ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΚΑΤΕΥΘΥΝΣΗ ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ ΙΣΤΟΥ

Τίτλος Εργασίας

ΣΥΣΤΗΜΑ ΕΥΡΕΣΗΣ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗΣ ΒΕΛΤΙΣΤΗΣ ΔΙΑΔΡΟΜΗΣ ΓΙΑ ΠΛΟΙΑ
VESSEL COURSE OPTIMIZATION AND VISUALIZATION

Αριστείδης Καραγιαννόπουλος

Αθήνα, 2020

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΚΑΤΕΥΘΥΝΣΗ ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ ΙΣΤΟΥ

Τριμελής Εξεταστική Επιτροπή

Όνομα Πρώτου Καθηγητή (Επιβλέπων)
Ηρακλής Βαρλάμης
Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Όνομα Δεύτερου Καθηγητή
Κωνσταντίνος Τσερπές
Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Όνομα Τρίτου Καθηγητή
Ανάργυρος Τσαδήμας
Τεχνικό εργαστηριακό προσωπικό, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Ο Καραγιαννόπουλος Αριστείδης δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

Πίνακας Περιεχομένων

Περίληψη

Στην εποχή μας, με την εκτεταμένη και υποχρεωτική χρήση του AutomaticIdentificationSystem (AIS) για πολλά σκάφη, δημιουργείται μια μεγάλη ποσότητα δεδομένων για τις τροχιές των πλοίων. Αυτά τα δεδομένα παρέχουν πληροφορίες για γεωγραφικό μήκος και πλάτος, ταχύτητα, πορεία, καθώς και πληθώρα άλλων πληροφοριών μέσω των οποίων γίνεται ευκολότερη η εξαγωγή θαλάσσιων μοτίβων και η πρόβλεψη συμπεριφοράς πλοίων. Με βάση τα παραπάνω, είναι δυνατή η δημιουργία ενός dataset που απεικονίζει τις πιο συνήθεις διαδρομές πλοίων. Ο κύριος σκοπός αυτής της μελέτης είναι αναλύοντας ένα dataset από πολύγωνα που έχουν παραχθεί μέσω των δεδομένων του AISνα συγκρίνει διάφορες προσεγγίσεις για την εύρεση της βέλτιστης διαδρομής μεταξύ δύο λιμανιών. Αυτό επιτυγχάνεται με τη δημιουργία μιας εφαρμογής που χρησιμοποιεί τον αλγόριθμο Dijkstra για την εύρεση της συντομότερης διαδρομής (shortest path). Η εν λόγω εφαρμογή δέχεται ως ορίσματα εισόδου τα επεξεργασμένα δεδομένα από τα πολύγωνα, βρίσκει και αναπαριστά την πιο σύντομη διαδρομή μεταξύ των δύο λιμανιών. Επομένως, δημιουργώντας διάφορα σενάρια με διαφορετικά input datasets μπορούμε να κάνουμε τις απαραίτητες συγκρίσεις, ώστε να επιλέξουμε το βέλτιστο σενάριο.

Λέξεις κλειδιά: Βέλτιστη διαδρομή, αλγόριθμος Dijkstra, πολύγωνα, σενάριο

Abstract

In modern times, with the extensive and mandatory use of the AutomaticIdentificationSystem (AIS) for many vessels, a large amount of data is created concerning ship orbits. This data provides information on latitude and longitude, speed, course, as well as a variety of other information that makes it easier to export marine motifs and predict ship behavior. Based on the above, it is possible to create a dataset that depicts the most common ship routes. The main purpose of this study is to analyze a dataset of polygons generated through AIS data to compare different approaches to finding the optimal path between two ports. This is achieved by creating an application that uses the Dijkstra algorithm to find the shortest path. This application finds and represents the shortest route between the two ports, by having as input values the processed data from the polygons. Therefore, by creating different scenarios with various input datasets we can make the necessary comparisons in order to select the optimal scenario.

Keywords: Optimal route, Dijkstra algorithm, polygon, scenario

Κατάλογος Εικόνων

Κατάλογος Πινάκων

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

SOLAS	International Convention for the Safety of Life at Sea
AIS	Automatic Identification System
OSPF	Open Shortest Path First
IMO	International Maritime Organization
IHO	International Hydrographic Organization

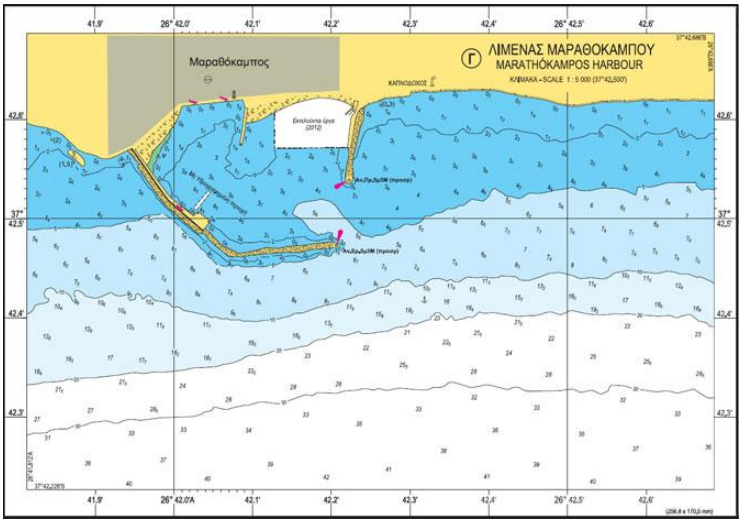
Εισαγωγή

Με την ταχεία ανάπτυξη των θαλάσσιων μεταφορών, η πολυπλοκότητα της πλοήγησης αυξάνεται διαρκώς. Προκειμένου να διασφαλιστεί η βέλτιστη ποιότητα ως προς την ασφάλεια και την ταχύτητα καθώς και η μικρότερη δυνατή χρήση καύσιμων, έχουν αναπτυχθεί διάφοροι τρόποι προσέγγισης επίλυσης αυτού του προβλήματος.

Η επιλογή της βέλτιστης διαδρομής πρέπει να γίνεται με γνώμονα την ασφάλεια, την αποτελεσματικότητα, την ταχύτητα, το κόστος και την ενεργειακή απόδοση. Ως εκ τούτου, η διαδικασία εύρεσης της βέλτιστης διαδρομής αποτελεί ένα περίπλοκο πρόβλημα, καθώς πρέπει να ληφθούν υπόψη πολλές παράμετροι, όπως επίσης και κάποιες απρόβλεπτες, όπως για παράδειγμα οι καιρικές συνθήκες. Για να καταστεί δυνατή η εύρεση της βέλτιστης διαδρομής, πρέπει να χρησιμοποιηθεί μία βάση δεδομένων με σημεία (συντεταγμένες) στον χάρτη και έπειτα να επιλεχθεί ένας αλγόριθμος που θα επιλέγει την πορεία χρησιμοποιώντας αυτή τη βάση. Σημειώνεται πως αυτά τα σημεία θα πρέπει να βρίσκονται στη θάλασσα και να επιτρέπεται να περάσει πλοίο από αυτά, δηλαδή να μην υπάρχει κάτι που να αποτρέπει την πλεύση.

Για την εύρεση αυτών των σημείων τα παλαιότερα χρόνια, αρχικά οι πλοηγητές έκαναν χρήση των αστεριών και του ήλιου. Με βάση τη θέση είχαν μια σαφέστερα εικόνα για το πού βρίσκονταν και πως τα που ήταν ο προορισμός τους. Βέβαια η γνώση της κατεύθυνσης που έπρεπε να πάρουν δεν ήταν απόλυτη καθώς εξαρτιόνταν και από τα καιρικά φαινόμενα. Με τα χρόνια η πλοήγηση έγινε ευκολότερη με τη χρήση ναυτικών χαρτών, οι οποίοι ήταν σχεδιασμένοι σε χαρτί και αποτελούσαν την επίσημη βάση δεδομένων των κυβερνητικών εξουσιοδοτημένων υδρογραφικών υπηρεσιών. Αυτοί οι χάρτες παρείχαν μια δισδιάστατη όψη των χερσαίων και υδάτινων περιοχών, καθώς και των κινδύνων πλοήγησης (π.χ. ύφαλοι, ναυάγια). Οι έντυποι αυτοί χάρτες πρέπει να ακολουθούν τις προβλεπόμενες διατάξεις της Ασφάλειας της Ζωής στη Θάλασσα (SOLAS[1]) του IMO[2]καθώς επίσης και το πρότυπο του IHO, S-4.

[1]http://www.imo.org/en/About/conventions/listofconventions/pages/international-convention-for-the-safety-of-life-at-sea-(solas),-1974.aspx
[2]http://www.imo.org/en/Pages/Default.aspx



Εικόνα 1 - Έντυπος Ναυτικός Χάρτης

Η δυσκολία ενημέρωσης αυτών των κλασικών έντυπων χαρτών, οδήγησε στη διαδοχή τους από τους ηλεκτρονικούς. Οι ηλεκτρονικοί κατασκευάζονται σύμφωνα με το πρότυπο S-57 Ed 3.1[3] του Διεθνούς Υδρογραφικού Οργανισμού - IHO.

Έχοντας στη διάθεση μας τον χάρτη με τα σημεία, καλούμαστε να επιλέξουμε τον κατάλληλο αλγόριθμο εύρεσης συντομότερης διαδρομής. Στην πάροδο των χρόνων έχουν αναπτυχθεί πολλοί τέτοιοι αλγόριθμοι, όπως για παράδειγμα[4]:

- Bellman–Ford
- Dijkstra
- A* search
- Floyd–Warshall
- Johnson's
- Viterbi

Συγκεκριμένα, στην παρούσα εργασία χρησιμοποιήθηκε ο αλγόριθμος Dijkstra και η λειτουργικότητά του αναλύεται σε επόμενο κεφάλαιο (βλ. Κεφάλαιο 3).

Γενικές πληροφορίες:

- Όλες οι αποστάσεις που αναφέρονται στην παρούσα εργασία έχουν υπολογισθεί σε ναυτικά μίλια (**n**autical **m**iles). Ένα ναυτικό μίλι ισούται με 1852 μέτρα ή 6.076

[3] http://www.s-57.com/
[4] https://en.wikipedia.org/wiki/Shortest_path_problem

πόδια.[5]

- Οι γεωγραφικές συντεταγμένες των λιμανιών αναχώρησης και άφιξης, αφορούν ένα κεντρικό σημείο μέσα στο λιμάνι.
- Η διαχείριση της βάσης έγινε με Mongodb - node.js
- Ο κώδικας έχει γραφεί σε html, javascript και css
- Το πρόγραμμα επεξεργασίας κώδικα που χρησιμοποιήθηκε ήταν το Visual Studio Code

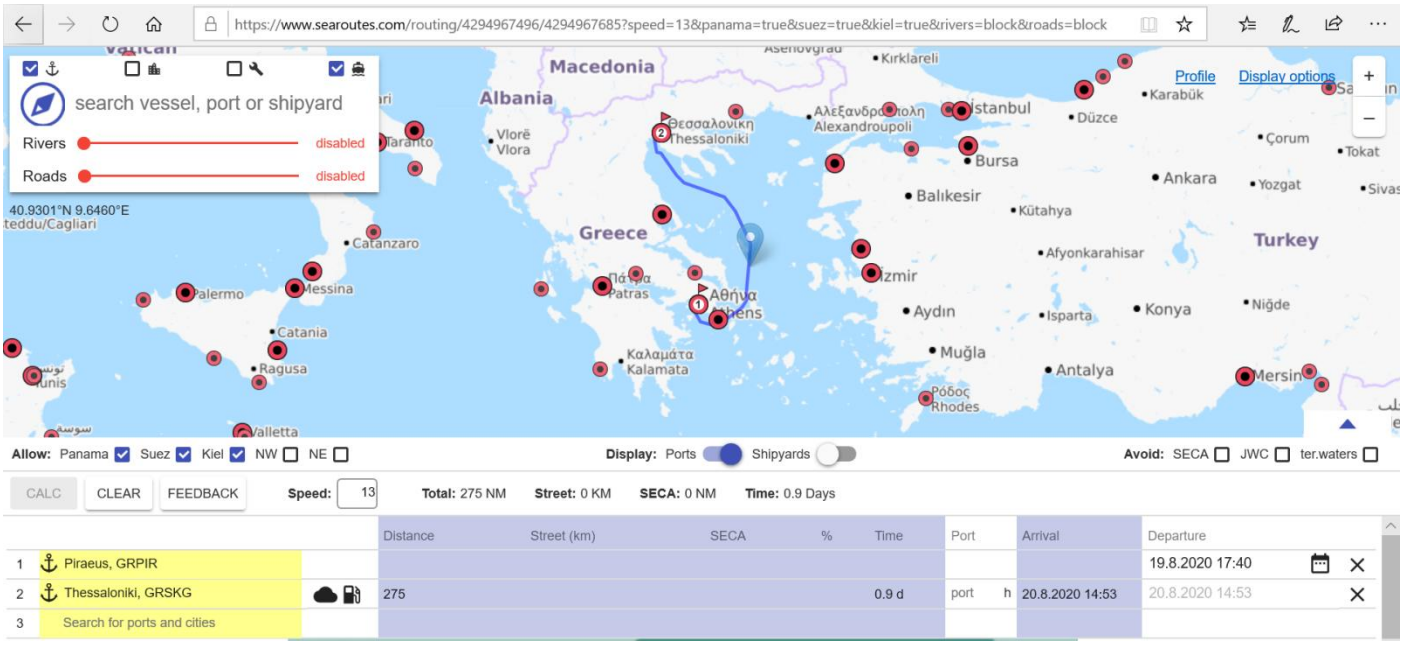
[5]https://el.wikipedia.org/wiki/Μίλι

Κεφάλαιο: 1. Σχετικές Εργασίες

Η επιτακτικότητα επίλυσης του προβλήματος εύρεσης βέλτιστης πορείας πλωτών, έχει οδηγήσει στην ανάπτυξη διάφορων αντίστοιχων εφαρμογών, όπως για παράδειγμα:

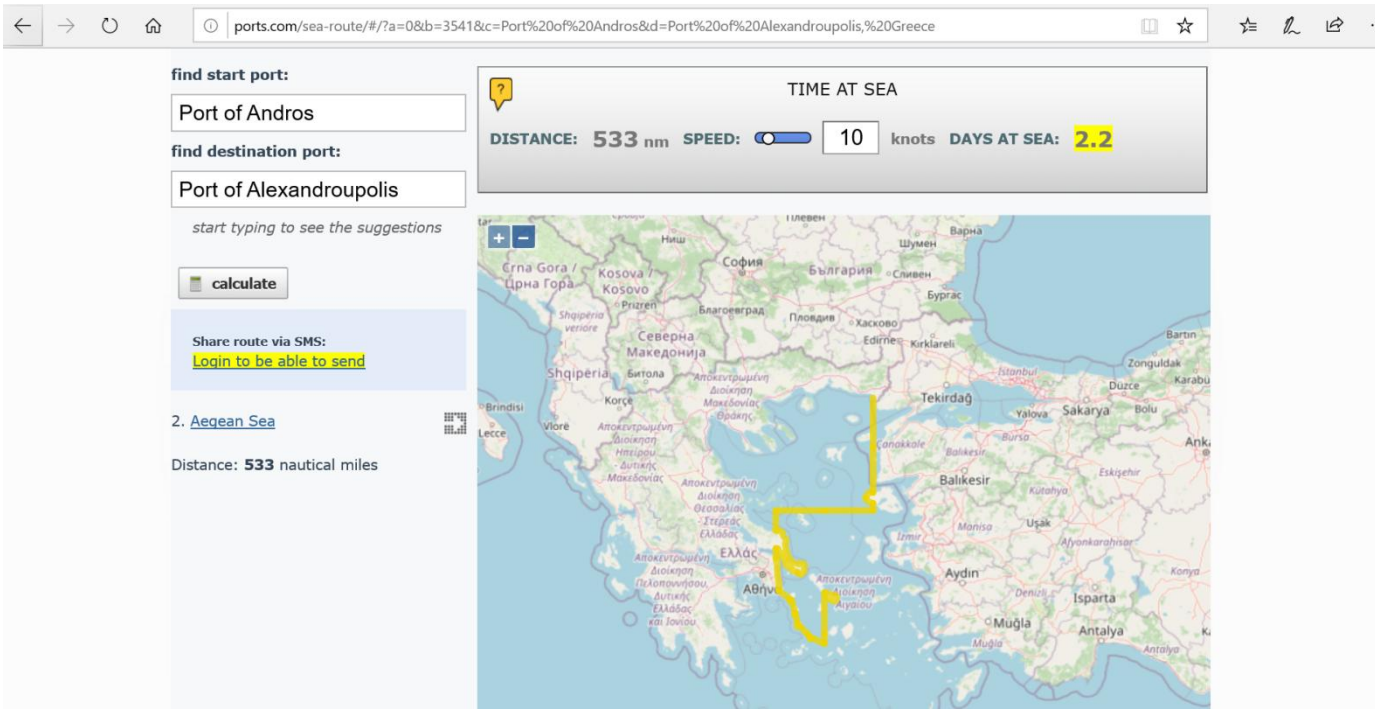
1. https://www.searoutes.com

Η ιστοσελίδα searoutes έχει δημιουργηθεί από την εταιρία Maritime Data Systems. Η φισολοφία της είναι αρκετά κοντά σε αυτή της εφαρμογής μας, καθώς χρησιμοποιεί δεδομένα του AIS για διαδρομές που έχουν ήδη γίνει, και με βάση αυτές επιλέγει την προτιμότερη. Προσφέρει πληθώρα επιλογών, όπως επιλογή ταχύτητας, χρήση ποταμών, επιλογή διωρύγων, αποφυγή αιγιαλίτιδας ζώνης, επιλογή ενδιάμεσων προορισμών, κ.α. Θεωρείται από τις πιο ολοκληρωμένες δωρεάν ιστοσελίδες για εύρεση διαδρομής μεταξύ λιμανιών.



