceSystem crs) :

Επιστρέφει ή δημιουργεί ένα αντικείμενο LayerST με συγκεκριμένο TemporalGeometryEncoder, παραμέτρους και CoordinateReferenceSystem crs.
- void deleteLayer(String name, Listener monitor): Διαγραφή του LayerST με τη συγκεκριμένη ονομασία.
- GraphDatabaseService getDatabase(): Επιστρέφει το instance GraphDatabaseService που χρησιμοποιήθηκε για την δημιουργία του αντικειμένου.
- int convertGeometryNameToType(String geometryName): Επιστρέφει τον κωδικό αριθμό της συγκεκριμένης γεωμετρίας.
- String convertGeometryTypeToName(Integer geometryType): Επιστρέφει την ονομασία της συγκεκριμένης γεωμετρίας.
- Class<? extends Geometry> convertGeometryTypeToJtsClass(Integer geometryType): Επιστροφή της JTS κλάσης που αντιστοιχεί στο συγκεκριμένο κωδικό αριθμό.
- int convertJtsClassToGeometryType(Class<? extends Geometry> jtsClass) : Επιστροφή του κωδικού αριθμού της συγκεκριμένης γεωμετρικής κλάσης.
- Layer createResultsLayer(String layerName, List<SpatialTemporal DatabaseRecord> results): Δημιουργία LayerST με γεωμετρικά αντικείμενα που έχουν προκύψει από τα αποτελέσματα συγκεκριμένου query.

8.2.4.7 Κλάση TemporalGeometryEncoder

Η κλάση TemporalGeometryEncoder είναι ένα interface που αφορά την υλοποίηση custom encoders για την αποθήκευση και ανάκτηση χωροχρονικών αντικειμένων από τη βάση δεδομένων Neo4j. Περιέχει τις παρακάτω μεθόδους προς υλοποίηση:

- void init(Layer layer): Αρχικοποίηση του layer με προσθήκη της ονομασίας της υλοποίησης του TemporalGeometryEncoder σε property στον βασικό κόμβο του layer.
- void encodeGeometry(Geometry geometry, PropertyContainer container): Μέθοδος που ορίζει τον τρόπο αποθήκευσης των χωρικών πληροφοριών ενός γεωμετρικού αντικειμένου στη Neo4j βάση.
- Geometry decodeGeometry(PropertyContainer container): Μέθοδος που ορίζει τον τρόπο ανάκτησης των χωρικών αντικειμένων από τη Neo4j βάση.
- boolean hasAttribute(Node geomNode, String name): Έλεγχος αν ο συγκεκριμένος κόμβος έχει property με ονομασία name.
- Object getAttribute(Node geomNode, String name): Επιστρέφει την τιμή του property name από τον κόμβο geomNode.

8.2.4.8 Κλάση AbstractTemporalGeometryEncoder

Η κλάση AbstractGeometryEncoder είναι abstract class και αποτελεί υλοποίηση των interfaces TemporalGeometryEncoder και TemporalConstants που αναφέρθηκαν προηγουμένως. Περιέχει τις παρακάτω μεθόδους:

- void init(LayerST layer) : Αρχικοποίηση του layer με προσθήκη της ονομασίας της υλοποίησης του TemporalGeometryEncoder σε property στον βασικό κόμβο του layer.
- void encodeGeometry(Geometry geometry, PropertyContainer container): Αποθήκευση του IndexEnvelope που περιβάλλει το γεωμετρικό αντικείμενο στη βάση σε Node με κατάλληλο property «bbox».
- void encodeEnvelope(IndexEnvelope mbb, PropertyContainer container): Αποθήκευση του αντικειμένου IndexEnvelope στη βάση σε Node με κατάλληλο property «bbox».
- IndexEnvelope decodeEnvelope(PropertyContainer container): Επιστρέφει ένα αντικείμενο IndexEnvelope με χαρακτηριστικά σύμφωνα με το όρισμα.
- protected abstract void encodeGeometryShape(Geometry geometry, PropertyContainer container): abstract κλάση προς υλοποίηση του αντικειμένου.
- protected Integer encodeGeometryType(String jtsGeometryType): Επιστροφή του κωδικού αριθμού της συγκεκριμένης γεωμετρικής κλάσης.
- boolean hasAttribute(Node geomNode, String name): Έλεγχος αν ο συγκεκριμένος κόμβος περιέχει το property name.
- Object getAttribute(Node geomNode, String name): Επιστρέφει την τιμή του property name του κόμβου geomNode.

