Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Μηχανολόγων Μηχανικών

Διπλωματική εργασία

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΔΡΟΜΟΛΟΓΗΣΗΣ ΣΤΟΛΟΥ ΔΙΑΝΟΜΗΣ ΣΕ ΜΕΤΑΦΟΡΙΚΗ ΕΤΑΙΡΕΙΑ

Επιμέλεια: Πυλαρινός-Μαρκαντωνάτος Ανδρέας

Επιβλέπων καθηγητής: Νενές Γιώργος

Κοζάνη, Οκτώβριος 2024

Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Μηχανολόγων Μηχανικών

Διπλωματική εργασία

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΔΡΟΜΟΛΟΓΗΣΗΣ ΣΤΟΛΟΥ ΔΙΑΝΟΜΗΣ ΣΕ ΜΕΤΑΦΟΡΙΚΗ ΕΤΑΙΡΕΙΑ

Επιμέλεια: Πυλαρινός-Μαρκαντωνάτος Ανδρέας

Επιβλέπων καθηγητής: Νενές Γιώργος

Κοζάνη, Οκτώβριος 2024

Περίληψη

Η παρούσα εργασία παρουσιάζει μια προσέγγιση για τη βελτιστοποίηση της δρομολόγησης των φορτηγών διανομής για μια εταιρεία logistics με τη χρήση γραμμικού ακέραιου προγραμματισμού, με στόχο την ελαχιστοποίηση της απόστασης ή της διάρκειας ταξιδιού. Η προτεινόμενη λύση περιλαμβάνει την εφαρμογή του προβλήματος του πλανόδιου πωλητή (Travelling Salesman Problem) με τη χρήση του LINGO, ενός μαθηματικού λογισμικού βελτιστοποίησης, για τον προσδιορισμό της πιο αποδοτικής διαδρομής για τις παραδόσεις των φορτηγών. Για να διευκολυνθεί η εφαρμογή της λύσης στον πραγματικό κόσμο, αναπτύχθηκε κώδικας στη γλώσσα προγραμματισμού Python, ο οποίος ενσωματώνει το Google Distance Matrix API για τη δημιουργία των απαιτούμενων πινάκων απόστασης και διάρκειας από ένα σύνολο διευθύνσεων εισόδου. Στη συνέχεια, το πρόγραμμα επιλύει το TSP, παράγοντας τη βέλτιστη ακολουθία στάσεων. Επιπλέον, η λύση ενσωματώνεται στους χάρτες Google, εμφανίζοντας αυτόματα τη βελτιστοποιημένη διαδρομή σε ένα πρόγραμμα περιήγησης μετά την ολοκλήρωσή της. Η προσέγγιση δοκιμάστηκε σε ένα πραγματικό σενάριο εντός της μεταφορικής εταιρείας, με αποτέλεσμα τη μείωση της συνολικής απόστασης ταξιδιού σε σύγκριση με τις διαδρομές που χρησιμοποιούνταν προηγουμένως βάσει εμπειρίας. Αυτή η μεθοδολογία καταδεικνύει τα πρακτικά οφέλη του συνδυασμού της μαθηματικής βελτιστοποίησης με σύγχρονα διαδικτυακά εργαλεία για τη βελτίωση της αποδοτικότητας των logistics.

Λέξεις κλειδιά: Βελτιστοποίηση, Δρομολόγηση, Πρόβλημα πλανόδιου πωλητή, Γραμμικός προγραμματισμός, Python

Abstract

This thesis presents a novel approach to optimize the routing of delivery trucks for a logistics company using linear integer programming, aiming to minimize travel distance or duration. The proposed solution involves the application of the Traveling Salesman Problem (TSP) using LINGO, a mathematical optimization software, to determine the most efficient route for truck deliveries. To facilitate real-world implementation, a Python script was developed, integrating the Google Distance Matrix API to generate the required distance and duration matrices from a set of input addresses. The script then solves the TSP, producing the optimal sequence of stops. Furthermore, the solution is seamlessly integrated with Google Maps, automatically displaying the optimized route in a web browser upon completion. The approach was tested in a real-life scenario within the truck company, resulting in a reduction in the total travel distance compared to the previously used routes based on experience. This methodology demonstrates the practical benefits of combining mathematical optimization with modern web-based tools for improving logistics efficiency.