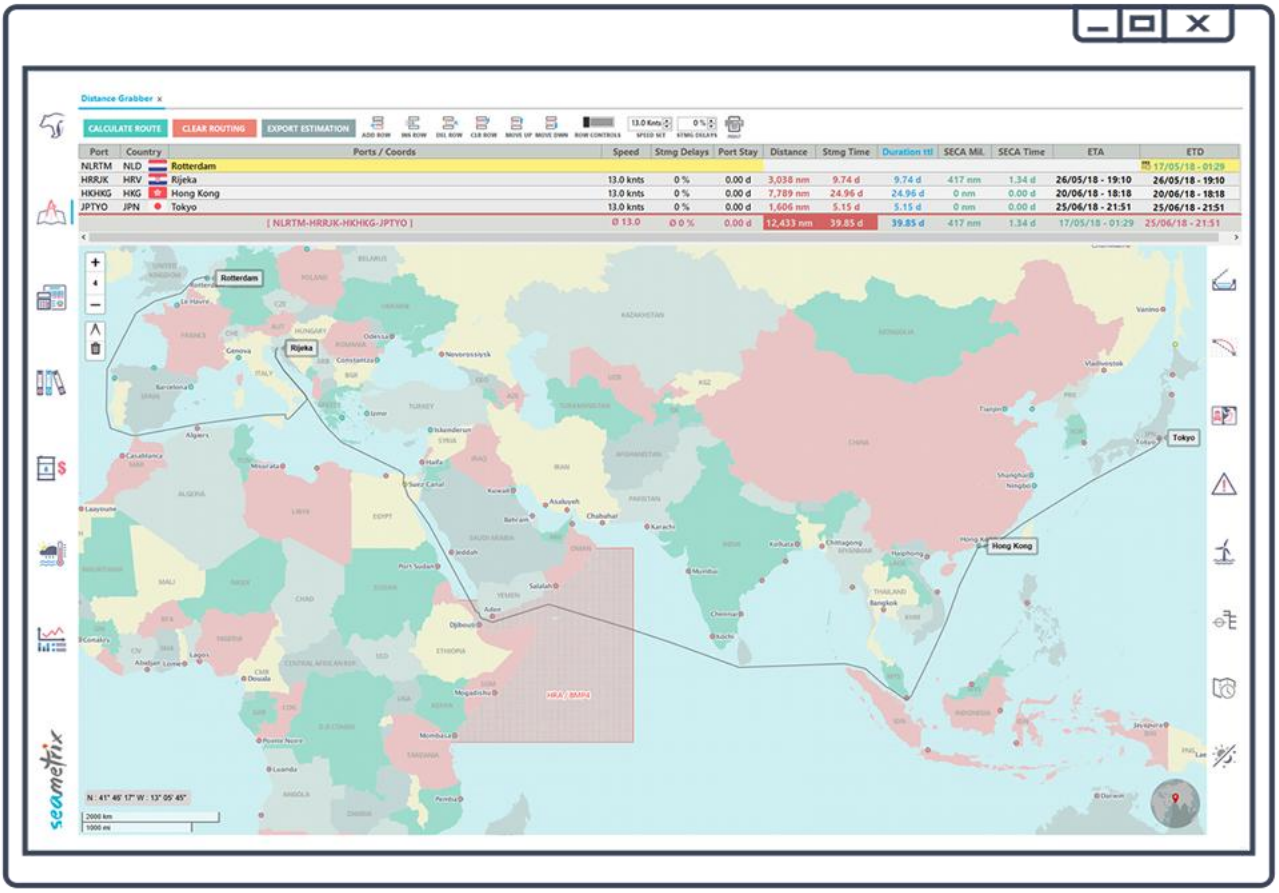
2. http://ports.com/sea-route/

Η ιστοσελίδα ports.com χρησιμοποιεί τις ακτογραμμές για την εύρεση διαδρομής μεταξύ δύο λιμανιών. Ως αποτέλεσμα, κανένα από τα αποτελέσματά της δεν είναι βέλτιστο, καθώς ακολουθάει μόνο ακτογραμμές, και περνάει με ευθείες γραμμές μέσα από κομμάτια ξηράς νησιών, για να κάνει μετάβαση από μια ακτογραμμή σε μια άλλη.



3. https://seametrix.net/

Η ιστοσελίδα seametrix.net παρέχει μια εφαρμογή, η οποία, σύμφωνα με τους δημιουργούς της, παρέχει πληθώρα υπηρεσιών. Σε αυτές περιλαμβάνονται η εύρεση της ταχύτερης διαδρομής, η πλήρως προσαρμόσιμη επιλογή σημείων, οι πληροφορίες για τη διαδρομή μέσω ενός εκτενούς API πληροφοριών, η πρόβλεψη καιρού για τροποποιήσεις στη διαδρομή και άλλα.



Οι δύο πρώτες διατίθενται δωρεάν, ενώ η τρίτη επί πληρωμή.

Υπάρχουν επίσης και αντίστοιχες εφαρμογές για κινητά:

➢ C-Map Embark

(https://www.c-map.com/app/)
Η εφαρμογή C-Map επικεντρώνεται κυρίως σε διαδρομές με μικρές αποστάσεις μεταξύ τους, χωρίς να εμφανίζει αποτελέσματα σε μεγαλύτερες.

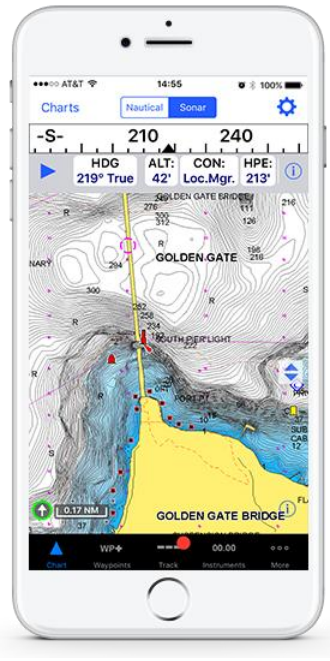
Αποφεύγει τμήματα ξηράς, χωρίς να συνηθίζει να επιλέγει πλεύση σε ανοιχτή θάλασσα για χρήση μεγαλύτερης ταχύτητας από τα πλοία. Για αυτό το λόγο, η συγκεκριμένη εφαρμογή ενδείκνυται περισσότερο για χρήση από μικρότερα σκάφη.

➢ iNavX

(https://inavx.com/)

Η εφαρμογή iNavX είναι από τις πλέον γνωστές εφαρμογές πλοήγησης για κινητά και tablet.

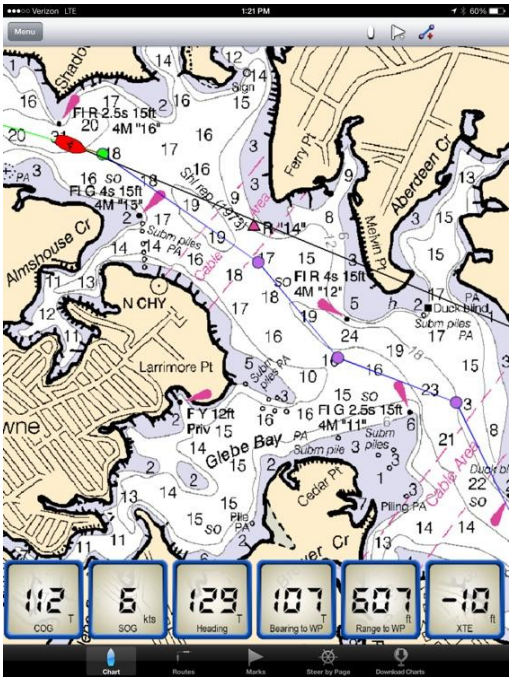
Παρέχει real-time ενημέρωση θέσης, ταχύτητας και πορείας, ενημέρωση καιρού, προειδοποίηση αγκυροβόλησης, δυνατότητα σύνδεσης με όργανα πλοίου, καθώς και καταγραφή ταξιδιών για μετέπειτα προβολή.



- MaptechiPlot

(http://www.richardsonscharts.com/)

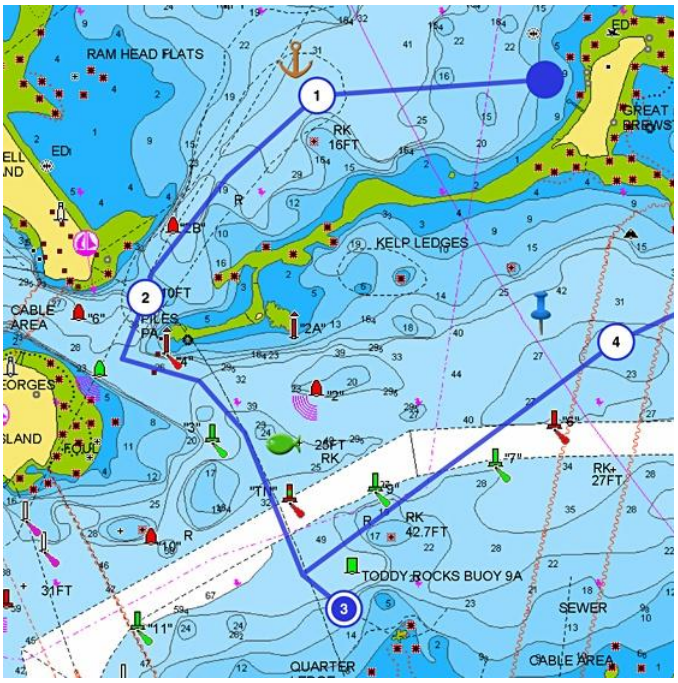
Η εφαρμογή MaptechiPlot διατίθεται για IOS και προσφέρει υπηρεσίες όπως, καταγραφή διαδρομών, προειδοποίηση αγκυροβόλησης, real-time δεδομένα πορείας και σκάφους και άλλα. Είναι πολύ κοντά σε λειτουργικότητα με την εφαρμογή iNavX.



- Navionics

(https://www.navionics.com/usa/apps/navionics-boating)

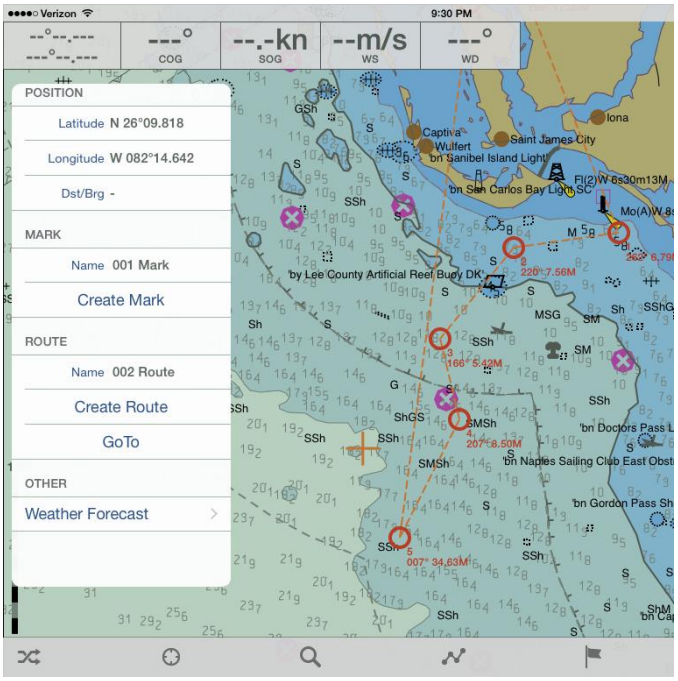
Η εφαρμογή navionics αναφέρεται κυρίως σε μικρότερα σκάφη. Προσφέρει μεγάλη ποικιλία χαρτών, συμπεριλαμβανομένων και 3D χαρτών, με ενημέρωση για βάθος θάλασσας, χάρτη sonar και άλλα.



➢ SeaPilot

(https://www.seapilot.com/)

Η εφαρμογή SeaPilot είναι μια αντίστοιχη εφαρμογή της Navionics, που επικεντρώνεται σε διαδρομές για μικρότερα σκάφη.



Η πρώτη διατίθεται δωρεάν, ενώ οι υπόλοιπες επί πληρωμή, με την εφαρμογή SeaPilot να προσφέρει και δωρεάν version. Οι εφαρμογές για κινητά προσφέρουν ανίχνευση σήματος GPS και με αυτό τον τρόπο οι περισσότερες προσφέρουν και προβολή δεδομένων AIS, για τα υπόλοιπα πλοία, καθώς και για το ίδιο.

Όπως παρατηρείται, οι εφαρμογές αυτές δεν ακολουθούν πάντα τη βέλτιστη διαδρομή, και σε αρκετές περιπτώσεις εμφανίζονται σφάλματα στις διαδρομές, ειδικότερα στις εφαρμογές που διατίθενται δωρεάν.

Για την καλύτερη σύγκριση μεταξύ των εφαρμογών, παρατίθεται ο παρακάτω πίνακας:

Εφαρμογή	Αποφυγή Ξηράς	Εύρεση διαδρομής με βάση τη ταχύτητα	Αποφυγή ακτογραμμών	Μεγάλες αποστάσεις	Επιλογή ενδιάμεσων σημείων	Δεδομένα διαδρομής με βάση το καιρό	Αποθήκευση παλαιότερων διαδρομών	3D χάρτες
VCOS (Vessel Course Optimization System)		✓	✓*	✓				
Searoutes.com	✓	✓	✓	✓	✓			
Ports.com		✓		✓				
Seametrix	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	
C-map	✓	✓	✓					
iNavX	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	
MaptechiPlot	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	✓(?)	
Navionics	✓(?)		✓(?)	✓(?)				✓(?)
SeaPilot	✓		✓	✓		✓	✓	✓

Πίνακας 1 - Διαφορές μεταξύ εφαρμογών

- *Χρησιμοποιείται το σύμβολο (?) στις περιπτώσεις τις οποίες οι εφαρμογές είναι επί πληρωμή και δεν μπορεί να γίνει έλεγχος των υπηρεσιών που προσφέρουν.*
- *Στην περίπτωση του συμβόλου * η υπηρεσία προσφέρεται με προϋποθέσεις.*

Κεφάλαιο 2. Περιγραφή Προβλήματος, Σύνολα Δεδομένων (Datasets) και Workflow

Όπως προαναφέρθηκε, η εύρεση της ταχύτερης δυνατής διαδρομής μεταξύ δύο σημείων στη θάλασσα αποτελεί πρόβλημα εδώ και χρόνια. Οι περισσότερες ναυτιλιακές εταιρείες προσπαθούν συνέχεια να αναβαθμίσουν τη δρομολόγηση των πλοίων τους, ώστε να μπορούν να έχουν μεγαλύτερο κέρδος μειώνοντας το χρόνο που απαιτείται για την άφιξη δρομολόγιων από το ένα λιμάνι στο άλλο. Μια γρήγορη διαδρομή μεταξύ λιμένων είναι επίσης σημαντική για τα επιβατηγά πλοία, που προσπαθούν να ολοκληρώσουν το δρομολόγιό τους όσο το δυνατόν γρηγορότερα, ώστε να μπορούν να εξυπηρετούν πολύ περισσότερους επιβάτες, ειδικά κατά τη διάρκεια του καλοκαιριού σε χώρες που εξαρτώνται σε μεγάλο βαθμό από τον τουρισμό.

Λόγω του απρόβλεπτου χαρακτήρα της θάλασσας (θαλάσσια ρεύματα, σημεία με δυσμενείς καιρικές συνθήκες, ύφαλοι), είναι πολύ δύσκολο να χαρτογραφηθεί μια διαδρομή στη θάλασσα χωρίς δεδομένα. Σε αυτό το σημείο τα δεδομένα από το AIS προσφέρουν σημαντική βοήθεια καθώς, με την επεξεργασία τους, είναι δυνατό να βρεθούν διαδρομές που χρησιμοποιούν τα πλοία συχνότερα, ώστε να φτάσουν στον προορισμό τους. Τέτοιες πληροφορίες κρίνονται σημαντικές, καθώς με αυτό το τρόπο γνωρίζουμε ότι αυτές οι διαδρομές είναι ασφαλείς και αποτελεσματικές. Με αυτές τις πληροφορίες δύναται να δημιουργηθούν πολύγωνα, με κάθε κορυφή του πολυγώνου να περιέχει πολύτιμα δεδομένα σχετικά με: τη μέση ταχύτητα του πλοίου, το γεωγραφικό πλάτος–μήκος, καθώς και άλλα δεδομένα.

Επιπλέον, είναι σημαντικό να επισημανθεί ότι αν και η πιο συνηθισμένη διαδρομή μπορεί να βοηθήσει να αποφευχθούν άλλες διαδρομές υψηλότερης επικινδυνότητας (π.χ. περιοχές με συχνές δυσμενείς καιρικές συνθήκες ή επιρρεπείς σε απειλές πειρατείας), τα δεδομένα AIS δεν μπορούν να χρησιμοποιηθούν για τον υπολογισμό της βέλτιστης διαδρομής με βάση τα καιρικά φαινόμενα ή οποιασδήποτε άλλης απρόβλεπτης κατάστασης.

Ένας άλλος παράγοντας που πρέπει να ληφθεί υπόψη είναι ότι, για να λειτουργεί σωστά ένας αλγόριθμος ταχύτερων διαδρομών όπως για παράδειγμα ο Dijkstra, απαιτείται πολλή σημαντική ποσότητα δεδομένων. Επίσης, δεδομένου ότι ο εν λόγω αλγόριθμος υπολογίζει την καλύτερη επιλογή μεταξύ γειτονικών σημείων, εάν δύο σημεία έχουν

μεγάλη απόσταση μεταξύ τους, είναι πιθανό η γραμμή μεταξύ αυτών των σημείων να περάσει από έναν ύφαλο ή ακόμη και από μια μικρή έκταση ξηράς. Προκειμένου να εξαλειφθεί αυτό το πρόβλημα, θα χρειαζόταν ένα άλλο σύνολο δεδομένων, με πληροφορίες σχετικά με το αν κάθε σημείο μεταξύ της διαδρομής βρίσκεται σε περιοχή που επιτρέπεται να περνάει ένα πλοίο.

Datasets

Για την υλοποίηση της παρούσας εργασίας δόθηκε ένα σύνολο δεδομένων (Polygons) που περιείχε διάφορες πληροφορίες που αφορούν τα σημεία που χρησιμοποιήθηκαν, όπως για παράδειγμα τις συντεταγμένες των πολυγώνων, τις συντεταγμένες αρχής και τέλους των πολυγώνων, τη μέση ταχύτητα, κ.ά.

Επιπλέον, χρησιμοποιήθηκε το PortsDataset για τις συντεταγμένες των λιμανιών, καθώς και τα datasets της βάσης AlgorithmData που περιέχουν όλη την απαραίτητη πληροφορία που χρειάζεται ο αλγόριθμος για να υπολογίσει τη βέλτιστη διαδρομή.

Για την ευκολότερη χρήση και διαχείριση των βάσεων, έγινε η χρήση του MongoDB Atlas. Το MongoDB Atlas είναι μια πλήρως διαχειριζόμενη βάση δεδομένων cloud που αναπτύχθηκε από τους ίδιους ανθρώπους που δημιούργησαν το MongoDB. Επιπλέον, το MongoDB είναι ένα πρόγραμμα βάσης δεδομένων NoSQL, το οποίο χρησιμοποιεί έγγραφα τύπου JSON με προαιρετικά σχήματα.

Χρησιμοποιώντας το MongoDB Atlas είναι δυνατό το ανέβασμα των δεδομένων σε ξεχωριστές βάσεις και η κλήση τους όταν χρειάζεται. Το ανέβασμα της κάθε βάσης γίνεται με μία απλή εντολή στο command prompt γνωρίζοντας τους κωδικούς του αντίστοιχου χρήστη:

```
Mongoimport
 --uri "mongodb+srv://UserName:<password>@cluster0-pnfpp.mongodb.net/VesselCourseOptimization"
 --collection Distances10kmSpd
–jsonArray
--drop
 --file PointDistances10kmSpd.json
```

Uri	Το uri ώστε να γίνει η σύνδεση με τη βάση
Collection	Το όνομα του υποσυνόλου που θα δημιουργηθεί η θα γίνει επεξεργασία της
JsonArray	Η μορφή που θα έχουν τα δεδομένα μας
Drop	Χρησιμοποιείται όταν θέλουμε να διαγράψουμε το παλιό υποσύνολο με το ίδιο όνομα και στη θέση του να βάλουμε νέα δεδομένα
File	Το αρχείο που θα ανέβει στη βάση μας

Πίνακας 2 – MongoDb Εντολές

Με αντίστοιχο τρόπο γίνεται και η κλήση των δεδομένων από την εφαρμογή. Η κλήση γίνεται μέσω Node.Js και χρησιμοποιώντας τα frameworks Mongoose και Express.