8.2.4.9 Κλάση DefaultLayerTemporal

Η κλάση DefaultLayerTemporal αποτελεί υλοποίηση των interfaces LayerST, SpatialTemporalDataset και TemporalConstants που αναφέρθηκαν προηγουμένως. Περιέχει τις παρακάτω μεθόδους:

- String getName(): Επιστρέφει την ονομασία του LayerST.
- SpatialTemporalDatabaseService getSpatialDatabase():Επιστρέφει το instance του SpatialTemporalDatabaseService που έχει χρησιμοποιηθεί.
- LayerIndexReaderTemporal getIndex(): Επιστρέφει το ευρετήριο του LayerST.
- SpatialTemporalDatabaseRecord add(Node geomNode) : Προσθήκη ενός node με χωρικό αντικείμενο στο LayerST.

- TemporalGeometryFactory getGeometryFactory(): Επιστρέφει ένα TemporalGeometryFactory instance για τη δημιουργία καινούριων χωρικών αντικειμένων.
- void setCoordinateReferenceSystem(CoordinateReferenceSystem crs): Ορισμός του Coordinate Reference System.
- CoordinateReferenceSystem getCoordinateReferenceSystem(): Επιστρέφει το Coordinate Reference System που χρησιμοποιείται.
- void setGeometryType(Integer geometryType): Καθορισμός της γεωμετρικής κατηγορίας που θα χρησιμοποιηθεί.
- Integer getGeometryType(): Επιστρέφει την κατηγορία των γεωμετρικών αντικειμένων που χρησιμοποιούνται.
- String[] getExtraPropertyNames(): Επιστρέφει σε String[] τις extra properties που έχουν χρησιμοποιηθεί κατά την δημιουργία των χωρικών αντικειμένων.
- void setExtraPropertyNames(String[] names): Καθορισμός με String[] επιπρόσθετων properties που έχουν χρησιμοποιηθεί κατά την δημιουργία των χωρικών αντικειμένων.
- void mergeExtraPropertyNames(String[] names): Καθορισμός με String[] επιπρόσθετων properties των χωρικών αντικειμένων.
- LayerST makeLayerFromNode(SpatialTemporalDatabaseService spatialDatabase, Node layerNode): Δημιουργία LayerST με root Node τον layerNode.
- LayerST makeLayerAndNode(SpatialTemporalDatabaseService spatialDatabase, String name,Class< ? extends TemporalGeometryEncoder> geometryEncoderClass, Class< ? extends LayerST> layerClass): Δημιουργία LayerST και καταλλήλου root Node για αυτό.
- LayerST makeLayerInstance(SpatialTemporalDatabaseService spatialDatabase, String name, Node layerNode, Class<? extends Layer> layerClass) : Δημιουργία LayerST instance με root Node τον layerNode.
- initialize(SpatialTemporalDatabaseService spatialDatabase, String name, Node layerNode): Αρχικοποίηση του LayerST με αρχικό node τον layerNode.
- Node getLayerNode(): Επιστρέφει τον root Node του LayerST.
- void delete(Listener monitor): Διαγραφή των στοιχείων του LayerST.
- GraphDatabaseService getDatabase(): Επιστρέφει το instance του GraphDatabaseService που έχει χρησιμοποιηθεί.
- SpatialTemporalDataset getDataset(): Επιστρέφει το instance του SpatialTemporalDataset που έχει χρησιμοποιηθεί.
- Iterable<Node> getAllGeometryNodes(): Επιστρέφει όλους τους γεωμετρικούς κόμβους του LayerST.
- boolean containsGeometryNode(Node geomNode): Έλεγχος αν ο συγκεκριμένος κόμβος περιέχει χωρική πληροφορία.

- Iterable<? extends Geometry> getAllGeometries(): Επιστρέφει όλα τα γεωμετρικά αντικείμενα του LayerST.
- TemporalGeometryEncoder getGeometryEncoder(): Επιστρέφει το TemporalGeometry Encoder που έχει χρησιμοποιηθεί στο συγκεκριμένο LayerST.
- Iterable< ? extends LayerST-> getLayers(): Επιστρέφει τα layers που έχουν χρησιμοποιηθεί από το συγκεκριμένο Dataset.
- Object getStyle(): Επιστρέφει το συγκεκριμένο style (SLD) για την οπτική απεικόνιση του περιεχομένου του Layer.
- PropertyMappingManagerTemporal getPropertyMappingManager():Επιστρέφει το instance του PropertyMappingManagerTemporal που έχει χρησιμοποιηθεί για την κατασκευή του LayerST.