Key words: Optimization, Routing, Travelling salesman problem, Linear programming, Python

Πνευματικά δικαιώματα

Δηλώνω ρητά ότι η παρούσα Διπλωματική Εργασία με τίτλο: «Βελτιστοποίηση δρομολόγησης στόλου διανομής σε μεταφορική εταιρεία», καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στο πλαίσιο αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν και η οποία έχει εκπονηθεί στο Τμήμα Μηχανολόγων Μηχανικών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του κ.. Γιώργου Νενέ, αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright © Ανδρέας Πυλαρινός Μαρκαντωνάτος & Γιώργος Νενές, Οκτώβριος 2024, Κοζάνη

Ευχαριστίες

Καταρχάς, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον καθηγητή μου, κ. Γιώργο Νενέ, για την πολύτιμη καθοδήγησή του καθ' όλη τη διάρκεια της διπλωματικής μου εργασίας αλλά και των σπουδών μου. Τα μαθήματά του και ο τρόπος που τα προσέγγισε με ενέπνευσαν να ακολουθήσω την κατεύθυνση της Βιομηχανικής Διοίκησης, διαμορφώνοντας το ακαδημαϊκό μου ενδιαφέρον και καθοδηγώντας με στις μετέπειτα επιλογές μου.

Θα ήθελα στη συνέχεια να εκφράσω την ευγνωμοσύνη μου προς τους γονείς μου, Αθηνά και Χαράλαμπο, για την αμέριστη υποστήριξή τους και την πίστη τους σε εμένα. Η αγάπη και οι θυσίες τους αποτέλεσαν τον θεμέλιο λίθο της προσπάθειάς μου, δίνοντάς μου τη δύναμη να ξεπεράσω κάθε δυσκολία. Το να είναι περήφανοι για μένα θα αποτελεί πάντα ένα από τα ισχυρότερά μου κίνητρα.

Τέλος, ευχαριστώ από καρδιάς τους φίλους μου, οι οποίοι με στήριξαν και με ενθάρρυναν σε κάθε μου βήμα. Ιδιαίτερη αναφορά θα ήθελα να κάνω στον Μπάμπη και την Έλενα που με τις γνώσεις τους συνέβαλαν στην υλοποίηση αυτής της εργασίας, αλλά και στον Σωτήρη με τον οποίο πορευτήκαμε παρέα από την πρώτη μέρα.

Περιεχόμενα

Περίληψη i
Abstract ii
Πνευματικά δικαιώματα iii
Ευχαριστίες iv
Κατάλογος εικόνων vii
Κατάλογος πλαισίων viii
Εισαγωγή 1
1. Βιβλιογραφική επισκόπηση 3
2. Περιγραφή του προβλήματος 6
2.1 Βασικές έννοιες 6
2.2 Περιγραφή εταιρείας 8
2.3 Σκοπός της εργασίας 9
3 Το μοντέλο 10
3.1 Μεταβλητές, σταθερές και δείκτες 10
3.2 Αντικειμενική συνάρτηση και περιορισμοί 11
4 Επίλυση στο Lingo 13
5 Το πρόγραμμα 15
5.1 Γενική περιγραφή 15
5.2 Βιβλιοθήκες που χρησιμοποιήθηκαν 16
5.3 Εισαγωγή δεδομένων 19
5.4 Εξαγωγή δεδομένων από την Google 21
5.5 Δημιουργία πινάκων 22
5.6 Χρήση του Lingo 28
5.7 Επεξεργασία αποτελεσμάτων 30
5.8 Εισαγωγή τελικής διαδρομής στο Google Maps 32
6 Πρακτική εφαρμογή του προγράμματος 35
6.1 Περιορισμοί του προγράμματος 35
6.2 Προσέγγιση της εταιρείας 36
6.3 Εφαρμογή του προγράμματος 39
7 Συμπεράσματα 41