Το mongoose framework είναι χτισμένο πάνω στο MongoDb και βοηθάει στην ευκολότερη μοντελοποίηση των δεδομένων. Το Express framework είναι χτισμένο πάνω στο Node.js framework και βοηθάει στο ευκολότερο routing των χρηστών σε διαφορετικά μέρη των εφαρμογών ιστού με βάση το αίτημα που υποβλήθηκε.

Έτσι, με τον παρακάτω τρόπο γίνεται η κλήση των βάσεων από το MongoDb Atlas:

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var mongoose = require('mongoose');

app.use(bodyParser.json());

const uri = "mongodb+srv://Username:<password>@cluster0-
pnfpp.mongodb.net/VesselCourseOptimization?retryWrites=true&w=majority";
mongoose.connect(uri,{ useUnifiedTopology: true });
var client = mongoose.connection;

app.listen(3000);
console.log('Running on port 3000');
```

Για κάθε βάση γίνεται ξεχωριστή κλήση με το αντίστοιχο σχήμα της. Για παράδειγμα:

```
app.get('/api/Distances10Nm', function (req, res) {
    Distances10Nm.getDistances10Nm(function (err, Distances10Nm) {
        if (err) {
            throw err;
        }
        res.json(Distances10Nm);
    });
});
```

```
//PointDistances Schema

var Distances10NmSchema = mongoose.Schema({
    _id: {
        type: String,
    },
    PointName: {
        type: String
    },
    distances: {
        type: Object
    },
},
// Επιλογή βασης από mongoDb Atlas
{ collection: 'Distances10Nm' }
);


var Distances10Nm = module.exports = mongoose.model('Distances10Nm', Distances10NmSchema);

//Get PointDistances
module.exports.getDistances10Nm= function (callback, limit) {
    Distances10Nm.find(callback);
};
```

Polygons

Τα αρχικά δεδομένα που δόθηκαν ήταν ένα σύνολο από πολύγωνα καθώς και πληροφορίες για το κάθε ένα εξ' αυτών, με περιοχές αναφοράς τα πελάγη γύρω από την Ελλάδα (Αιγαίο Πέλαγος, Ιόνιο Πέλαγος, Κρητικό Πέλαγος) και ορισμένα σημεία των υδάτινων περιοχών μεταξύ Ιταλίας και Ελλάδας. Κάθε πολύγωνο συνοδευόταν με πληροφορίες όπως:

i. **Polygon:** Το σύνολο των συντεταγμένων των κορυφών ενός πολυγώνου. Γνωρίζοντας τις συντεταγμένες των κορυφών του πολυγώνου, μπορούμε να βρούμε το κέντρο του και να το χρησιμοποιήσουμε ως σημείο αναφοράς για τον αλγόριθμό Dijkstra.

ii. **Start_Lon:** Το γεωγραφικό μήκος της αρχής του πολυγώνου.

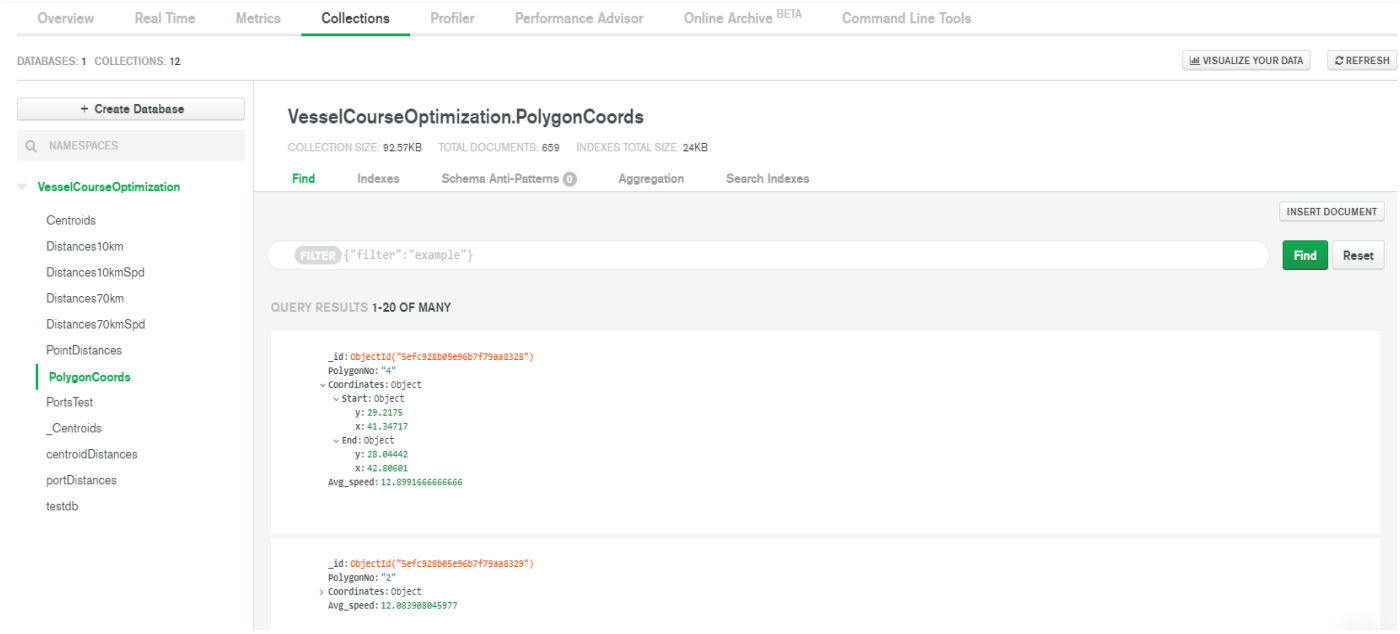
iii. **Start_Lat:** Το γεωγραφικό πλάτος της αρχής του πολυγώνου.

iv. **End_Lon:** Το γεωγραφικό μήκος του τέλους του πολυγώνου.

v. **End_Lat:** Το γεωγραφικό πλάτος του τέλους του πολυγώνου. Γνωρίζοντας τις συντεταγμένες της αρχής και του τέλους του πολυγώνου έχουμε μια δεδομένη μοναδική ευθεία γραμμή κατά μήκος του, η οποία δεν περνά από ξηρά, καθώς τα πολύγωνα περικλείουν υδάτινες περιοχές. Με αυτό το τρόπο, αντί να χρησιμοποιήσουμε το κέντρο του πολυγώνου ως σημείο αναφοράς για τον αλγόριθμο, μπορούμε να χρησιμοποιήσουμε την αρχή και το τέλος του. Επίσης, μπορούμε να υπολογίσουμε την απόσταση από την αρχή έως το τέλος του πολύγωνου.

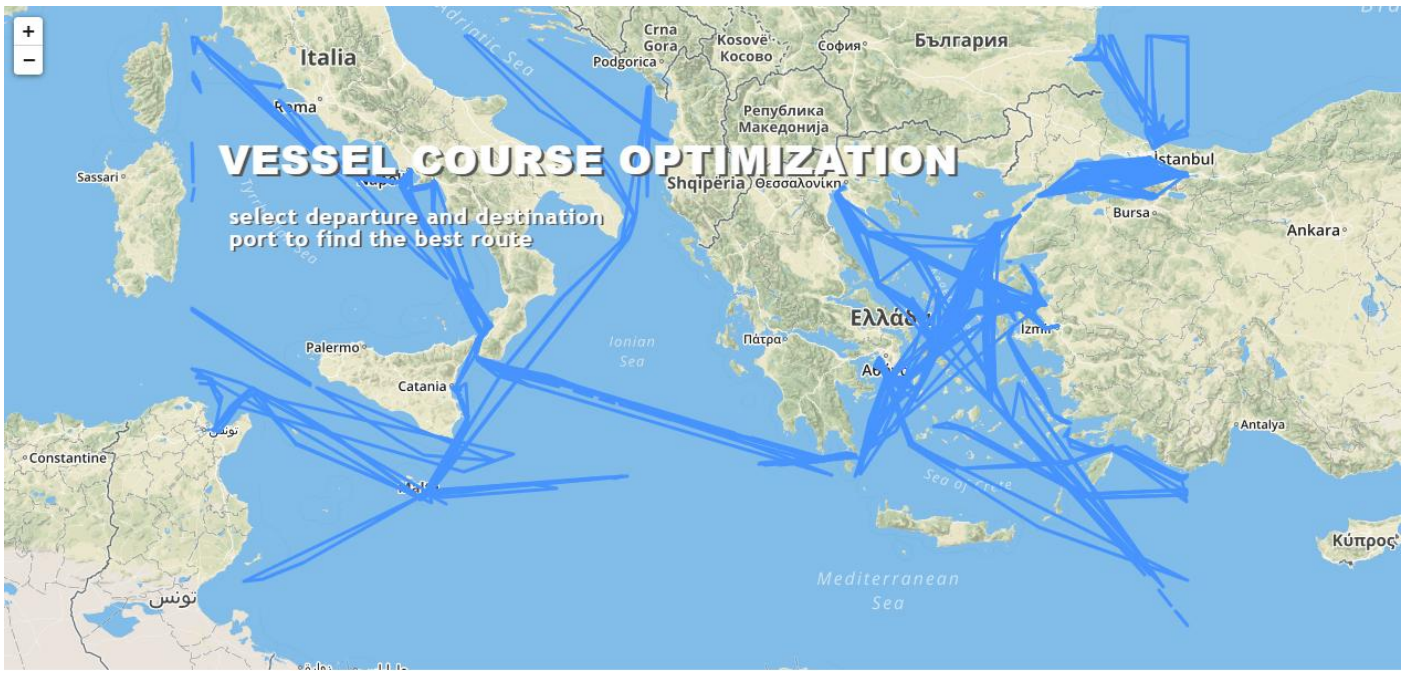
vi. **Avg_speed:** Η μέση ταχύτητα σε κόμβους των πλοίων που έχουν κάνει τη συγκεκριμένη διαδρομή. Γνωρίζοντας τη μέση ταχύτητα, μπορούμε να βρούμε τον χρόνο διάσχισης από την αρχή έως το τέλος του πολύγωνου και να το χρησιμοποιήσουμε ως βάρος στον αλγόριθμο Dijkstra.

Με βάση το παραπάνω dataset, δημιουργήθηκε ένα νέο dataset με αρίθμηση για κάθε πολύγωνο που κρατάει τα παραπάνω στοιχεία χωρίς το σύνολο των κορυφών του πολύγωνου.



Εικόνα 2 - Βάση Δεδομένων στο MongoDBAtlas – Στοιχεία Πολυγώνων

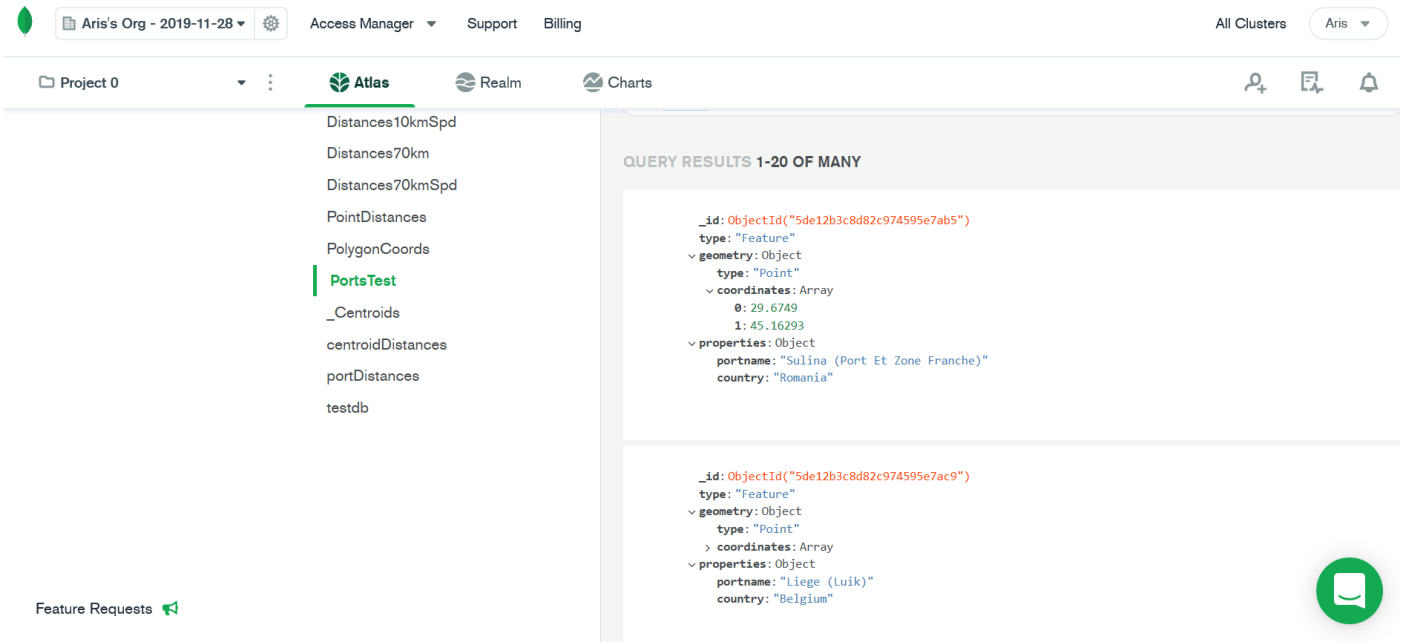
Οι κορυφές των πολυγώνων χρησιμοποιήθηκαν για να γίνει η αναπαράσταση όλων των πολύγωνων του dataset, ώστε να υπάρχει καλύτερη απεικόνιση των δεδομένων και των περιοχών που αντιπροσωπεύουν. Ανέβηκαν σε μία ξεχωριστή βάση που χρησιμοποιήθηκε αποκλειστικά για σκοπούς αναπαράστασης. Με αυτό τον τρόπο τα πολύγωνα που δόθηκαν αναπαριστώνται στο χάρτη ως εξής:



Εικόνα 3 - Αναπαράσταση κορυφών πολυγώνων

Ports Dataset

Η εφαρμογή για την εύρεση της βέλτιστης απόστασης στηρίζεται στην ύπαρξη ενός λιμανιού ως αφετηρία, καθώς και ενός ως προορισμό. Για να είναι αυτό εφικτό χρειαζόμαστε μια βάση με όλα τα λιμάνια, καθώς και τις συντεταγμένες τους , ώστε να υπάρχει πληθώρα επιλογών από τον χρήστη. Η παραπάνω πληροφορία είναι διαθέσιμη στον ιστότοπο https://data.humdata.org/dataset/global-portsσε μορφή excel. Με την κατάλληλη επεξεργασία του αρχείου και κρατώντας τα πεδία που μας χρειάζονται, δημιουργήθηκε ένα υποσύνολο στη βάση μας με τις συντεταγμένες, το όνομα λιμανιού και τη χώρα για κάθε εγγραφή.



Εικόνα 4 - Βάση Δεδομένων στο MongoDBAtlas – Στοιχεία Λιμανιών

Algorithm Data

Για να λειτουργήσει ο αλγόριθμος Dijkstra χρειαζόμαστε μία πληθώρα από σημεία, καθώς και τις αποστάσεις μεταξύ τους. Τα δεδομένα αυτά δεν αρκούν πάντα, καθώς τις περισσότερες φορές χρειαζόμαστε περισσότερες πληροφορίες για να βρεθεί η βέλτιστη διαδρομή, η οποία δε θα βασίζεται μόνο στην απόσταση μεταξύ των σημείων, αλλά και στον χρόνο διάσχισης από σημείο σε σημείο. Για να μπορέσουμε να υπολογίσουμε τον χρόνο διάσχισης χρειαζόμαστε και τη μέση ταχύτητα για κάθε ζεύγος από δύο σημεία.

Ένα ακόμα πρόβλημα που παρουσιάστηκε είναι ο αριθμός των γειτονικών σημείων που θα επιλέγονταν. Αν επιλέγαμε σημεία με απόσταση μεγαλύτερη των εβδομήντα ναυτικών μιλίων μεταξύ τους, ώστε να έχουμε μεγάλο αριθμό σημείων για μεγαλύτερη ακρίβεια, δημιουργείται το πρόβλημα της ύπαρξης στεριάς μεταξύ δύο σημείων, καθώς ο αλγόριθμος μπορεί να επέλεγε δύο σημεία σε μεγάλη απόσταση μεταξύ τους ως βέλτιστα. Από την άλλη πλευρά, αν επιλέγαμε σημεία με αποστάσεις μικρότερες των δέκα ναυτικών μιλίων υπάρχει η πιθανότητα κάποια σημεία να μην είχαν γειτονικά και με αυτό τον τρόπο να μην ολοκληρωνόταν ο υπολογισμός. Για να μπορέσουν λοιπόν να αποφευχθούν αυτά τα προβλήματα, θα έπρεπε να είχαμε έναν πολύ ισχυρό αριθμό δεδομένων, ώστε να υπάρχουν πάντα σημεία το ένα κοντά στο άλλο.

Το αρχικό σενάριο περιείχε τον υπολογισμό του κέντρου κάθε πολύγωνου ως σημείο αναφοράς, και στην συνέχεια των αποστάσεων μεταξύ τους, ώστε να χρησιμοποιηθούν στον αλγόριθμο Dijkstra. Για αυτόν τον λόγο, δημιουργήθηκαν δύο βάσεις δεδομένων, μία με το κέντρο για κάθε πολύγωνο και μία με τις αποστάσεις μεταξύ των κέντρων.

Για τον υπολογισμό των παραπάνω, χρησιμοποιήθηκαν δύο ξεχωριστοί αλγόριθμοι με δεδομένο εισόδου το αρχικό dataset.

Όμως, στο συγκεκριμένο σενάριο παρουσιάστηκε ένα πρόβλημα. Ο αλγόριθμος ψάχνοντας τη βέλτιστη διαδρομή μπορεί να επέλεγε σημεία πολύ μακριά το ένα από το άλλο που ανάμεσα τους είχαν ξηρά. Ακόμα και αν περιορίζαμε τα γειτονικά σημεία σε αυτά που απείχαν μόνο πενήντα ναυτικά μίλια μεταξύ τους, η ακρίβεια του αλγόριθμου δεν ήταν η επιθυμητή τις περισσότερες φορές και για αυτόν τον λόγο το παραπάνω σενάριο εγκαταλείφθηκε.

Το σενάριο στο οποίο βασίστηκε η οριστική υλοποίηση αναφέρεται στη χρήση της απόστασης μεταξύ αρχής και τέλους κάθε πολύγωνου ως μια ευθεία που έχει παραπάνω πιθανότητες να μην εμπεριέχει τμήματα ξηράς και την οποία θα δεχόμαστε πάντα ως δεδομένο, ανεξαρτήτως απόστασης. Καθώς και των αποστάσεων από το τέλος του κάθε πολύγωνου σε όποια αρχή ή τέλος των υπόλοιπων πολύγωνων με συγκεκριμένους περιορισμούς στις αποστάσεις.

Με βάση τα παραπάνω, όπως προαναφέρθηκε, δημιουργήθηκε ένα υποσύνολο με τα στοιχεία από το αρχικό dataset πλην των κορυφών των πολύγωνων.

Στη συνέχεια, τα παραπάνω δεδομένα χρησιμοποιήθηκαν ως είσοδος σε έναν αλγόριθμο υπολογισμού αποστάσεων μεταξύ των σημείων.

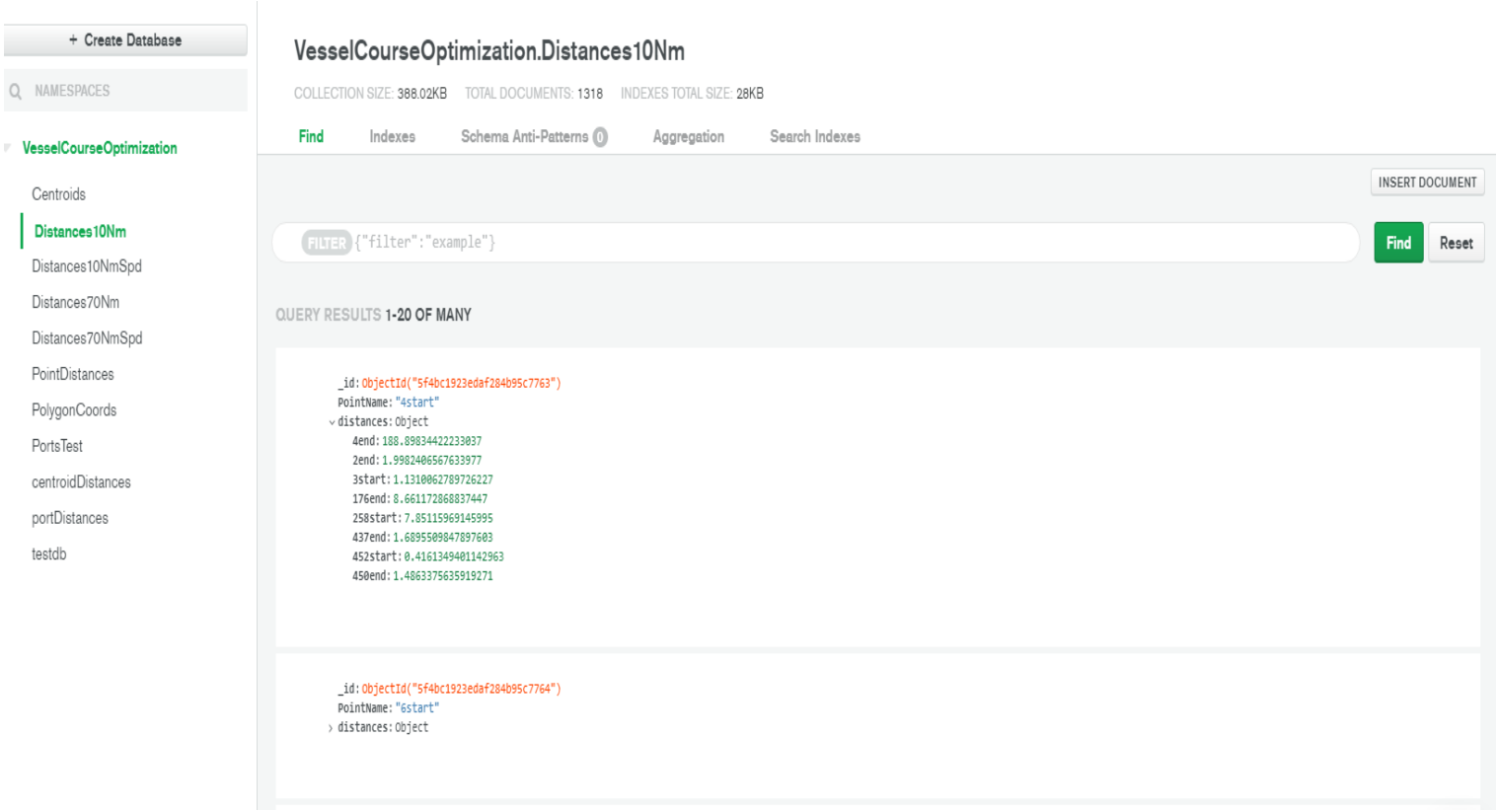
Πιο συγκεκριμένα έγινε υπολογισμός των εξής αποστάσεων:

- Της απόστασης από την αρχή μέχρι το τέλος κάθε πολυγώνου, ώστε να έχουμε μια ευθεία που είναι πιθανότερο να περνάει αποκλειστικά από θάλασσα.
- Των αποστάσεων από το τέλος κάθε πολύγωνου σε όλες τις αρχές των πολύγωνων. Ο υπολογισμός αυτός έγινε με τέσσερις παραδοχές, ώστε να δημιουργηθούν τέσσερα διαφορετικά σενάρια προς σύγκριση μεταξύ τους:

    i. Αποστάσεις μέχρι εβδομήντα ναυτικά μίλια μεταξύ σημείων με βάρος την απόσταση.
    ii. Αποστάσεις μέχρι εβδομήντα ναυτικά μίλια με βάρος τον χρόνο διάσχισης αντί της απόστασης.
    iii. Αποστάσεις μέχρι δέκα ναυτικά μίλια μεταξύ σημείων με βάρος την απόσταση.
    iv. Αποστάσεις μέχρι δέκα ναυτικά μίλια με βάρος τον χρόνο διάσχισης.

Για κάθε μια από τις παραπάνω παραδοχές δημιουργήθηκαν τα αντίστοιχα υποσύνολα στη βάση μας στο MongoDb Atlas. Πιο συγκεκριμένα κάθε εγγραφή στα υποσύνολα αποτελείται από το όνομα του κάθε σημείου (π.χ. 15start, 17endκτλ) καθώς και ένα object με όλες τις αποστάσεις από κάθε άλλο σημείο.

Ένα παράδειγμα υποσύνολου στο περιβάλλον του MongoDb Atlas έχει ως εξής:



Εικόνα 5 - Βάση Δεδομένων στο MongoDBAtlas – Στοιχεία Αποστάσεων για 10 Ναυτικά μίλια

Στη συνέχεια έγινε κλήση του αλγόριθμου Dijkstra για κάθε μία ξεχωριστά.

Κεφάλαιο 3: Μεθοδολογία

Σε αυτό το κεφάλαιο περιγράφεται η μεθοδολογία και ο τρόπος λειτουργίας της εφαρμογής, καθώς και περαιτέρω πληροφορίες για τα εργαλεία και τους αλγόριθμους που χρησιμοποιήθηκαν.

Technical System Information

Η υλοποίηση της εφαρμογής βασίστηκε στις εξής τεχνολογίες:

Το front κομμάτι της υλοποιήθηκε με χρήση html και css, ενώ για τη λειτουργία των αλγόριθμων και την κλήση των βάσεων χρησιμοποιήθηκε node.js σε συνδυασμό με MongoDb.

Πληροφορίες για την εγκατάσταση της MongoDb, καθώς και διάφορων frameworks βρίσκονται μαζί με την εργασία στο github repository: https://github.com/ArisKaragiannopoulos/VesselCourseOptimization και πιο συγκεκριμένα στο αρχείο README.md.

Interface

Για το κομμάτι της απεικόνισης χρησιμοποιήθηκε χάρτης από τη βιβλιοθήκη leaflet, μια open source βιβλιοθήκη χαρτών που προσφέρει διάφορα plugins. Επίσης, περιέχονται δύο textboxes, ένα για το λιμάνι αφετηρίας και ένα για το λιμάνι προορισμού, καθώς και ένα κουμπί(Calculate)με το οποίο γίνεται ο υπολογισμός της βέλτιστης διαδρομής. Με την ολοκλήρωση του υπολογισμού, εμφανίζεται μια γραμμή που συνδέει τα δύο λιμάνια εμφανίζοντας τη βέλτιστη διαδρομή.

Επιλογή λιμανιών

Πληκτρολογώντας το επιθυμητό λιμάνι στο κατάλληλο textbox, εμφανίζονται αυτόματα οι κοντινότερες επιλογές, με βάση τα πρώτα γράμματα του λιμανιού. Αυτό γίνεται με την κλήση τη βάσης των λιμανιών με το κατάλληλο keyword κάθε φορά, χρησιμοποιώντας τις κατάλληλες regular expression επιλογές για το περιβάλλον mongoDb.

Αρχικά,καλείται η function searchStartPort:

```
function searchStartPort() {
                var xhr = new XMLHttpRequest();
                var searchInput = document.getElementById("myInput").value;
                xhr.open("GET", 'http://localhost:3000/api/ports/' + searchInput, true);
                //Δεν χρειάζεται xhr.readyState check γιατί το  xhr.onload λειτουργεί μόνο αν το  xhr.rea
dyState == 4: request finished and response is ready
                xhr.send();
                xhr.onload = function () {
                    if (this.status == 200) {
                        var ports = JSON.parse(this.responseText);
                        var coordinates = [];
```

```
                        for (var i = 1; i < ports.length; i++) {
                            document.getElementById("StartportValue" + i).value = ports[i].properties.cou
ntry + ',' + ports[i].properties.portname;

                        }
                    }
                };
            }
```

Η οποία με τη σειρά της κάνει ένα Get στη βάση με τις πληροφορίες για τα λιμάνια:

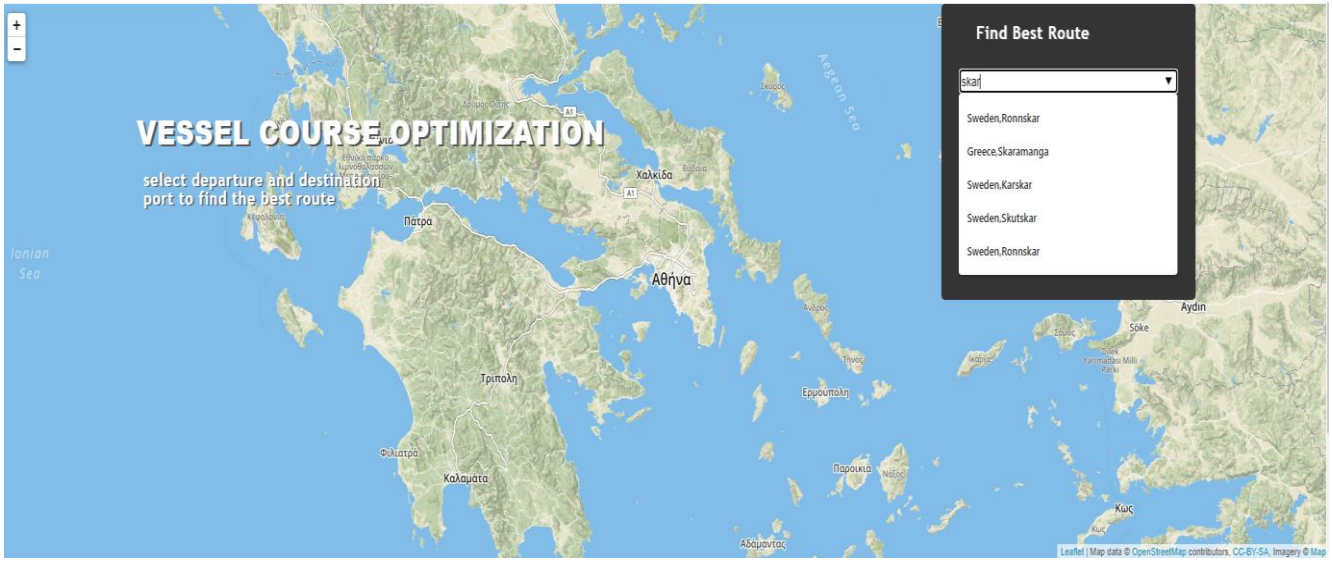
```
app.get('/api/ports/:port', function(req, res){
    Ports.getPortByName(req.params.port, function(err, port){
        if(err){
            throw err;
        }
        res.json(port);
    });
});
```

Για κάθε get, post υπάρχει το αντίστοιχο schema που καθορίζει τον τρόπο που είναι δομημένη πληροφορία που ζητάμε, στέλνουμε:

```
var mongoose = require('mongoose');

//Port Schema

var portSchema = mongoose.Schema({
    _id:{
        type: String,
  },
    geometry:{
        coordinates:{
      type: Array
    },
    },
    properties:{
        portname:{
      type: String
    },
    country:{
      type: String
    },
    },
},
{ collection : 'PortsTest'}
);


var Ports = module.exports = mongoose.model('Ports', portSchema);

//Get Ports
module.exports.getPorts = function(callback, limit){
    Ports.find(callback);
};

//Get PortByName
module.exports.getPortByName = function(port, callback){
```

```
    Ports.find({ "properties.portname":{$regex: port, $options: 'i'}}, callback).limit(10);
};
```

$regex : Θα βρει αυτόματα όλες τις επιλογές που ξεκινάνε από τα γράμματα που έχουμε πληκτρολογήσει
$options : ‘i’ : Δε λαμβάνει υπόψη κεφαλαία και πεζά στην εύρεση λιμανιού:



Εικόνα 6 - Επιλογή λιμανιού με αυτόματη εύρεση λιμανιών

Ακολουθεί ένα παράδειγμα επιλογής λιμανιού με αυτόματη εύρεση λιμανιών που αρχίζουν από «skar»:

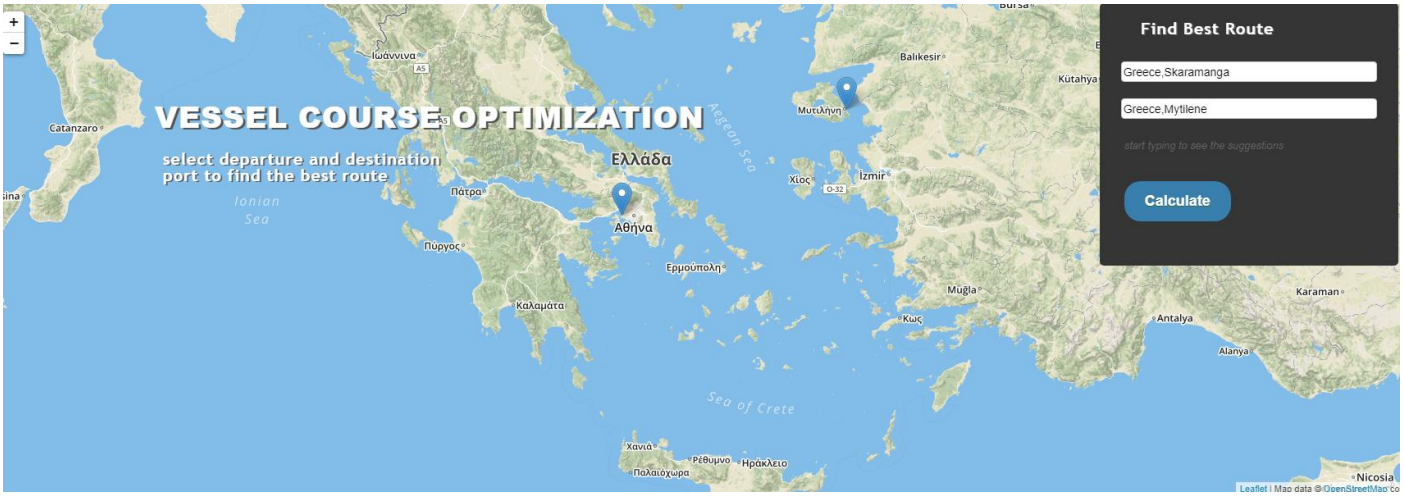
Επιλέγοντας το λιμάνι, τοποθετείται μια πινέζα στον χάρτη στο σημείο όπου βρίσκεται. Για να γίνει αυτό, καλείται η αντίστοιχη functionη οποία καθαρίζει τις προηγούμενες πινέζες, βρίσκει τις συντεταγμένες του λιμανιού και τοποθετεί μία πινέζα σε αυτό. Επίσης, βρίσκει και το κοντινότερο σημείο στο λιμάνι από τη βάση των πολύγωνων, ώστε να μπορέσει να χρησιμοποιηθεί ως αφετηρία για τον αλγόριθμο Dijkstra.

Με την επιλογή του λιμένα προορισμού, τοποθετείται μια δεύτερη πινέζα στις αντίστοιχες συντεταγμένες και πραγματοποιείται εκ νέου κλήση της βάσης των λιμανιών με ένα συγκεκριμένο όνομα κάθε φορά, ώστε να ληφθούν οι συντεταγμένες του. Στη συνέχεια, με βάση αυτές τις συντεταγμένες, τοποθετείται η πινέζα στο αντίστοιχο σημείο. Για παράδειγμα κατά τη τοποθέτηση πινέζας στο λιμάνι αφετηρίας έχουμε :

```
function putStartPin() {
                mymap.removeLayer(startMarker);
                var value;
                newdist = 100000;
                value = document.getElementById("myInput").value;
                startPort = value.split(',')[1];  /* Επιλογή μόνο του λιμανιού από το ζευγάρι χώρα, λιμάνι */
                /* Κλήση στη βάση με πληροφορίες για τα λιμάνια */
                var xhr = new XMLHttpRequest();
                xhr.open("GET", 'http://localhost:3000/api/ports/' + startPort, true);
                xhr.send();
                xhr.onload = function () {
                    if (this.status == 200) {
```

```
                    var ports = JSON.parse(this.responseText);
                    /* Κλήση στη βάση με πληροφορίες για τα πολύγωνα */
                    var xhr2 = new XMLHttpRequest();
                    xhr2.open("GET", 'http://localhost:3000/api/polygons/', true);
                    xhr2.send();
                    xhr2.onload = function () {
                        if (this.status == 200) {
                            var points = JSON.parse(this.responseText)
                            if (ports.length > 0) {
                                Scoordinates = ports[0].geometry.coordinates;
                                Sreal_coord = [Scoordinates[1], Scoordinates[0]];
                                if (startMarker != 'undefined') {
                                    mymap.removeLayer(startMarker);
                                }
                                /* Εύρεση κοντινότερου σημείου αρχής η τέλους πολύγωνων από το λιμάνι */
                                for (i = 0; i < points.length; i++) {
                                    var distance = findDistance(Scoordinates[1], Scoordinates[0], points[i].
Coordinates.End.x, points[i].Coordinates.End.y, "N")
                                    if (distance < 162 && distance < newdist) {
                                        newdist = distance;
                                        closestPoint = [points[i].Coordinates.End.x, points[i].Coordinates.E
nd.y]
                                        closestStartPointName = points[i].PolygonNo + "end";
                                    }
                                }
                                /* Τοποθέτηση πινέζας και εστίαση στο λιμάνι αφετηρίας */
                                connectStart = [(Sreal_coord), (closestPoint)];
                                startMarker = L.marker(Sreal_coord).addTo(mymap);
                            }
                        }
                    };
                }
            };
        }
```

Για το παράδειγμα των λιμένων Σκαραμαγκά και Μυτιλήνης οι πινέζες τοποθετούνται ως εξής:



Εικόνα 7 - Απεικόνιση πινεζών στον χάρτη

Εύρεση Αποστάσεων

Για τη δημιουργία των τεσσάρων datasets που αναφέρονται στις αποστάσεις μεταξύ σημείων, χρειάστηκε η υλοποίηση αλγόριθμου ο οποίος με δεδομένα εισόδου τις συντεταγμένες των σημείων βρίσκει τις αποστάσεις μεταξύ τους. Στην συνέχεια, οι αποστάσεις αποτυπώθηκαν σε ένα json format το οποίο ανέβηκε ως νέο dataset στο MongoDb Atlas. Ο αλγόριθμος που χρησιμοποιήθηκε είναι:

```
function findDistance(lat1, lon1, lat2, lon2, unit) {
                if ((lat1 == lat2) && (lon1 == lon2)) {
                    return 0;
                }
                else {
                    var radlat1 = Math.PI * lat1 / 180;
                    var radlat2 = Math.PI * lat2 / 180;
                    var theta = lon1 - lon2;
                    var radtheta = Math.PI * theta / 180;
                    var dist = Math.sin(radlat1) * Math.sin(radlat2) + Math.cos(radlat1) * Math.cos(radla
t2) * Math.cos(radtheta);
                    if (dist > 1) {
                        dist = 1;
                    }
                    dist = Math.acos(dist);
                    dist = dist * 180 / Math.PI;
                    dist = dist * 60 * 1.1515;
                    if (unit == "K") { dist = dist * 1.609344 }
                    if (unit == "N") { dist = dist * 0.8684 }
                    return dist;
                }
            }
```

Σε συνδυασμό με μία function αποτύπωσης του αποτελέσματος με τις παραμέτρους που θέλαμε.

Για χρήση των αποστάσεων μεταξύ σημείων ως βάρος στον αλγόριθμο, χρησιμοποιήθηκε η εξής function:

```
function findCoordDistance() {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", 'http://localhost:3000/api/polygons/', true);
    xhr.send();
    xhr.onload = function () {
        if (this.status == 200) {
            var allpolygons = JSON.parse(this.responseText);
            for (var i = 0; i < allpolygons.length; i++) {
                var startx = allpolygons[i].Coordinates.Start.x;
                var starty = allpolygons[i].Coordinates.Start.y;
                var endx = allpolygons[i].Coordinates.End.x;
                var endy = allpolygons[i].Coordinates.End.y;
                var distanceArray = [];
                testlength = testlength + 1
                var distance = findDistance(startx, starty, endx, endy, "N")
                for (var j = 0; j < allpolygons.length; j++) {
                var newstartx = allpolygons[j].Coordinates.Start.x;
                var newstarty = allpolygons[j].Coordinates.Start.y;
```

```
                var newendx = allpolygons[j].Coordinates.End.x;
                var newendy = allpolygons[j].Coordinates.End.y;
                var distanceStart = findDistance(endx, endy, newstartx, newstarty, "N")
                //var distanceEnd = findDistance(startx, starty, newendx, newendy, "N")
                /* Αποτελέσματα για αποστάσεις μικρότερες των 10 Ναυτικών μιλίων από το σημείο αρχής */
                /*if (distanceStart <10 && allpolygons[j].PolygonNo != allpolygons[i].PolygonNo ){
                    distanceArray.push("'" + allpolygons[j].PolygonNo + "start'" + ":" + distanceStart +
"");
                } */
                /* Αποτελέσματα για αποστάσεις μικρότερες των 10 ναυτικών μιλίων από το σημείο τέλους */
                if (distanceEnd <10 && allpolygons[j].PolygonNo != allpolygons[i].PolygonNo ){
                    distanceArray.push("'" + allpolygons[j].PolygonNo + "end'" + ":" + distanceEnd + "");
                }
                /* Αποτελέσματα για αποστάσεις μικρότερες των 70 ναυτικών μιλίων από το σημείο αρχής */
                /*if (distanceStart <70 && allpolygons[j].PolygonNo != allpolygons[i].PolygonNo ){
                    distanceArray.push("'" + allpolygons[j].PolygonNo + "end'" + ":" + distanceStart + ""
);
                } */
                /* Αποτελέσματα για αποστάσεις μικρότερες των 70 ναυτικών μιλίων από το σημείο τέλους */
                /*if (distanceEnd <70 && allpolygons[j].PolygonNo != allpolygons[i].PolygonNo ){
                    distanceArray.push("'" + allpolygons[j].PolygonNo + "end'" + ":" + distanceEnd + "");
                } */
                }
                /* Αποτύπωση αποτελεσμάτων για endpoints σε JSON format */
                console.log("{'PointName':'" + allpolygons[i].PolygonNo + "end','distances':{'" + allpoly
gons[i].PolygonNo + "start':" + distance + ","+ distanceArray + "}},")
            }
        }
    }
};
```

Ενώ, για χρήση του χρόνου διάσχισης της διαδρομής ως βάρος χρησιμοποιήθηκε:

```
function findCoordDistanceSpd() {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", 'http://localhost:3000/api/polygons/', true);
    xhr.send();
    xhr.onload = function () {
        if (this.status == 200) {
            var allpolygons = JSON.parse(this.responseText);
            for (var i = 0; i < allpolygons.length; i++) {
                var startx = allpolygons[i].Coordinates.Start.x;
                var starty = allpolygons[i].Coordinates.Start.y;
                var endx = allpolygons[i].Coordinates.End.x;
                var endy = allpolygons[i].Coordinates.End.y;
                var speedi = allpolygons[i].Avg_speed;
                var distanceArray = [];
                testlength = testlength + 1
                var distance = findDistance(startx, starty, endx, endy, "N");
                var time = distance/speedi;
                for (var j = 0; j < allpolygons.length; j++) {
                var newstartx = allpolygons[j].Coordinates.Start.x;
                var newstarty = allpolygons[j].Coordinates.Start.y;
                var newendx = allpolygons[j].Coordinates.End.x;
                var newendy = allpolygons[j].Coordinates.End.y;
                var speedj = allpolygons[j].Avg_speed;
                var avgspeed = (speedi + speedj)/2;
                /* Εύρεση χρόνου διάσχισης για τα σημεία αρχής πολύγωνου */
                var distanceStart = findDistance(endx, endy, newstartx, newstarty, "N")
                var timeStart = distanceStart/avgspeed;
```