8.2.4.10 Κλάση DynamicLayerTemporal

Η κλάση DynamicLayerTemporal αποτελεί επέκταση της κλάσης EditableLayerImplTemporal. Με την συγκεκριμένη κλάση δίνεται δυνατότητα σε ένα αντικείμενο LayerST να μπορεί «εικονικά» να διαιρεθεί σε επιμέρους Layers (δηλαδή έχει τον ρόλο όψης), βάσει συγκεκριμένων κριτηρίων (DynamicLayerConfigTemporal αντικείμενα που αναλύονται στη συνέχεια). Περιέχει τις παρακάτω μεθόδους:

- Synchronized Map<String, LayerST> getLayerMap(): Επιστρέφει τα διαφορετικά dynamic layers με την αντίστοιχη ονομασία τους.
- boolean removeLayerConfig(String name): Διαγραφή dynamic layer με τη συγκεκριμένη ονομασία.
- String makeGeometryName(int gtype): Επιστρέφει την ονομασία μιας γεωμετρίας σε String ανάλογα με την κωδική ονομασία της.
- String makeGeometryCQL(int gtype): Επιστρέφει CQL syntax query για την γεωμετρία με την συγκεκριμένη κωδική ονομασία.
- DynamicLayerConfigTemporal addCQLDynamicLayerOnGeometryType(int gtype): Δημιουργεί DynamicLayerConfigTemporal αντικείμενο με κριτήριο CQL syntax query για γεωμετρίες με την συγκεκριμένη κωδική ονομασία.
- DynamicLayerConfigTemporal addCQLDynamicLayerOnAttribute(String key, String value, int gtype): Δημιουργεί DynamicLayerConfigTemporal αντικείμενο με κριτήριο CQL syntax query για γεωμετρίες με την συγκεκριμένη κωδική ονομασία.
- DynamicLayerConfigTemporal addCQLDynamicLayerOnAttributes(String[] attributes, int gtype): Δημιουργεί DynamicLayerConfigTemporal αντικείμενο με κριτήριο CQL syntax query με συγκεκριμένες attributes.

- DynamicLayerConfigTemporal addLayerConfig(String name, int type, String query): Δημιουργεί DynamicLayerConfigTemporal αντικείμενο με κριτήριο CQL syntax query.
- DynamicLayerConfigTemporal restrictLayerProperties(String name, String[] names): Δημιουργεί DynamicLayerConfigTemporal αντικείμενο με κριτήριο συγκεκριμένα layer properties.
- DynamicLayerConfigTemporal restrictLayerProperties(String name): Δημιουργεί DynamicLayerConfigTemporal αντικείμενο με κριτήριο όσα layer properties υπάρχουν αυτή τη στιγμή στο LayerST.
- List<String> getLayerNames(): Επιστρέφει την ονομασία των layers σε List<String>.
- List<LayerST> getLayers(): Επιστρέφει τα layers σε List<LayerST>.
- LayerST getLayer(String name): Επιστρέφει το LayerST με ονομασία name.

8.2.4.11 Κλάση DynamicLayerConfigTemporal

Η κλάση DynamicLayerConfigTemporal αποτελεί υλοποίηση των interfaces LayerST, SpatialTemporalDataset και TemporalConstants που αναφέρθηκαν προηγουμένως και ορίζονται σε αυτήν οι παράμετροι για τη δημιουργία ενός DynamicLayerTemporal (read-only view). Περιέχει τις παρακάτω μεθόδους:

- String getName(): Επιστρέφει την ονομασία του αντικειμένου.
- String getQuery(): Επιστρέφει τη σύνταξη του query.
- SpatialTemporalDatabaseRecord add(Node geomNode): Μη χρησιμοποίηση της μεθόδου καθώς ένα dynamic layer είναι read-only. Υπάρχει λόγω υλοποίησης interface κλάσης.
- void delete(Listener monitor): Μη χρησιμοποίηση της μεθόδου καθώς ένα dynamic layer είναι read-only. Υπάρχει λόγω υλοποίησης interface κλάσης.
- CoordinateReferenceSystem getCoordinateReferenceSystem(): Επιστρέφει το Coordinate Reference System που χρησιμοποιείται.
- SpatialTemporalDataset getDataset(): Επιστρέφει το instance του συγκεκριμένου SpatialTemporalDataset που χρησιμοποιείται από το DynamicLayerTemporal που αφορά το συγκεκριμένο αντικείμενο.
- String[] getExtraPropertyNames(): Επιστρέφει σε String[] τις extra properties που έχουν χρησιμοποιηθεί κατά την δημιουργία των χωρικών αντικειμένων.
- void restrictLayerProperties(): Αναζήτηση στο layer των properties που πραγματικά χρησιμοποιούνται και χρησιμοποίηση τους στο συγκεκριμένο dynamic layer.
- void setExtraPropertyNames(String[] names): Καθορισμός extra properties.