8 Παραρτήματα 42
8.1 Κώδικας LINGO 42
8.2 Κώδικας Python 43
Βιβλιογραφία 48

Κατάλογος εικόνων

Εικόνα 1: Tkinter box για εισαγωγή αριθμού διευθύνσεων και επιλογή πίνακα 20
Εικόνα 2: Tkinter box για εισαγωγή διευθύνσεων 20
Εικόνα 3: Πλαίσιο διαλόγου για εισαγωγή διευθύνσεων στο Google maps..... 33
Εικόνα 4: Χάρτης Κεφαλονιάς χωρισμένος σε διαμερίσματα σύμφωνα με την εταιρεία 36
Εικόνα 5: Σημεία στο χάρτη που καλείται να επισκεφθεί το φορτηγό 37
Εικόνα 6: Διαδρομή που ακολούθησε το φορτηγό βάσει εμπειρίας..... 38
Εικόνα 7:Βέλτιστη διαδρομή όπως προέκυψε από το πρόγραμμα..... 39

Κατάλογος πλαισίων

Πλαίσιο 1: Ψευδοκώδικας για τον ορισμό των SETS στο Lingo 13
Πλαίσιο 2: Ψευδοκώδικας για τον ορισμό των DATA στο Lingo..... 14
Πλαίσιο 3: Ψευδοκώδικας για των ορισμό αντικειμενικής συνάρτησης και περιορισμών στο Lingo 14
Πλαίσιο 4: Ψευδοκώδικας για τη δημιουργία Tkinter box για εισαγωγή αριθμού διευθύνσεων και επιλογή πίνακα 19
Πλαίσιο 5: Ψευδοκώδικας για εισαγωγή και αποθήκευση διευθύνσεων..... 20
Πλαίσιο 6: Ψευδοκώδικας για άντληση δεδομένων από το API 22
Πλαίσιο 7: Δεδομένα που ελήφθησαν από το API σε μορφή JSON..... 23
Πλαίσιο 8: Πίνακας διαρκειών ταξιδιού για το παράδειγμα των πόλεων Α,Β,Γ..... 25
Πλαίσιο 9: Πίνακας αποστάσεων για το παράδειγμα των πόλεων Α,Β,Γ 25
Πλαίσιο 10: Ψευδοκώδικας για την αποθήκευση των πινάκων ως αρχεία κειμένου..... 27
Πλαίσιο 11: Ψευδοκώδικας για τη δημιουργία κώδικα Lingo 29
Πλαίσιο 12: Ψευδοκώδικας για άνοιγμα και χρησιμοποίηση του Lingo..... 30
Πλαίσιο 13: Ψευδοκώδικας για αφαίρεση δεκαδικών από αποτελέσματα του Lingo ... 31
Πλαίσιο 14: Ψευδοκώδικας για επανατοποθέτηση και εκτύπωση διευθύνσεων στη σωστή σειρά..... 32
Πλαίσιο 15: Ψευδοκώδικας για εισαγωγή διευθύνσεων στο Google Maps 34
Πλαίσιο 16: Κώδικας LINGO 42
Πλαίσιο 17: Κώδικας Python 47