```
                var timestartround = (Math.round(timeStart * 100) / 100).toFixed(2);
                /* Εύρεση χρόνου διάσχισης για τα σημεία τέλους πολύγωνου */
                /*var distanceEnd = findDistance(startx, starty, newendx, newendy, "N")
                var timeEnd = distanceEnd/avgspeed;
                var timeendround = (Math.round(timeEnd * 100) / 100).toFixed(2);*/
                /* Αποτελέσματα για αποστάσεις μικρότερες των 10 ναυτικών μιλίων από το σημείο αρχής με β
άση τον χρόνο διάσχισης */
                if (distanceStart < 10 && allpolygons[j].PolygonNo != allpolygons[i].PolygonNo ){
                    distanceArray.push("'" + allpolygons[j].PolygonNo + "start'" + ":" + timestartround +
 "");
                }
                /* Αποτελέσματα για αποστάσεις μικρότερες των 10 ναυτικών μιλίων από το σημείο τέλους με
βάση τον χρόνο διάσχισης */
                /*if (distanceEnd < 70 && allpolygons[j].PolygonNo != allpolygons[i].PolygonNo ){
                    distanceArray.push("'" + allpolygons[j].PolygonNo + "end'" + ":" + timeendround + "")
;
                }*/
                }
                console.log("{'PointName':'" + allpolygons[i].PolygonNo + "end','distances':{'" + allpoly
gons[i].PolygonNo + "start':" + time + ","+ distanceArray + "}},")
            }
        }
    }
};
```

Εύρεση βέλτιστης διαδρομής - Αλγόριθμος Dijkstra

Για τον υπολογισμό της βέλτιστης διαδρομής μεταξύ δύο σημείων υπάρχουν διάφοροι αλγόριθμοι. Στη συγκεκριμένη περίπτωση, χρησιμοποιήθηκε ο αλγόριθμος Dijkstra.

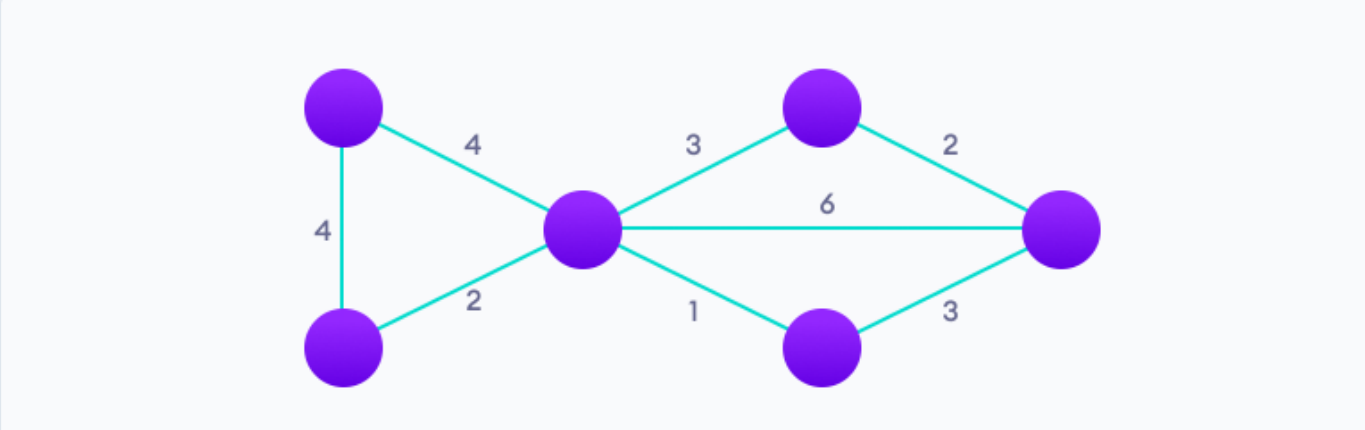
Ο Ολλανδός Edsger W. Dijkstra, από τον οποίο πήρε και το όνομα του ο αλγόριθμος του Dijkstra επινόησε τον αλγόριθμο το 1956 και το 1959 τον δημοσίευσε. Λειτουργεί ως ένας αλγόριθμος εύρεσης συντομότερων διαδρομών (single-source shortest path problem) με δεδομένη κοινή αφετηρία σε έναν γράφο που μπορεί να είναι κατευθυνόμενος η και όχι και ο οποίος δεν έχει αρνητικά βάρη στις ακμές του. Ο αλγόριθμος λειτουργεί βήμα βήμα και για κάθε ένα επιλέγεται η τοπικά βέλτιστη λύση, με αποτέλεσμα στο τελευταίο βήμα να συνθέτεται η συνολικά βέλτιστη λύση[6]. Είναι σημαντικό στο γράφο να μην περιέχονται αρνητικά βάρη, καθώς σε τέτοια περίπτωση ο αλγόριθμος του Dijkstra δεν θα μας δώσει σωστό αποτέλεσμα. Άλλοι αλγόριθμοι, οι οποίοι διαχειρίζονται γράφους με μεγαλύτερη πολυπλοκότητα, όπως ο αλγόριθμος των Bellmankαι Fordή αυτός των Floydκαι Warshall είναι ικανοί να διαχειριστούν περιπτώσεις στις οποίες τα βάρη είναι αρνητικά.

Πολλές εφαρμογές κάνουν χρήση του αλγόριθμου του Dijkstra καθώς είναι ευρέως διαδεδομένος. Ένα παράδειγμα χρήσης του είναι στο πρωτόκολλο OSPF, το οποίο είναι το εσωτερικό πρωτόκολλο πύλης δικτύου του Διαδικτύου.

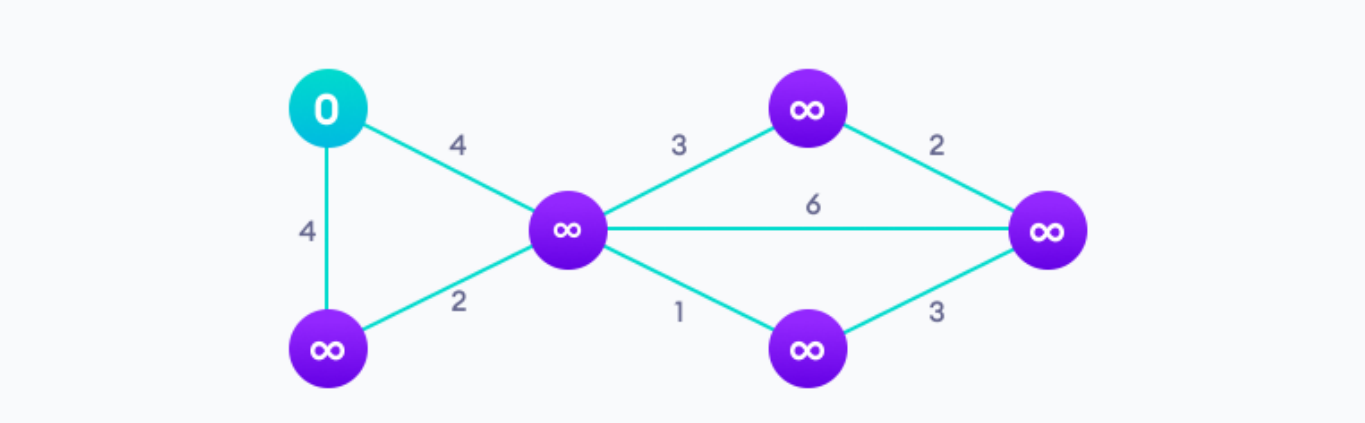
Ο αλγόριθμος του Dijkstra μπορεί να περιγραφεί καλύτερα με τη χρήση ενός παραδείγματος:

[6]https://el.wikipedia.org/wiki/%CE%91%CE%BB%CE%B3%CF%8C%CF%81%CE%B9%CE%B8%CE%BC%CE%BF%CF%82_%CF%84%CE%BF%CF%85_Dijkstra

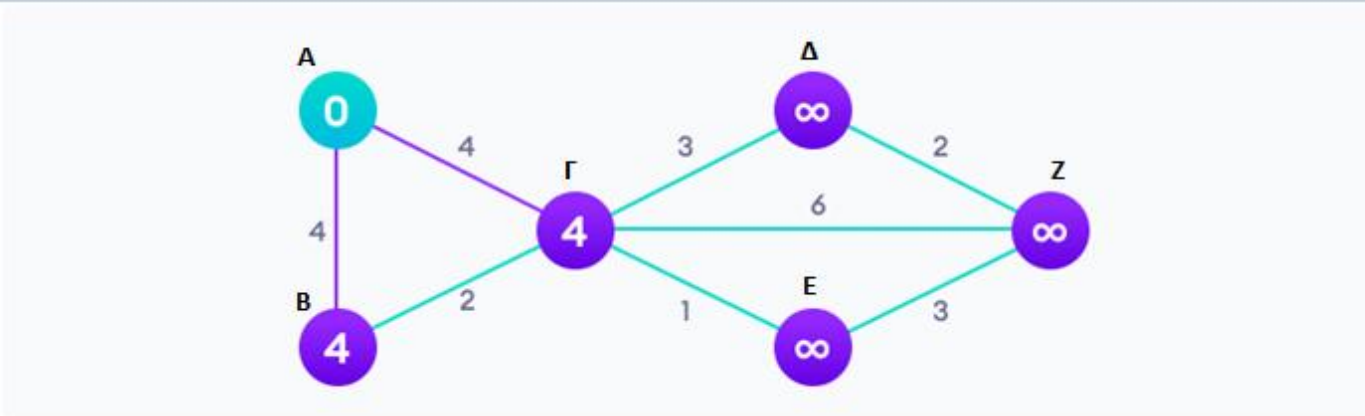
Έστω το παρακάτω γράφημα:



Θεωρούμε αποστάσεις από όλους τους κόμβους με τιμή απόστασης 0 για τον κόμβο εκκίνησης και τιμή απόστασης άπειρο για όλους τους υπόλοιπους.[7]

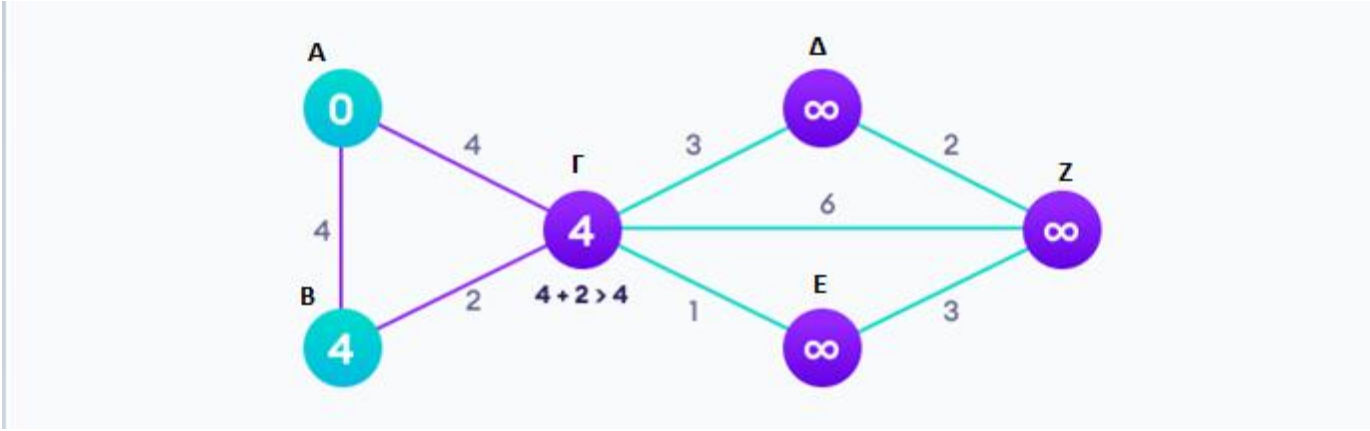


Επισημαίνουμε όλους τους κόμβους ως κόμβους που δεν τους έχουμε επισκεφθεί. Ο κόμβος εκκίνησης είναι και ο τρέχων κόμβος.
Για τον τρέχοντα κόμβο λαμβάνουμε υπόψη όλους τους μη ελεγμένους γείτονες και υπολογίζουμε τις προσωρινές αποστάσεις τους μέσω του τρέχοντος κόμβου. Συγκρίνουμε την πρόσφατα υπολογισμένη προσωρινή απόσταση, με την τρέχουσα τιμή και επιλέγουμε τη μικρότερη. Στη συγκεκριμένη περίπτωση η απόσταση από το σημείο Α στο σημείο Γ είναι 4. Ανανεώνουμε την τιμή του σημείου Γ από άπειρο σε 4.

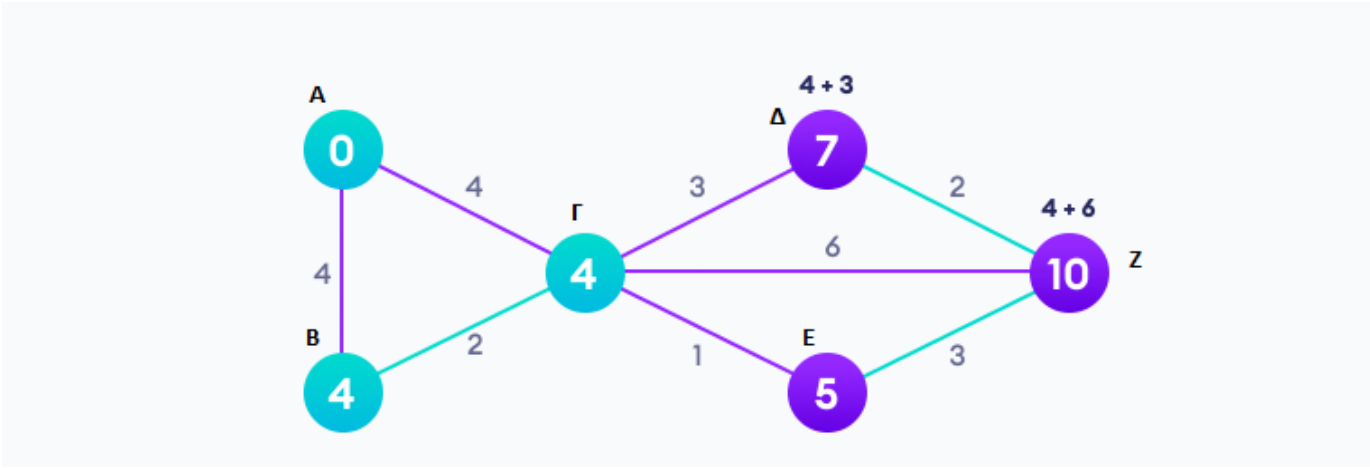


Παρατηρούμε ότι η διαδρομή Α → Β → Γ έχει μεγαλύτερη απόσταση από την ήδη καταχωρημένη τιμή, οπότε δεν ανανεώνουμε την τιμή για το σημείο Γ (4 < 4+2).

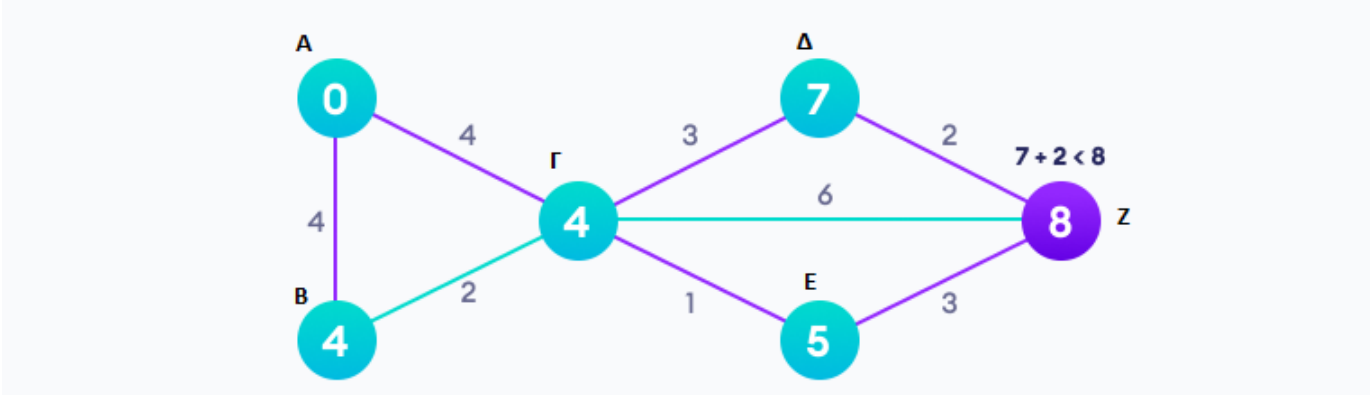
[7]Gass, Saul; Fu, Michael (2013). Gass, Saul I; Fu, Michael C (eds.). "Dijkstra's Algorithm". Encyclopedia of Operations Research and Management Science. Springer. 1. doi:10.1007/978-1-4419-1153-7. ISBN 978-1-4419-1137-7 – via Springer Link.



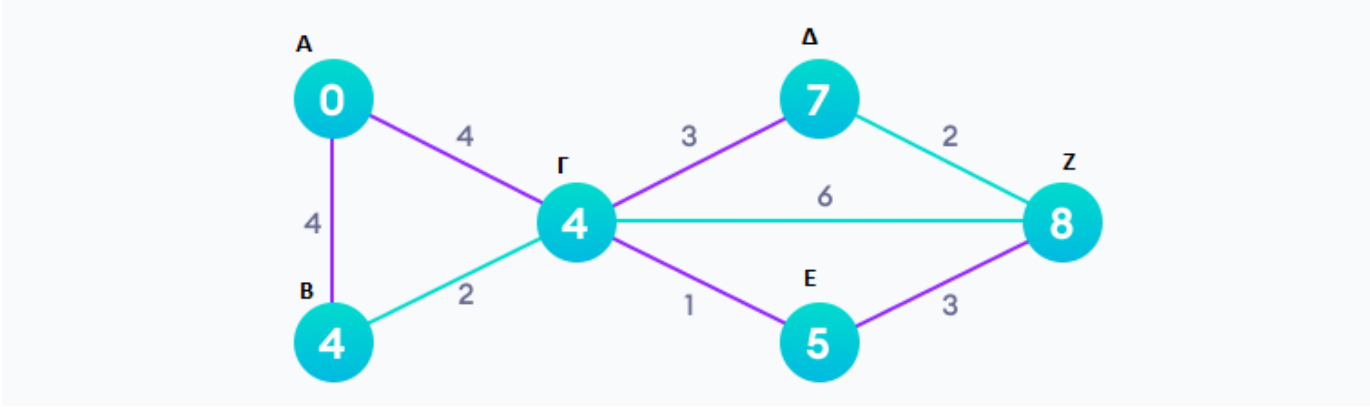
Έτσι θεωρούμε το σημείο Β ανενεργό και δεν το ξαναεπισκεπτόμαστε, συνεχίζοντας από το σημείο Γ με ενεργή ακμή την ΑΓ.
Ανανεώνουμε τις αποστάσεις για τα γειτονικά σημεία από το σημείο Γ.



Μετά από κάθε υπολογισμό επιλέγουμε τη διαδρομή με τη μικρότερη απόσταση. Στο παράδειγμα μας η διαδρομή αυτή είναι η Γ → Ε.
Υπολογίζοντας την απόσταση στο σημείο προορισμού ανανεώνουμε για δεύτερη φορά την τιμή του σημείου Z, θεωρώντας την ακμή ΓΖ ανενεργή καθώς έχει μεγαλύτερη απόσταση.
Τέλος, υπολογίζουμε την απόσταση για την ακμή ΔΖ και δεν ανανεώνουμε την τιμή στο σημείο Z, καθώς η απόσταση είναι μεγαλύτερη.



Η Παραπάνω διαδικασία μπορεί να επαναληφθεί τόσες φορές, όσα σημεία και αν υπάρχουν. Επομένως, το επιθυμητό – βέλτιστο μονοπάτι είναι το ΑΓ → ΓΕ → ΕΖ.



Η χρήση του αλγόριθμου του Dijkstra δεν είναι πάντα η καλύτερη επιλογή, καθώς ο χρόνος που απαιτείται για εύρεση αποτελέσματος ποικίλει ανάλογα με τον γράφο και τις δομές δεδομένων που χρησιμοποιούνται για την επίλυσή του. Σε μια γενική περίπτωση, ο χρόνος που χρειάζεται ο αλγόριθμος είναι O(|V|m+(|V|+|E|)k)[2], όπου V είναι το σύνολο των κόμβων, E το σύνολο των ακμών του γράφου, m ο χρόνος που χρειάζεται για την εξαγωγή του μη-επεξεργασμένου κόμβου με την ελάχιστη ετικέτα απόστασης και k ο χρόνος που χρειάζεται για την αλλαγή της ετικέτας απόστασης.

Στην πιο απλή περίπτωση, όπου οι (μη-επεξεργασμένοι) κόμβοι V\S αποθηκεύονται σε μια λίστα (linkedlist) ή έναν πίνακα (array), η αναζήτηση του στοιχείου με το μικρότερο κλειδί (ετικέτα απόστασης, d) γίνεται γραμμικά, εξετάζοντας ένα προς ένα όλα τα στοιχεία. Επομένως, ο τύπος για την εύρεση του χρόνου που χρειάζεται ο αλγόριθμος γίνεται O(|V|^{2}+|E|)=O(|V|^{2}). Υπάρχει η δυνατότητα περαιτέρω μείωσής του με τη χρήση μιας ουράς προτεραιότητας, για να αποθηκεύονται οι κόμβοι που είναι μη-επεξεργασμένοι. Ανάλογα με την υλοποίηση της ουράς προτεραιότητας (με σωρό Fibonacci ή δυαδικό σωρό κ.λ.π.), οι χρόνοι υλοποίησης ποικίλλουν.

Το γεγονός ότι ο αλγόριθμος Dijkstra υπολογίζει τη βέλτιστη διαδρομή για κάθε βήμα, κάτι που μπορούμε να παρατηρήσουμε εύκολα από το παραπάνω παράδειγμα, δεν μας εγγυάται ότι η σύνθεση αυτών των διαδρομών κάθε βήματος δημιουργεί και την ιδανική διαδρομή. Για το λόγο αυτό είναι απαραίτητη μια απόδειξη ορθότητας του αλγόριθμου. Κάτι τέτοιο έχει πραγματοποιηθεί από διάφορους μελετητές, και διδάσκεται σε ινστιτούτα αν το κόσμο.[8][9][10]

Όπως προαναφέρθηκε, σε περίπτωση που ακμές του γράφου έχουν αρνητικά βάρη, ο αλγόριθμος του Dijkstra δεν μπορεί μας εγγυηθεί σωστά αποτελέσματα[11]. Αυτό συμβαίνει επειδή σε περίπτωση που ένας κόμβος έχει θεωρηθεί ανενεργός, ο αλγόριθμος δεν επανεξετάζει τον κόμβο αυτό. Έτσι, ακόμα και αν μπορεί να προκύψει μια συντομότερη διαδρομή μέσω μιας ακμής αρνητικού βάρους, αυτή δεν λαμβάνεται υπόψη από τον αλγόριθμο. Η λειτουργία του χαρακτηρισμού ενός κόμβου ως ανενεργού (επεξεργασμένου) δεν μπορεί να αφαιρεθεί, καθώς σε μια τέτοια περίπτωση ο αλγόριθμος μπορεί να μπει σε έναν ατέρμονο βρόγχο με συνεχή επανεξέταση βρόγχων και μείωση των ετικετών απόστασής τους. Αυτό μπορεί να συμβεί όταν ένας γράφος περιέχει αρνητικό κύκλο,

[8]https://www2.cs.duke.edu/courses/fall14/compsci330/notes/scribe7.pdf

[9]https://web.cs.ucdavis.edu/~amenta/w10/dijkstra.pdf

[10]http://community.wvu.edu/~krsubramani/courses/fa05/gaoa/qen/dijk.pdf

11http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra_new/index.html#dijkstra

δηλαδή μια διαδρομή από έναν κόμβο προς τον ίδιο, για την οποία το αρνητικό βάρος είναι αρνητικό. Με αυτό τον τρόπο ο αλγόριθμος θα εξετάζει συνέχεια τους ίδιους κόμβους στον κύκλο, και θα μειώνει τις ετικέτες τους όλο και περισσότερο, χωρίς να σταματάει ποτέ. Με λίγα λόγια, σε έναν γράφο με κύκλο αρνητικού βάρους, δεν μπορεί να υπάρξει συντομότερη διαδρομή, αφού κάθε φορά που θα περνάμε από την ακμή αρνητικού βάρους, η διαδρομή θα γίνεται όλο και συντομότερη.

Ενσωμάτωση του αλγόριθμου Dijkstra στην εφαρμογή

Στην περίπτωση της εφαρμογής μας, πατώντας το κουμπί “Calculate” γίνεται κλήση του αλγόριθμου υπολογισμού βέλτιστης διαδρομής του Dijkstra. Για κάθε μια από τις τέσσερις περιπτώσεις, γίνεται κλήση σε διαφορετική βάση και έχουμε διαφορετικά αποτελέσματα. Για το παράδειγμα Σκαραμαγκά – Μυτιλήνη γίνονται οι εξής κλήσεις:

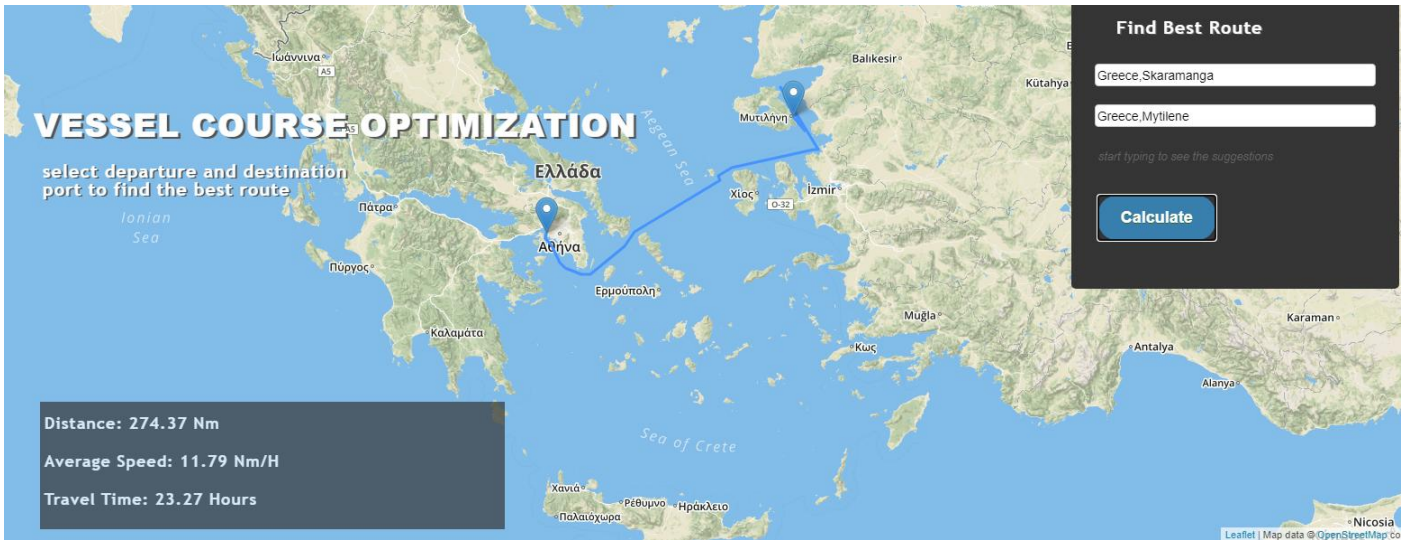
1. Δέκα ναυτικά μίλια με την απόσταση ως βάρος
2. Δέκα ναυτικά μίλια με την ταχύτητα διάσχισης ως βάρος
3. Εβδομήντα ναυτικά μίλια με την απόσταση ως βάρος
4. Εβδομήντα ναυτικά μίλια με την ταχύτητα διάσχισης ως βάρος

Σε κάθε μια περίπτωση γίνεται κλήση του αλγόριθμου Dijkstra. Κάθε φορά παράγεται ένα array από σημεία ως η βέλτιστη διαδρομή.
Για το array αυτό, καλείται η βάση με τα στοιχεία των πολύγωνων, ώστε οι συντεταγμένες του να τοποθετηθούν σε ένα νέο array.
Στη συνέχεια χρησιμοποιείται η επιλογή L.polyline που παρέχεται από το API του leaflet, ώστε να δημιουργηθεί μια ευθεία με ενωμένες τις συντεταγμένες.
Τέλος, απεικονίζουμε αυτή την ευθεία στο χάρτη μας και τα αποτελέσματα που αφορούν την απόσταση, το χρόνο άφιξης στον προορισμό και τη μέση ταχύτητα. Στο συγκεκριμένο παράδειγμα κώδικα γίνεται χρήση της βάσης για τις αποστάσεις δέκα ναυτικών μιλίων.

```
function findNewShortestPath() {
    /* Κλήση της εκάστοτε βάσης με τα κατάλληλα δεδομένα ώστε να δημιουργηθεί ο γράφος
        Οι βάσεις που μπορούν να χρησιμοποιηθούν είναι οι Distances10Nm, Distances10NmSpd, Distances70Nm, Distan
ces70NmSpd */
    var xhr = new XMLHttpRequest();
    xhr.open("GET", 'http://localhost:3000/api/Distances70Nm/', true);
    xhr.send();
    xhr.onload = function () {
        if (this.status == 200) {
            var pointDistances = JSON.parse(this.responseText);
            /* Δημιουργία γράφου – εκτέλεση του αλγόριθμου Dijsktra */
            var g = new Graph();
            for (i = 0; i < pointDistances.length; i++) {
                g.addVertex(pointDistances[i].PointName, pointDistances[i].distances)
            };
            /* Εκτέλεση αλγόριθμου shortest path */
            // Αποτύπωση διαδρομής σε log, αντιστρέφοντας το array και προσθέτοντας το πρώτο node
            var path = g.shortestPath(closestStartPointName, closestEndPointName).concat([closestStartPointName]
).reverse()
            var xhr3 = new XMLHttpRequest();
            xhr3.open("GET", 'http://localhost:3000/api/polygons/', true);
            xhr3.send();
            xhr3.onload = function () {
                if (this.status == 200) {
                    var polygonCoords = JSON.parse(this.responseText);
```

```
                    var coordArray = [];
                    var speed = 0;
                    total_dist = 0;
                    for (var p = 0; p < path.length; p++) {
                        for (var po = 0; po < polygonCoords.length; po++) {
                            if (polygonCoords[po].PolygonNo + "start" == path[p]) {
                                ar2 = [];
                                speed = speed + polygonCoords[po].Avg_speed;
                                if (p > 0) {
                                    var dist = findDistance(polygonCoords[po].Coordinates.Start.x, polygonCoords
[po].Coordinates.Start.y, point_coords_x, point_coords_y, "N");
                                    total_dist = total_dist + dist;
                                }
                                var point_coords_x = polygonCoords[po].Coordinates.Start.x;
                                var point_coords_y = polygonCoords[po].Coordinates.Start.y;
                                ar2.push(polygonCoords[po].Coordinates.Start.x);
                                ar2.push(polygonCoords[po].Coordinates.Start.y);
                                coordArray.push(ar2);
                            }
                            else if (polygonCoords[po].PolygonNo + "end" == path[p]) {
                                ar2 = [];
                                speed = speed + polygonCoords[po].Avg_speed;
                                if (p > 0) {
                                    var dist = findDistance(polygonCoords[po].Coordinates.End.x, polygonCoords[p
o].Coordinates.End.y, point_coords_x, point_coords_y, "N");
                                    total_dist = total_dist + dist;
                                }
                                var point_coords_x = polygonCoords[po].Coordinates.End.x;
                                var point_coords_y = polygonCoords[po].Coordinates.End.y;
                                ar2.push(polygonCoords[po].Coordinates.End.x);
                                ar2.push(polygonCoords[po].Coordinates.End.y);
                                coordArray.push(ar2);
                            }
                        }