- TemporalGeometryEncoder getGeometryEncoder(): Επιστρέφει το αντικείμενο TemporalGeometryEncoder που έχει χρησιμοποιηθεί στο συγκεκριμένο LayerST.
- TemporalGeometryFactory getGeometryFactory(): Επιστρέφει το αντικείμενο TemporalGeometryFactory που έχει χρησιμοποιηθεί στο συγκεκριμένο LayerST.
- Integer getGeometryType(): Επιστρέφει το συγκεκριμένο Geometry type για το συγκεκριμένο DynamicLayerTemporal αντικείμενο.
- LayerIndexReaderTemporal getIndex(): Επιστρέφει instance του αντικειμένου LayerIndexReader για την ανάγνωση του index του LayerST.
- Node getLayerNode(): Επιστρέφει τον αρχικό κόμβο του LayerST.
- SpatialTemporalDatabaseService getSpatialDatabase(): Επιστρέφει το instance του SpatialTemporalDatabaseService που έχει χρησιμοποιηθεί.
- void initialize(SpatialTemporalDatabaseService spatialDatabase, String name, Node layerNode)
- Object getStyle(): Επιστρέφει το συγκεκριμένο style (SLD) για την οπτική απεικόνιση του περιεχομένου του LayerST.
- LayerST getParent(): Επιστρέφει το DynamicLayerTemporal που αφορά το συγκεκριμένο αντικείμενο και έχει οριστεί στον κατασκευαστή του.
- String toString(): Επιστρέφει σε String πληροφορίες για το συγκεκριμένο configuration.
- PropertyMappingManagerTemporal getPropertyMappingManager():Επιστρέφει το instance του PropertyMappingManagerTemporal που έχει χρησιμοποιηθεί για την κατασκευή του LayerST.

8.2.4.12 EditableLayerTemporal

Η κλάση EditableLayerTemporal είναι interface που αφορά την υλοποίηση αντικειμένων LayerST που δίνουν τη δυνατότητα προσθήκης, αφαίρεσης, επεξεργασίας γεωγραφικών αντικειμένων σε αυτό. Περιέχει τις παρακάτω μεθόδους:

- SpatialTemporalDatabaseRecord add(Geometry geometry): Προσθήκη γεωμετρικής εγγραφής στο LayerST.
- SpatialTemporalDatabaseRecord add(Geometry geometry, String[] fieldsName, Object[] fields): Προσθήκη εγγραφής με συγκεκριμένα properties στο layer.
- void delete(long geometryNodeId): Διαγραφή geometry node με id geometryNodeId.
- void update(long geometryNodeId, Geometry geometry): Επανακαθορισμός της γεωμετρίας ενός geometry node με id geometryNodeId.
- void setCoordinateReferenceSystem(CoordinateReferenceSystem coordinateReferenceSystem): Καθορισμός του Coordinate Reference System.

- void removeFromIndex(long geomNodeId): Αφαίρεση από το index ενός geometry node με id geometryNodeId.

8.2.4.13 EditableLayerImplTemporal

Η κλάση EditableLayerImplTemporal αποτελεί υλοποίηση των interfaces DefaultLayerTemporal και EditableLayerTemporal που αναφέρθηκαν προηγουμένως. Περιέχει τις παρακάτω μεθόδους:

- SpatialTemporalDatabaseRecord add(Geometry geometry): Προσθήκη γεωμετρικής εγγραφής στο LayerST.
- SpatialTemporalDatabaseRecord add(Geometry geometry, String[] fieldsName, Object[] fields): Προσθήκη εγγραφής με συγκεκριμένα properties στο layer.
- void update(long geometryNodeId, Geometry geometry): Επανακαθορισμός της γεωμετρίας ενός geometry node με id geometryNodeId.
- void delete(long geometryNodeId): Διαγραφή geometry node με id geometryNodeId.
- void removeFromIndex(long geomNodeId): Αφαίρεση από το index ενός geometry node με id geometryNodeId.
- protected Node addGeomNode(Geometry geom, String[] fieldsName, Object[] fields): Προσθήκη Geometry node με συγκεκριμένα properties στο layer.

8.2.4.14 LayerIndexReaderTemporal

Η κλάση LayerIndexReaderTemporal είναι ένα interface που αποτελεί επέκταση του interface SpatialTemporalIndexReader, που αναφέρθηκε σε προηγούμενη ενότητα. Περιλαμβάνει τις παρακάτω μεθόδους προς υλοποίηση:

- LayerST getLayer(): Επιστροφή του LayerST που χρησιμοποιείται για ανάγνωση.
- SpatialTemporalDatabaseRecord get(Long geomNodeId): Επιστροφή της χωρικής εγγραφής με συγκεκριμένο Node id.
- List<SpatialTemporalDatabaseRecord> get(Set<Long> geomNodeIds): Επιστροφή του συνόλου των geometry node ids του Layer.
- TemporalSearchRecords search(SearchFilter filter): Αναζήτηση με συγκεκριμένα κριτήρια σύμφωνα με το instance του αντικειμένου TemporalSearchFilter.

8.2.4.15 LayerTreeIndexReaderTemporal

Η κλάση LayerTreeIndexReaderTemporal είναι ένα interface που αποτελεί επέκταση του interface LayerIndexReaderTemporal, που αναφέρθηκε προηγουμένως. Περιλαμβάνει τις παρακάτω μεθόδους προς υλοποίηση:

- Node getIndexRoot(): Επιστρέφει το root Node.
- void visit(SpatialTemporalIndexVisitor visitor, Node indexNode): Διάσχιση του δέντρου του ευρετηρίου.

8.2.4.16 LayerRTreeIndexReaderTemporal

Η κλάση LayerRTreeIndexReaderTemporal αποτελεί υλοποίηση των interfaces LayerTreeIndexReaderTemporal και TemporalConstants που αναφέρθηκαν προηγουμένως. Περιλαμβάνει τις παρακάτω μεθόδους:

- LayerST getLayer(): Επιστρέφει το LayerST που έχει οριστεί από τον κατασκευαστή κατά τη δημιουργία του αντικειμένου.
- SpatialTemporalDatabaseRecord get(Long geomNodeId): Επιστρέφει την χωρική εγγραφή που ανήκει σε geometry node με id geomNodeId.
- List<SpatialTemporalDatabaseRecord> get(Set<Long> geomNodeIds): Επιστροφή του συνόλου των geometry node ids του Layer.
- TemporalSearchRecords search(SearchFilter filter): Αναζήτηση με συγκεκριμένα κριτήρια σύμφωνα με το instance του αντικειμένου TemporalSearchFilter.

8.2.4.17 Neo4jFeatureBuilderTemporal

Η κλάση Neo4jFeatureBuilderTemporal είναι μία κλάση που χρησιμοποιείται για την ανάκτηση και επεξεργασία geometry properties από ένα συγκεκριμένο αντικείμενο LayerST.

8.2.4.18 OrderedEditableLayerTemporal

Η κλάση OrderedEditableLayerTemporal αποτελεί επέκταση της κλάσης EditableLayerImplTemporal που αναφέρθηκε σε προηγούμενη ενότητα. Η

συγκεκριμένη κλάση δίνει τη δυνατότητα iteration στα geometry objects της βάσης με τη σειρά που δημιουργήθηκαν και έγινε εισαγωγή τους στη βάση. Περιλαμβάνει τις παρακάτω μεθόδους:

- Node addGeomNode(Geometry geom, String[] fieldsName, Object[] fields): Προσθήκη στο layer γεωμετρίας με συγκεκριμένες παραμέτρους.
- Iterable<Node> getAllGeometryNodes(): Επιστροφή όλων των geometry nodes του LayerST.