                    }
                }
            }
        }
    }
};
```

Στη περίπτωση της εύρεσης βέλτιστης απόστασης μεταξύ των λιμένων Σκαραμαγκά και Μυτιλήνης η απεικόνιση στο χάρτη έχει ως εξής:



Εικόνα 8 - Αποτέλεσμα Υπολογισμού Βέλτιστης Διαδρομής από Σκαραμαγκά σε Μυτιλήνη με βάρος την απόσταση μεταξύ σημείων.

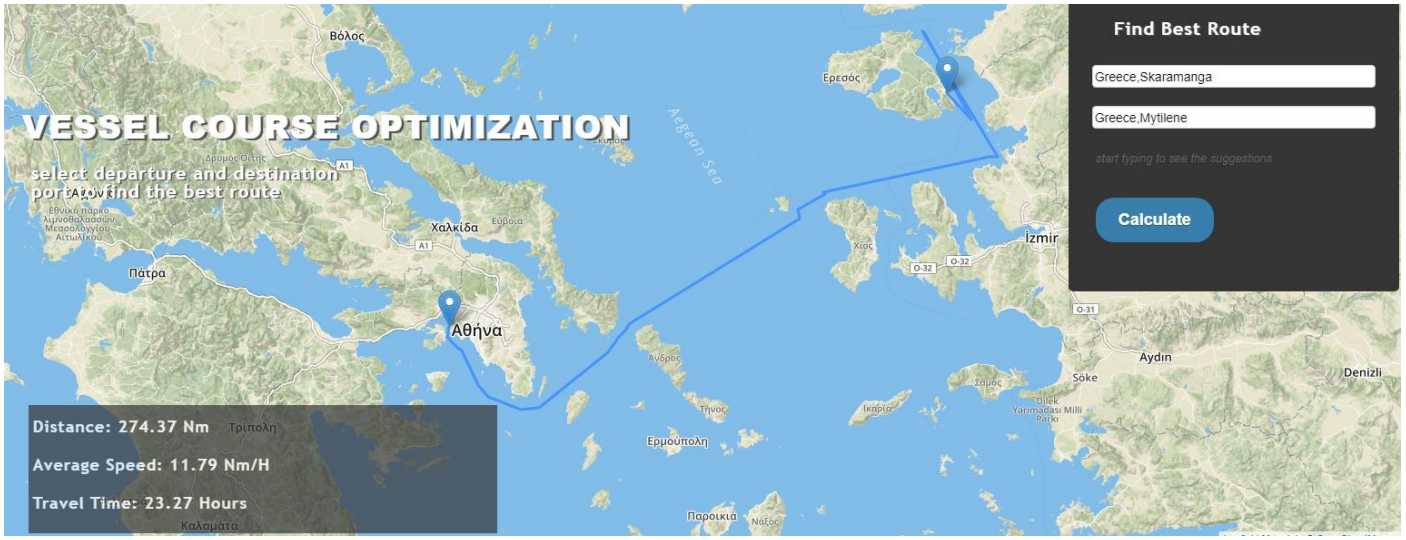
Συμπεράσματα

Σύγκριση μεθόδων και αποτελέσματα.

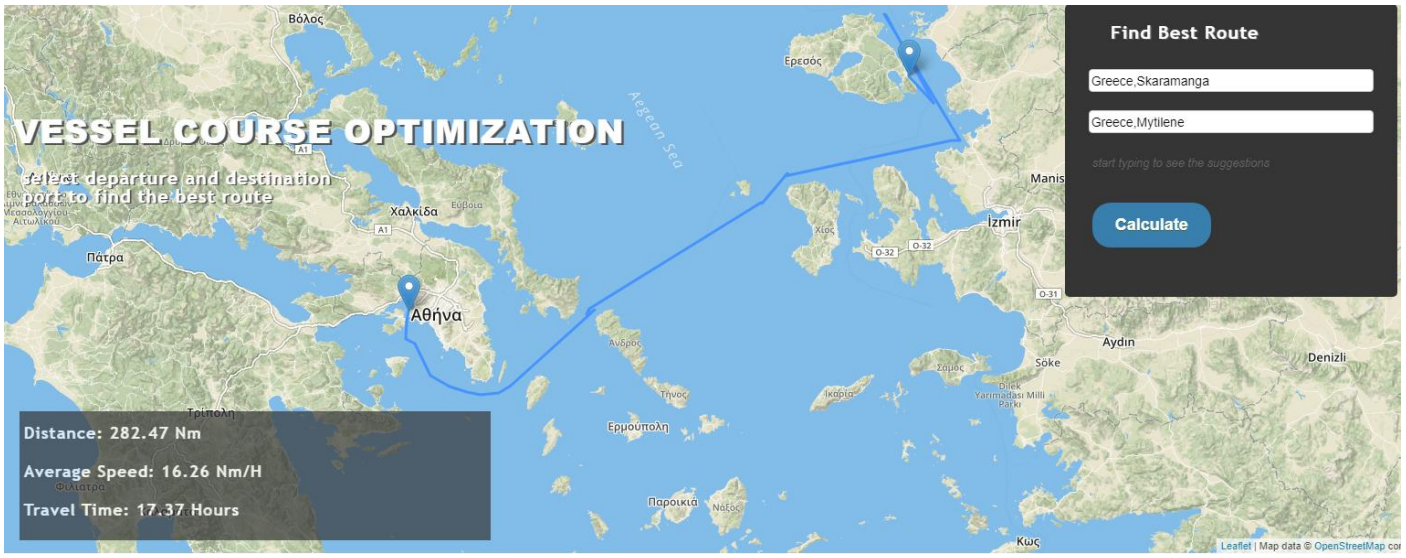
Κύριος σκοπός αυτής της εργασίας είναι η μελέτη των διαφορετικών βέλτιστων διαδρομών που υπολογίζονται, αλλάζοντας τις παραμέτρους στον αλγόριθμό μας. Με τη σύγκριση των διαφορετικών αποτελεσμάτων μπορούμε να έχουμε μια καλύτερη εικόνα για το ποια μέθοδος λειτουργεί καλύτερα, αναλόγως των λιμανιών που έχουμε επιλέξει. Επίσης, μπορούμε να οδηγηθούμε σε νέα συμπεράσματα για τροποποιήσεις που μπορούν να γίνουν, ή και νέες λειτουργικότητες που μπορούν να προστεθούν στο μέλλον.

Με βάση τα παραπάνω έγινε σύγκριση των τεσσάρων διαφορετικών τρόπων υπολογισμού για τρείς διαφορετικές περιπτώσεις. Και στις τρείς εξ' αυτών χρησιμοποιήθηκε ως λιμένας αφετηρίας ο λιμένας του Σκαραμαγκά. Στην πρώτη περίπτωση ο λιμένας προορισμού ήταν η Μυτιλήνη, στη δεύτερη και τρίτη περίπτωση λιμένας προορισμού ήταν η Λευκάδα και η Ρόδος αντίστοιχα.

Για την πρώτη περίπτωση έχουμε τα εξής αποτελέσματα:



Εικόνα 9 - Υπολογισμός με βάρος στις ακμές μεταξύ σημείων την απόσταση από ένα σημείο στο άλλο με περιορισμό αποστάσεων τα δέκα ναυτικά μίλια

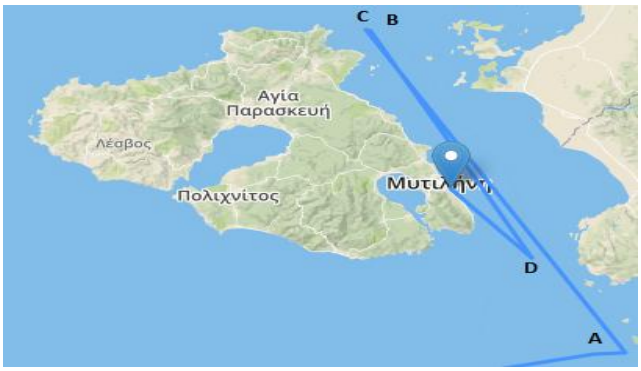


Εικόνα 10 - Υπολογισμός με βάρος στις ακμές μεταξύ σημείων τον χρόνο διάσχισης από ένα σημείο στο άλλο με περιορισμό αποστάσεων τα δέκα ναυτικά μίλια.

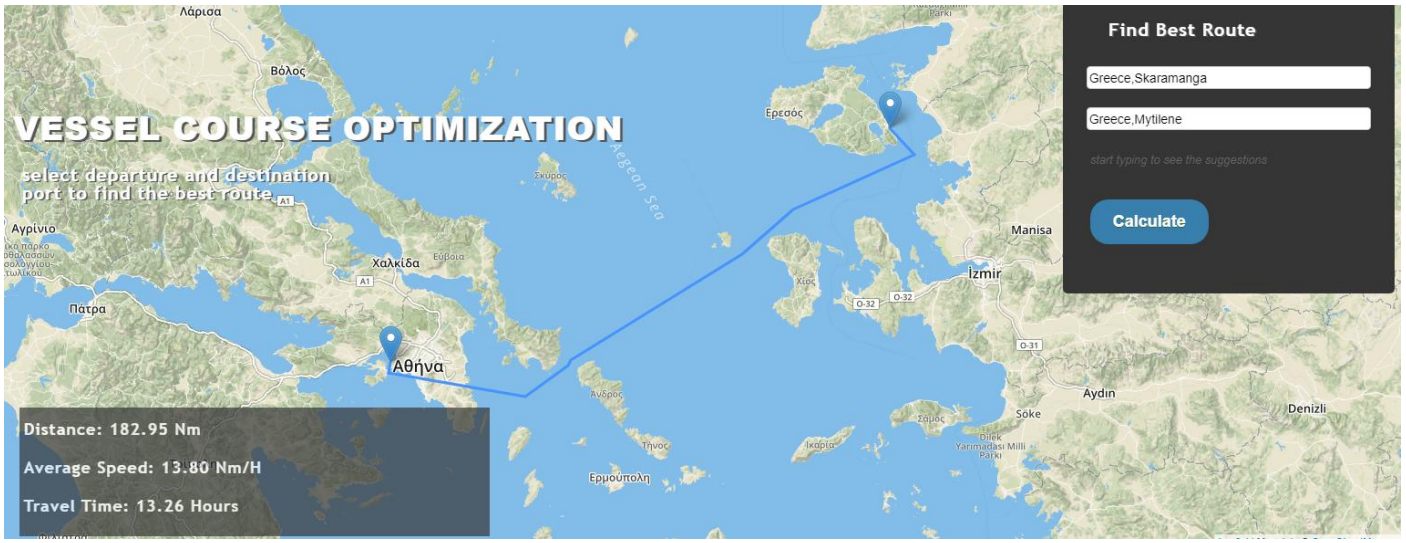
Στην περίπτωση του περιορισμού των δέκα ναυτικών μιλίων δεν παρατηρούμε σημαντικές διαφορές μεταξύ του βάρους απόστασης με το βάρος χρόνου διάσχισης. Αυτό συμβαίνει διότι ο αριθμός των επιλογών από σημείο σε σημείο που διαθέτει ο αλγόριθμος είναι πολύ περιορισμένος. Παρόλα αυτά με την ελευθερία για επιλογή διαδρομών με μικρότερο χρόνο διάσχισης βλέπουμε ότι ο χρόνος άφιξης στον προορισμό μας μειώνεται σημαντικά καθώς έχουμε αρκετά μεγαλύτερη μέση ταχύτητα και ας διασχίζουμε λίγο μεγαλύτερη απόσταση.

Επίσης, κοντά στο νησί της Λέσβου παρατηρείται μια περιττή ακολουθία διαδρομών μέχρι το λιμάνι. Αυτό συμβαίνει για δύο λόγους. Αρχικά, όπως και πριν λόγο του περιορισμού των δέκα ναυτικών μιλίων, δεν υπάρχει πληθώρα σημείων που μπορούν να επιλεχθούν. Τι εμποδίζει όμως τον αλγόριθμο να πάει από το σημείο κοντά στη Τουρκία αμέσως στο λιμάνι της Μυτιλήνης ώστε να αποφύγει την περιττή διαδρομή;

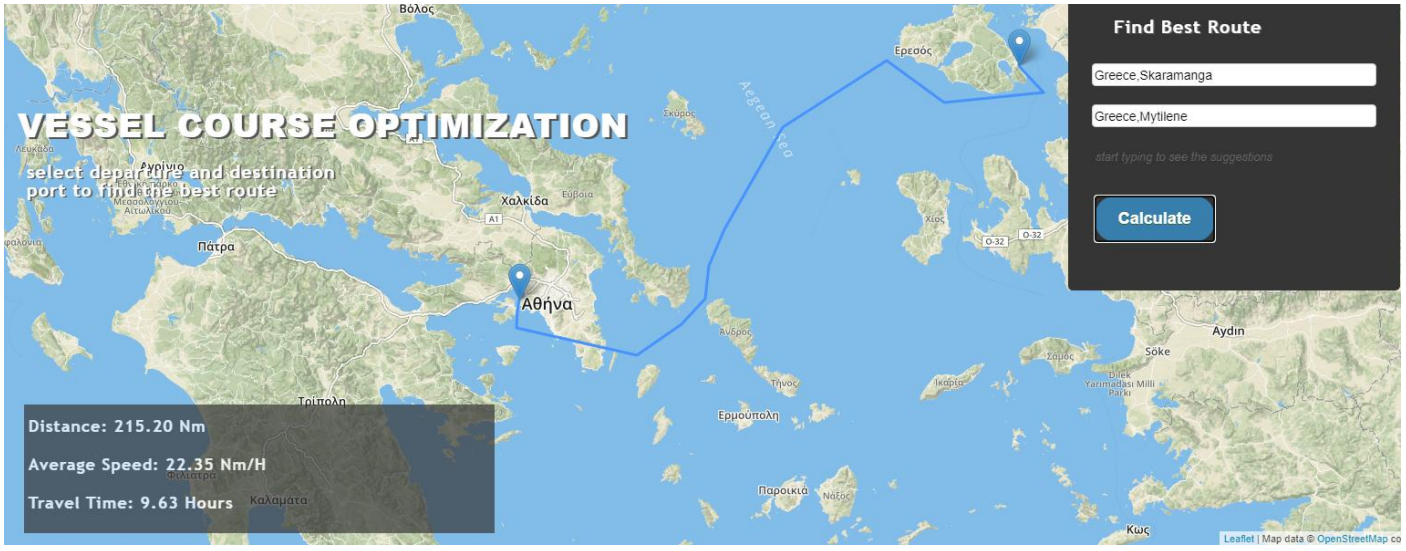
Σε αυτή τη περίπτωση, φταίει η παραδοχή που έχουμε κάνει δηλαδή ότι μέσα σε ένα πολύγωνο υπάρχει πάντα μια γραμμή που συνδέει αρχή και τέλος και μπορεί να χρησιμοποιηθεί όσο μακριά και αν είναι. Έτσι, ακολουθείται αυτή η διαδρομή για δύο αρχές και τέλη πολύγωνων, την AB και την CD. Σε αντίθετη περίπτωση, το πιο πιθανό είναι ότι δεν θα υπήρχε διαδρομή από το σημείο A στο σημείο D και ο αλγόριθμος δεν θα μπορούσε να έχει αποτέλεσμα.



Για να αντιμετωπιστεί αυτό το πρόβλημα, θα πρέπει να αλλάξουμε τον περιορισμό, ώστε να επιτρέπει ακμές με μεγαλύτερες αποστάσεις μεταξύ των σημείων. Τα παραδείγματα με βάρος την απόσταση και τον χρόνο διάσχισης με περιορισμό τα εβδομήντα ναυτικά μίλια που ακολουθούν απεικονίζουν μια τέτοια περίπτωση.



Εικόνα 11 - Υπολογισμός με βάρος στις ακμές μεταξύ σημείων την απόσταση από ένα σημείο στο άλλο με περιορισμό αποστάσεων τα εβδομήντα ναυτικά μίλια



Εικόνα 12 - Υπολογισμός με βάρος στις ακμές μεταξύ σημείων τον χρόνο διάσχισης από ένα σημείο στο άλλο με περιορισμό αποστάσεων τα εβδομήντα ναυτικά μίλια.

Στο παραπάνω παράδειγμα βλέπουμε αρκετές διαφορές μεταξύ των δύο υλοποιήσεων. Αυτό συμβαίνει διότι υπάρχει πληθώρα σημείων που μπορεί να επιλέξει ο αλγόριθμος με βάση τα δεδομένα που του δίνουμε. Οι διαφορές αυτές είναι ένα πολύ καλό παράδειγμα για το πώς επηρεάζεται η εύρεση της συντομότερης διαδρομής, ανάλογα με το βάρος που έχουμε δώσει στις

ακμές.

Στην πρώτη περίπτωση έχουμε σχεδόν μια ευθεία γραμμή (εάν εξαιρέσουμε τα σημεία πριν την Εύβοια) από την αφετηρία στον προορισμό μας. Αυτό είναι κάτι πολύ λογικό με βάση την παραδοχή της απόστασης μεταξύ σημείων ως βάρος.

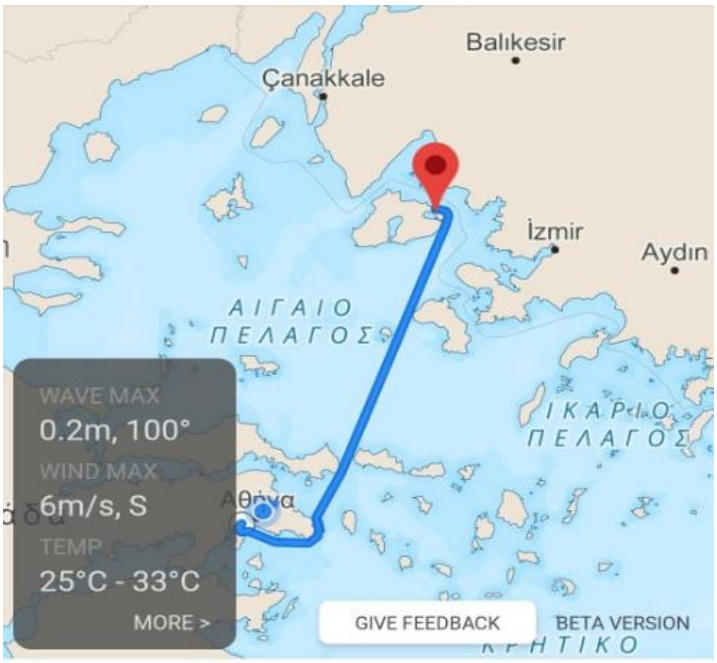
Αντίθετα, στη δεύτερη περίπτωση βλέπουμε ότι ο αλγόριθμος έχει επιλέξει μία διαφορετική διαδρομή. Αυτό συμβαίνει, καθώς λόγω της διέλευσης ανάμεσα στα νησιά Χίο και Ψαρά τα πλοία αναγκάζονται να ελαττώσουν ταχύτητα και με αυτό τον τρόπο να αυξήσουν τον χρόνο άφιξης στον προορισμό. Έτσι, επιλέγεται μία διαδρομή που επικεντρώνεται περισσότερο στην πλοήγηση στην ανοιχτή θάλασσα καθώς είναι και η ταχύτερη. Πιθανόν αν είχαμε περισσότερα σημεία η διαδρομή να ήταν πιο σωστά συντονισμένη, ώστε να μην έχουμε περιττές περιοχές που θα διασχίσει το πλοίο.

Παρατηρούμε επίσης ότι σε σύγκριση με το παράδειγμα των δέκα ναυτικών μιλίων, η άφιξη στο λιμάνι της Μυτιλήνης δεν περνάει από περιττά σημεία πολύγωνων μιας και η απόσταση στο σημείο D μπορεί να καλυφθεί πλέον.

Όπως και πριν παρατηρούμε ότι η χρήση του χρόνο διάσχισης ως βάρος μας φέρνει πολύ καλύτερα αποτελέσματα σχετικά με το χρόνο άφιξης στον προορισμό μας.

Το μεγαλύτερο πρόβλημα που προκαλεί η χρήση δεδομένων με τόσο μεγάλες αποστάσεις μεταξύ σημείων είναι η πιθανότητα διέλευσης από ξηρά. Κάτι τέτοιο είναι εμφανές στο παράδειγμα που εξετάζουμε καθώς και στις δύο περιπτώσεις η διαδρομή περνάει μέσα από τμήμα του νομού Αττικής. Διαδρομές σαν αυτή είναι αναπόφευκτες στις περιπτώσεις που έχουμε τόσο μεγάλες αποστάσεις. Βέβαια, τέτοια περιστατικά είναι πιθανόν να συμβούν όσο μικρές και αν είναι οι αποστάσεις που έχουμε ορίσει, όπως π.χ. σε περιπτώσεις όπου το λιμάνι μιας περιοχής βρίσκεται πίσω από ένα ακρωτήρι και το τελευταίο σημείο πριν το λιμάνι βρίσκεται από την άλλη πλευρά του ακρωτηρίου.

Οι αποστάσεις για κάθε περίπτωση του πρώτου παραδείγματος υπολογίστηκαν, ώστε να γίνει η σύγκριση, με την χρήση της εφαρμογής C – MAP (βλ. Κεφάλαιο 1, Σχετική Εργασία).



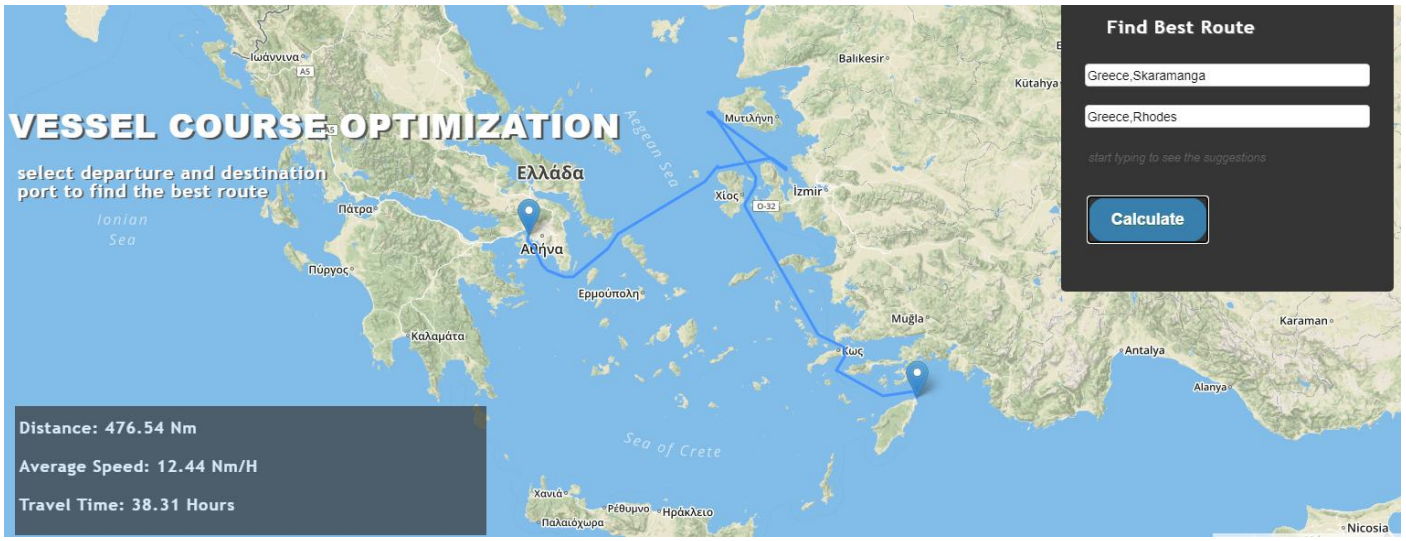
38h 7m (190.45 nm)

Όπως παρατηρούμε η εφαρμογή υπολογίζει την απόσταση στα 190.45 ναυτικά μίλια. Συγκριτικά έχουμε:

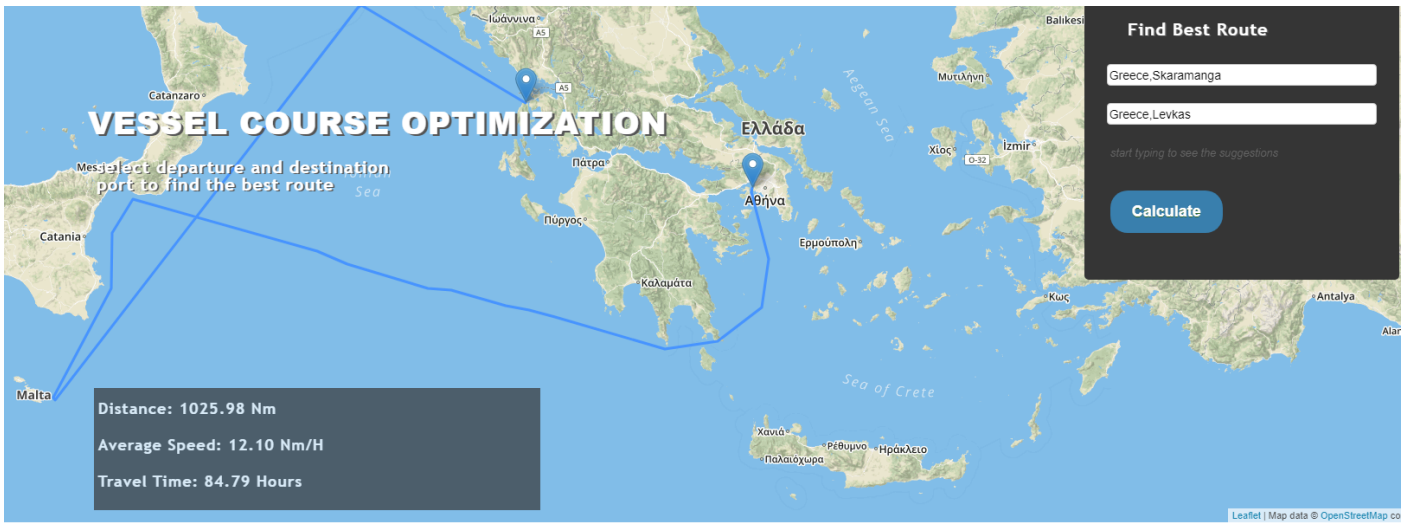
Μέθοδος	Απόσταση
C MAP	190.45 Nm
Βάρος απόσταση για 10Nm	274.37 Nm
Βάρος ταχύτητα διάσχισης για 10Nm	282.47 Nm
Βάρος απόσταση για 70Nm	182.95 Nm
Βάρος ταχύτητα διάσχισης για 70Nm	215.20 Nm

Πίνακας 3 - Σύγκριση Αποστάσεων

Παρόμοια αποτελέσματα είχαμε και στις περιπτώσεις των λιμένων προορισμού για Λευκάδα και Ρόδο. Ενδεικτικά έχουμε:



Εικόνα 13 - Διαδρομή μεταξύ λιμένων Σκαραμαγκά – Ρόδου με βάρος την απόσταση μεταξύ σημείων για αποστάσεις μικρότερες των δέκα ναυτικών μιλίων.



Εικόνα 14 - Διαδρομή μεταξύ λιμένων Σκαραμαγκά – Λευκάδας με βάρος την ταχύτητα διάσχισης μεταξύ σημείων για αποστάσεις μικρότερες των εβδομήντα ναυτικών μιλίων.

Στις παραπάνω περιπτώσεις η έλλειψη πληθώρας σημείων επιλογής για τον αλγόριθμο, τον αναγκάζει να επιλέξει διαδρομές που απέχουν πολύ από τις βέλτιστες, καθώς είναι οι μόνες που μπορούν να οδηγήσουν στο προορισμό μας.

Συμπεράσματα

Από τις διαφορετικές περιπτώσεις που μελετήσαμε προκύπτει ότι τα δεδομένα που λαμβάνουμε μέσω του AIS είναι ικανά, με την κατάλληλη επεξεργασία τους, να χρησιμοποιηθούν σε μια εφαρμογή εύρεσης βέλτιστης διαδρομής. Για να γίνει όμως αυτό δυνατό, πρέπει να συμβαίνουν τα εξής :

Για τη βέλτιστη επίδοση του αλγόριθμου του Dijkstra σε ένα περιβάλλον, όπως αυτό της θάλασσας, χρειάζεται ένας πολύ μεγάλος αριθμός σημείων. Όσο περισσότερα τα σημεία, τόσο μεγαλύτερη ακρίβεια θα έχει και η εύρεση της βέλτιστης διαδρομής. Βέβαια, με την εισαγωγή ενός τέτοιου αριθμού σημείων είναι πολύ πιθανό να αυξηθεί σημαντικά ο χρόνος για την παραγωγή αποτελέσματος, οπότε πρέπει να βρεθεί μια «χρυσή τομή» μεταξύ αριθμού δεδομένων και χρόνου ολοκλήρωσης των υπολογισμών του αλγόριθμου.