8.2.4.19 SimplePointLayerST

Η κλάση SimplePointLayerST αποτελεί επέκταση της κλάσης EditableLayerImplTemporal που αναφέρθηκε σε προηγούμενη ενότητα. Η συγκεκριμένη κλάση δίνει τη δυνατότητα δημιουργίας LayerST για αποθήκευση και αναζήτηση χωροχρονικών σημείων (αντικείμενα TemporalPoint). Περιλαμβάνει τις παρακάτω μεθόδους:

- SpatialTemporalDatabaseRecord add(TemporalCoordinate coordinate): Προσθήκη στο LayerST μίας εγγραφής με συγκεκριμένες χωροχρονικές συνταταγμένες.
- SpatialTemporalDatabaseRecord add(TemporalCoordinate coordinate, String[] fieldsName, Object[] fields): Προσθήκη στο LayerST μίας εγγραφής με συγκεκριμένες χωροχρονικές συνταταγμένες και παραμέτρους.
- List<GeoPipeFlowTemporal> findClosestPointsTo(TemporalCoordinate coordinate, double d): Αναζήτηση πλησιέστερων χωροχρονικές σημείων σε απόσταση d.
- List<GeoPipeFlowTemporal> findClosestPointsTo(TemporalCoordinate coordinate, int numberOfItemsToFind): Αναζήτηση συγκεκριμένου αριθμού πλησιέστερων χωροχρονικές σημείων.
- List<GeoPipeFlowTemporal> findClosestPointsTo(TemporalCoordinate coordinate): Αναζήτηση πλησιέστερων χωροχρονικές σημείων.

8.2.4.20 UtilitiesST

Η κλάση UtilitiesST περιλαμβάνει μεθόδους για την μετατροπή ενός αντικειμένου MBB της library JTS σε ένα αντικείμενο IndexEnvelope του πακέτου Hybrid3DRTree και μεθόδους αναζήτησης που χρησιμοποιούν το αντικείμενο IndexEnvelope.

8.2.4.21 WKBTemporalGeometryEncoder

Η κλάση WKBTemporalGeometryEncoder αποτελεί υλοποίηση της abstract κλάσης AbstractTemporalGeometryEncoder που αναφέρθηκε σε προηγούμενη ενότητα. Η συγκεκριμένη κλάση δίνει τη δυνατότητα encoding και decoding γεωγραφικών αντικειμένων χρησιμοποιώντας το WKB (Well-Known Binary) format. Περιλαμβάνει τις παρακάτω μεθόδους:

- Geometry decodeGeometry(PropertyContainer container): Ανάκτηση γεωμετρίας από τη βάση.
- void encodeGeometryShape(Geometry geometry, PropertyContainer container): Αποθήκευση γεωμετρίας στη βάση.
- void setConfiguration(String configuration): Ορισμός την ονομασίας του WKB property στον geometry node.
- String getConfiguration(): Επιστρέφει την ονομασία του WKB property.

8.2.4.22 WKTTemporalGeometryEncoder

Η κλάση WKTTemporalGeometryEncoder αποτελεί υλοποίηση της abstract κλάσης AbstractTemporalGeometryEncoder που αναφέρθηκε σε προηγούμενη ενότητα. Η συγκεκριμένη κλάση δίνει τη δυνατότητα encoding και decoding γεωγραφικών αντικειμένων χρησιμοποιώντας το WKT (Well-Known Text) format. Περιλαμβάνει τις παρακάτω μεθόδους:

- Geometry decodeGeometry(PropertyContainer container): Ανάκτηση γεωμετρίας από τη βάση.
- void encodeGeometryShape(Geometry geometry, PropertyContainer container): Αποθήκευση γεωμετρίας στη βάση.
- void setConfiguration(String configuration): Ορισμός την ονομασίας του WKT property στον geometry node.
- String getConfiguration(): Επιστρέφει την ονομασία του WKT property.

8.2.4.23 PropertyMappingManagerTemporal

Η κλάση PropertyMappingManagerTemporal αποτελεί κλάση για την ανάκτηση και αποθήκευση της PROPERTY_MAPPING property. Περιλαμβάνει τις παρακάτω μεθόδους που αφορούν την αποθήκευση, ανάκτηση και αφαίρεση με προφανή ονομασία:

- LinkedHashMap<String, PropertyMapper> getPropertyMappers()
- Map<Node, PropertyMapper> loadMappers()
- void save()
- void addPropertyMapper(PropertyMapper mapper)
- PropertyMapper removePropertyMapper(String to)
- PropertyMapper getPropertyMapper(String to)
- void addPropertyMapper(String from, String to, String type, String params)

8.2.4.24 LayerIndexReaderWrapperTemporal

Η κλάση LayerIndexReaderWrapperTemporal αποτελεί υλοποίηση του interface LayerIndexReaderTemporal που αναφέρθηκε σε προηγούμενη ενότητα. Περιλαμβάνει τις παρακάτω μεθόδους:

- LayerST getLayer(): Επιστρέφει το instance του layer που χρησιμοποιείται.
- IndexEnvelopeDecoder getEnvelopeDecoder(): Επιστρέφει το instance του IndexEnvelopeDecoder που χρησιμοποιείται για την κατασκευή των αντικειμένων IndexEnvelope.
- int count(): Απαρίθμηση των γεωγραφικών αντικειμένων στο ευρετήριο.
- boolean isNodeIndexed(Long nodeId) : Έλεγχος αν ο κόμβος με id nodeId υπάρχει στο ευρετήριο.
- SpatialTemporalDatabaseRecord get(Long geomNodeId): Επιστρέφει την χωροχρονική εγγραφή με node id geomNodeId.
- List<SpatialTemporalDatabaseRecord> get(Set<Long> geomNodeIds): Επιστρέφει το σύνολο των geometry node ids.
- IndexEnvelope getBoundingBox(): Επιστροφή του αντικειμένου IndexEnvelope που περιβάλλει το ευρετήριο.
- boolean isEmpty(): Έλεγχος αν ο root κόμβος περιέχει ή όχι την ιδιότητα «bbox».
- -public Iterable<Node> getAllIndexedNodes(): Επιστροφή όλων των κόμβων που περιέχουν γεωγραφικά αντικείμενα και υπάρχουν στο ευρετήριο.

- TemporalSearchResults searchIndex(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter και επιστροφή αντικειμένων TemporalSearchResults.
- TemporalSearchRecords search(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter και επιστροφή αντικειμένων TemporalSearchRecords.

8.2.4.25 DynamicIndexReaderTemporal

Η κλάση DynamicIndexReaderTemporal αποτελεί υλοποίηση του interface LayerIndexReaderWrapperTemporal που αναφέρθηκε στην προηγούμενη ενότητα. Η συγκεκριμένη κλάση επιτρέπει την διάσχιση του γράφου σύμφωνα με JSON queries. Περιλαμβάνει τις παρακάτω μεθόδους:

- boolean queryLeafNode(Node geomNode): Χρησιμοποίηση των δύο παρακάτω μεθόδων για υλοποίηση των JSON queries.
- boolean stepAndQuery(Node source, JSONObject step): Έλεγχος της δομής του JSON query αν αντιστοιχεί με τη δομή του γράφου.
- boolean queryNodeProperties(Node node, JSONObject properties): Έλεγχος αντιστοιχίας των Node properties με τα JSON properties για τον Node node.
- int count(): Απαρίθμηση των γεωγραφικών αντικειμένων στο ευρετήριο.
- TemporalSearchResults searchIndex(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter και επιστροφή αντικειμένων TemporalSearchResults.
- TemporalSearchRecords search(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter και επιστροφή αντικειμένων SearchRecords.

8.2.4.26 CQLIndexReaderTemporal

Η κλάση CQLIndexReaderTemporal αποτελεί υλοποίηση του interface LayerIndexReaderWrapperTemporal που αναφέρθηκε στην προηγούμενη ενότητα. Η συγκεκριμένη κλάση επιτρέπει την διάσχιση του γράφου σύμφωνα με CQL queries. Περιλαμβάνει τις παρακάτω μεθόδους:

- boolean queryIndexNode(IndexEnvelope indexNodeEnvelope): Ελέγχει αν γίνεται intersection μεταξύ του indexNodeEnvelope και του filterEnvelope που έχει

οριστεί στον κατασκευαστή κατά την αρχικοποίηση του αντικειμένου σε συνδυασμό με το λογικό (&&) αν το filterEnvelope είναι null.
- boolean queryLeafNode(Node indexNode): Έλεγχος αν το query επαληθεύεται στον συγκεκριμένο geometry node.
- TemporalSearchResults searchIndex(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter και επιστροφή αντικειμένων TemporalSearchResults.
- TemporalSearchRecords search(TemporalSearchFilter filter) : Αναζήτηση στο ευρετήριο με συγκεκριμένα κριτήρια, σύμφωνα με το αντικείμενο TemporalSearchFilter και επιστροφή αντικειμένων TemporalSearchRecords.

8.2.4.27 LayerNodeIndexTemporal

Η κλάση LayerNodeIndexTemporal αποτελεί υλοποίηση του interface TemporalConstants που αναφέρθηκε στην προηγούμενη ενότητα. Η συγκεκριμένη κλάση δίνει τη δυνατότητα δημιουργίας και αναζήτησης ενός αντικειμένου LayerST με κατηγορία αναλόγως των ορισμάτων που δίνονται στον κατασκευαστή του αντικειμένου (lat/lon-WKB-WKT). Περιλαμβάνει τις παρακάτω μεθόδους:

- String getName(): Επιστροφή της μεταβλητής name του LayerST.
- Class<Node> getEntityType(): Επιστροφή της κλάσης του αντικειμένου Node.
- void add(Node geometry, String key, Object value): Προσθήκη μίας συγκεκριμένης node property.
- Node findExistingNode(Node geometry): Αναζήτηση ύπαρξης ενός συγκεκριμένου Node.
- void remove(Node entity, String key, Object value) : Διαγραφή του συγκεκριμένου node.
- void delete(): Διαγραφή του LayerST.
- IndexHits<Node> get(String key, Object value): Μέθοδος αναζήτησης με συγκεκριμένα κριτήρια.
- IndexHits<Node> query(String arg0, Object arg1): Μέθοδος αναζήτησης με συγκεκριμένα κριτήρια.
- IndexHits<Node> query(String key, Object params): Μέθοδος αναζήτησης με συγκεκριμένα κριτήρια.
- IndexHits<Node> query(Object queryOrQueryObject) Μέθοδος αναζήτησης με συγκεκριμένα κριτήρια.
- void remove(Node node, String s): Αφαίρεση του συγκεκριμένου κόμβου καλώντας την παρακάτω μέθοδο.
- void remove(Node node): Αφαίρεση του συγκεκριμένου κόμβου.

- boolean isWriteable(): Επιστρέφει true (υποχρεωτική υλοποίηση από interface).
- GraphDatabaseService getGraphDatabase(): Επιστρέφει το instance του GraphDatabaseService που έχει χρησιμοποιηθεί.
- Node putIfAbsent(Node entity, String key, Object value): Επιστρέφει exception (υποχρεωτική υλοποίηση από interface).

8.2.4.28 SpatialTemporalIndexProviderTemporal

Η αρχική κλάση SpatialIndexProvider αποτελεί επέκταση της κλάσης IndexProvider του Neo4j. Η κλάση SpatialTemporalIndexProvider αποτελεί τροποποίηση της κλάσης SpatialIndexProvider έτσι ώστε οι μέθοδοι να περιλαμβάνουν το αντικείμενο LayerNodeIndexTemporal που περιέχει τη χρονική διάσταση της πληροφορίας.

8.2.4.29 SpatialTemporalKernelExtensionFactory

Η αρχική κλάση SpatialKernelExtensionFactory αποτελεί επέκταση της κλάσης KernelExtensionFactory του Neo4j.Η κλάση SpatialTemporalKernelExtensionFactory αποτελεί τροποποίηση της κλάσης SpatialKernelExtensionFactory έτσι ώστε οι μέθοδοι να περιλαμβάνουν το αντικείμενο SpatialTemporalIndexProvider που περιέχει τη χρονική διάσταση της πληροφορίας.

8.2.4.30 SpatialTemporalRecordHits

Η κλάση SpatialTemporalRecordHits αποτελεί επέκταση της κλάσης CathingIteratorWrapper του Neo4j και υλοποίηση της IndexHits<Node> που αναφέρθηκε σε προηγούμενη ενότητα. Περιλαμβάνει τις παρακάτω μεθόδους:

- int size(): Επιστρέφει την τιμή της μεταβλητής size, που έχει οριστεί στον κατασκευαστή του αντικειμένου.
- float currentScore(): Επιστρέφει 0 (Υποχρεωτική υλοποίηση από interface).
- Iterator<Node> iterator(): Επιστρέφει instance του αντικειμένου.
- void close(): Υποχρεωτική υλοποίηση από interface.
- Node getSingle(): Επιστρέφει ένα αντικείμενο Node σε περίπτωση μοναδικού αποτελέσματος.

- Node underlyingObjectToObject(SpatialTemporalDatabaseRecord object): Επιστρέφει αντικείμενο Node που αντιστοιχεί στη συγκεκριμένη εγγραφή.
- void itemDodged(SpatialTemporalDatabaseRecord item): Διαγραφή της συγκεκριμένης χωρικής εγγραφής.
- boolean exceptionOk(Throwable t): Επιστρέφει NotFoundException (υποχρεωτική υλοποίηση από interface).