Η διάσχιση σημείων που θεωρούνται ξηρά, ύφαλοι, ξέρες κτλ αποτελεί ένα σημαντικό πρόβλημα, του οποίου η επίλυση προϋποθέτει την ύπαρξη δεδομένων ή λογισμικού που να επιβεβαιώνει αν κάθε σημείο μίας διαδρομής βρίσκεται σε θαλάσσια περιοχή. Η διέλευση από περιοχές όπως οι παραπάνω μπορεί να περιοριστεί με την ύπαρξη πληθώρας σημείων, στα οποία θα εμπεριέχεται η πληροφορία της ταχύτητας των πλοίων που έχουν περάσει από εκεί. Με αυτό τον τρόπο πλοία κοντά σε σημεία ξηράς θα έχουν πολύ μικρή ταχύτητας και ο αλγόριθμος δε θα επιλέξει αυτά, καθώς η ταχύτητα διάσχισης θα είναι πολύ μικρή.

Η εύρεση της βέλτιστης διαδρομής δεν μπορεί να περιοριστεί μόνο σε δεδομένα όπως ο χρόνος διάσχισης από ένα σημείο σε ένα άλλο. Για να έχει χρήση η εφαρμογή σε πραγματικές συνθήκες, πρέπει να ληφθούν πολλοί ακόμη παράγοντες υπόψη, οι οποίοι θα αναλυθούν παρακάτω.

Στη σύγκριση αποστάσεων μεταξύ εφαρμογής C MAP και των δικών μας μεθόδων βλέπουμε ότι για την περίπτωση των εβδομήντα ναυτικών μιλίων με βάρος την απόσταση, πετυχαίνουμε καλύτερο αποτέλεσμα. Αυτό βέβαια συμβαίνει λόγω της διάσχισης ξηράς σε μια από τις ακμές.

Μελλοντικές Βελτιώσεις

Με βάση τις παραπάνω παρατηρήσεις, συμπεραίνουμε ότι ο αλγόριθμος μπορεί να λειτουργήσει πολύ καλύτερα και ότι υπάρχουν πολλές διαφοροποιήσεις που μπορούν να γίνουν ώστε να έχουμε καλύτερα αποτελέσματα. Ενδεικτικά :

Η ανάγκη για μεγαλύτερο αριθμό δεδομένων είναι ένα από τα πιο σημαντικά προβλήματα του αλγόριθμου. Κάτι τέτοιο όπως προαναφέρθηκε μπορεί να ελαττώσει την εμφάνιση πολλών από τα προβλήματα που συναντήσαμε, αλλά όχι απαραίτητα να την εξαλείψει. Βέβαια αν η κλίμακα της εύρεσης βέλτιστης διαδρομής επεκταθεί για τα λιμάνια εκτός των θαλάσσιων υδάτων της Ελλάδας, ίσως χρειαστεί να χρησιμοποιηθεί άλλος αλγόριθμος με καλύτερη επίδοση για έναν πολύ μεγάλο αριθμό δεδομένων και υψηλότερη πολυπλοκότητα λόγο αυτών.

Εκτός από την ταχύτητα διάσχισης, υπάρχει πληθώρα άλλων παραγόντων που επηρεάζουν το χρόνο που μπορεί να κάνει ένα πλοίο να φτάσει στο προορισμό του. Οι περισσότεροι επηρεάζουν τη ταχύτητα του πλοίου ή πιθανές παρακάμψεις που πρέπει να κάνει και ως εκ τούτου το τελικό αποτέλεσμα. Παραδείγματα τέτοιων παραγόντων είναι :

- Τα καιρικά φαινόμενα. Σε περίπτωση δυσμενών καιρικών συνθηκών το πλοίο θα έχει ελαττωμένη ταχύτητα ή θα αναγκαστεί να κάνει κάποια παράκαμψη για να τα αποφύγει με αποτέλεσμα να διασχίσει μεγαλύτερη διαδρομή.
- Περιοχές υψηλού κινδύνου λόγο πειρατείας. Σε περιοχές όπως αυτές το πλοίο θα πρέπει και πάλι να επιλέξει μια μεγαλύτερη διαδρομή για να ταξιδέψει με τη μεγαλύτερη δυνατή ασφάλεια. Στη συγκεκριμένη περίπτωση οι περιοχές αυτές μπορούν να χαρακτηριστούν ως ξηρά ώστε να αποφεύγονται από το πλοίο.

- Θαλάσσιες περιοχές που περιέχουν ύφαλους, ξέρες, ναυάγια θα πρέπει να συνυπολογιστούν στη χαρτογράφηση και στα σημεία που πρέπει να αποφευχθούν ως ξηρά. Εκτεταμένη αναφορά σε σημεία ξηράς γίνεται παρακάτω.

Ένας ακόμη παράγοντας που περιόριζε την αξιοπιστία των διαδρομών μας είναι η πιθανότητα ύπαρξης ξηράς μεταξύ δύο σημείων. Όπως προαναφέρθηκε αυτή η πιθανότητα υπάρχει όσο μικρός και αν είναι ο περιορισμός της απόστασης ανάμεσα σε δύο σημεία. Ως ξηρά επίσης πρέπει να θεωρούνται και σημεία από τα οποία δεν πρέπει να περνάει ένα πλοίο. Για να μπορέσουμε να αποφύγουμε περιπτώσεις όπως αυτές, θα πρέπει να υπάρχουν δεδομένα στα οποία εμπεριέχεται η πληροφορία για το αν κάθε σημείο της διαδρομής μας είναι ξηρά η όχι. Τέτοια δεδομένα παρέχονται αλλά δεν είναι πάντα αξιόπιστα, ειδικά σε περιπτώσεις ύφαλων όπου εκτός και είναι δεδομένα που έχουν παραχθεί για θαλάσσιες διαδρομές, ένας ύφαλος μπορεί να θεωρηθεί κομμάτι θάλασσας. Ακόμα όμως και αν είχαμε αυτή τη πληροφορία, θα έπρεπε για κάθε ζευγάρι συντεταγμένων μεταξύ δύο σημείων να ελέγχεται αν αυτό βρίσκεται σε ξηρά, κάτι το οποίο θα έκανε το χρόνο υλοποίησης του αλγόριθμου πολύ μεγαλύτερο. Το δεδομένο της ταχύτητας σε κάθε σημείο μπορεί να μας παρέχει μια μεγαλύτερη αξιοπιστία καθώς κοντά σε ξηρά θα έχουμε πολύ μικρότερη ταχύτητα αλλά και πάλι δεν είναι μια παραδοχή που μπορεί με σιγουριά να δώσει μια ασφαλή διαδρομή. Μία λύση θα ήταν να είχαμε για όλες τις συντεταγμένες του χάρτη την πληροφορία που παίρνουμε από το AIS ώστε να έχουμε δεδομένα για όλη την επιφάνεια της θάλασσας, και να μην επιλέγουμε ζευγάρια συντεταγμένων για τα οποία δεν έχουμε.

Τέλος μία ακόμη ιδέα για την βελτιστοποίηση του αλγόριθμου θα ήταν να υπάρχει η δυνατότητα να γνωρίζουμε τα δρομολόγια των πλοίων που περνάνε από τη διαδρομή μας. Με αυτό τον τρόπο θα μπορούσαν να αποτραπούν παρακάμψεις και μειώσεις ταχύτητας που γίνονται για αποφυγή συγκρούσεων και να οργανωθεί ένα βέλτιστο δρομολόγιο. Κάτι τέτοιο βέβαια προϋποθέτει ότι κάθε πλοίο θα εισάγει το δρομολόγιο του σε ένα σύστημα και στην συνέχεια θα γίνονται οι υπολογισμοί συμπεριλαμβανομένων όλων των δρομολόγιων σαν παράμετρο, που ίσως να μην είναι εφικτό ακόμα.

Μια τελευταία αλλαγή που θα μπορούσε να βελτιώσει τα αποτελέσματα μας έχει να κάνει με τη μέση ταχύτητα σε ένα πολύγωνο. Για τον υπολογισμό της έχουμε πάρει την ταχύτητα στην αρχή του, την έχουμε προσθέσει με τη ταχύτητα στο τέλος του και έχουμε διαιρέσει δια δύο. Με αυτό τον τρόπο όμως, ιδιαίτερα σε μεγάλα πολύγωνα δεν έχουμε ακριβή εικόνα της ταχύτητας, καθώς μέσα στο πολύγωνο μπορεί να υπάρχουν πολλά νησιά και το πλοίο να αναγκαστεί να μειώσει ταχύτητα, είτε σε άλλη περίπτωση στην αρχή η τέλος του πολυγώνου να έχουμε πολύ χαμηλή ταχύτητα λόγο διάσχισης περιοχής κοντά σε ξηρά που δεν αντιπροσωπεύει το υπόλοιπο πολύγωνο. Και σε αυτή τη περίπτωση, το πρόβλημα περιορίζεται με την ύπαρξη πολλών σημείων το ένα κοντά στο άλλο ώστε να μην έχουμε πολλές αποκλίσεις.

Βιβλιογραφία

1) Yuanqiao Wen, Zhongyi Sui, Chunhui Zhou, Changshi Xiao, Qianqian Chen, Dong Han, Yimeng Zhang, Automatic ship route design between two ports: A data-driven method, Applied Ocean Research, 2020
2) S.W. Kim, H.K. Jang, Y.J. Cha, H.S. Yu, S.J. Lee, D.H. Yu, A.R. Lee, E.J. Jin, Development of a ship route decision-making algorithm based on a real number grid method, Applied Ocean Research, 2020
3) Έντυποι Ναυτικοί Χάρτες, Πολεμικό Ναυτικό Υδρογραφική Υπηρεσία, https://www.hnhs.gr/el/2015-05-28-16-58-20/2015-05-28-16-59-17/2015-05-16-18-43-48 , 02/09/2020.
4) Ηλεκτρονικοί Ναυτικοί Χάρτες, Πολεμικό Ναυτικό Υδρογραφική Υπηρεσία, https://www.hnhs.gr/el/2015-05-28-16-58-20/2015-05-28-16-59-17/2015-05-16-18-43-31 , 02/09/2020.
5) Dijkstra's algorithm, Wikipedia The Free Encyclopedia , https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#cite_note-14 , 02/09/2020.
6) Gass, Saul; Fu, Michael (2013). Gass, Saul I; Fu, Michael C (eds.). "Dijkstra's Algorithm". Encyclopedia of Operations Research and Management Science. Springer. 1. doi:10.1007/978-1-4419-1153-7. ISBN 978-1-4419-1137-7 – viaSpringerLink.
7) Σχεδιασμόςαλγορίθμων, JohnKleinberg, EvaTardos, 2008, σελ.173-175, ISBN 978-960-461-207-9
8) Example of Dijkstra's algorithm, Programiz , https://www.programiz.com/dsa/dijkstra-algorithm , 31/08/2020
9) Algorithms, S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani, 2006, σελ. 124-125
10) Nat Kell, Debmalya Panigrahi, Design and Analysis of Algorithms, Duke University, https://www2.cs.duke.edu/courses/fall14/compsci330/notes/scribe7.pdf , 9/16/2014
11) Proof for Dijkstra's Algorithm, University of California, Davis, https://web.cs.ucdavis.edu/~amenta/w10/dijkstra.pdf
12) K. Subramani, Correctness of Dijkstra's algorithm, West Virginia University, Morgantown, WV, http://community.wvu.edu/~krsubramani/courses/fa05/gaoa/gen/dijk.pdf
13) Moshe Sniedovich, Dijkstra's Algorithm revisited: the OR/MS Connexion, Department of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3052, Australia, http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra_new/index.html#dijkstra , 30/08/2020
14) Dijkstra's shortest path JavaScript implementation , GitHub https://gist.github.com/jaewook77/7be24f504285c3572c2f , 25/08/2020