Εισαγωγή

Στη σημερινή ταχέως εξελισσόμενη και άκρως ανταγωνιστική παγκόσμια αγορά, η αποτελεσματικότητα των λειτουργιών εφοδιαστικής αλυσίδας παίζει κρίσιμο ρόλο στην επιτυχία κάθε επιχείρησης που ασχολείται με τη μεταφορά εμπορευμάτων. Οι εταιρείες δέχονται αυξανόμενες πιέσεις για την ελαχιστοποίηση του κόστους, εξασφαλίζοντας παράλληλα έγκαιρες παραδόσεις, καθώς οι προσδοκίες των πελατών για ταχεία εκπλήρωση των απαιτήσεων συνεχίζουν να αυξάνονται. Μία από τις σημαντικότερες προκλήσεις που αντιμετωπίζουν οι εταιρείες μεταφορών είναι η βελτιστοποίηση της δρομολόγησης του στόλου διανομής, όπου στόχος είναι η ελαχιστοποίηση της διανυόμενης απόστασης και, κατά συνέπεια, της κατανάλωσης καυσίμων, των χρόνων παράδοσης και του συνολικού λειτουργικού κόστους.

Το πρόβλημα της δρομολόγησης του στόλου αποτελεί αντικείμενο ενδιαφέροντος εδώ και αρκετές δεκαετίες, με τις ρίζες του να ανάγονται στο κλασικό πρόβλημα του πλανόδιου πωλητή (Travelling Salesman Problem - TSP). Το TSP, που διατυπώθηκε για πρώτη φορά στη δεκαετία του 1930 [1], επιδιώκει να προσδιορίσει τη συντομότερη δυνατή διαδρομή για έναν πωλητή που πρέπει να επισκεφθεί ένα σύνολο πόλεων και να επιστρέψει στην πόλη προέλευσης. Αν και εννοιολογικά απλό, το TSP είναι ένα NP-hard πρόβλημα, που σημαίνει ότι η εύρεση ακριβούς λύσης καθίσταται υπολογιστικά ανέφικτη όσο αυξάνεται ο αριθμός των πόλεων. Αυτή η πολυπλοκότητα έχει οδηγήσει στην ανάπτυξη διάφορων ευρετικών και ακριβών αλγορίθμων για την εύρεση βέλτιστων ή σχεδόν βέλτιστων λύσεων σε εύλογο χρονικό διάστημα.

Στο πλαίσιο της εφοδιαστικής, το TSP είναι ένα θεμελιώδες μοντέλο που έχει επεκταθεί σε πιο σύνθετες παραλλαγές, όπως το πρόβλημα δρομολόγησης οχημάτων (VRP), το οποίο λαμβάνει υπόψη πολλαπλά οχήματα, ποικίλες απαιτήσεις πελατών, χρονικά παράθυρα και άλλους περιορισμούς του πραγματικού κόσμου. Η βελτιστοποίηση της δρομολόγησης του στόλου με τη χρήση αυτών των μοντέλων μπορεί να μειώσει σημαντικά τη συνολική διανυόμενη απόσταση, να μειώσει την κατανάλωση καυσίμων και να βελτιώσει τους χρόνους παράδοσης - παράγοντες που επηρεάζουν άμεσα το τελικό αποτέλεσμα και το περιβαλλοντικό αποτύπωμα μιας εταιρείας.

Τα τελευταία χρόνια, η πρόοδος στην υπολογιστική ισχύ και η διαθεσιμότητα μεγάλων συνόλων δεδομένων επέτρεψαν την ανάπτυξη πιο εξελιγμένων τεχνικών βελτιστοποίησης. Μια τέτοια τεχνική είναι η χρήση εξειδικευμένου λογισμικού όπως το LINGO, ένα ισχυρό εργαλείο για την επίλυση γραμμικών, μη γραμμικών και ακέραιων προβλημάτων βελτιστοποίησης. Με τη δημιουργία ενός κώδικα LINGO για την επίλυση του TSP, οι ερευνητές και οι επαγγελματίες μπορούν να διερευνήσουν διάφορα σενάρια και περιορισμούς που σχετίζονται με επιχειρήσεις εφοδιαστικής αλυσίδας.

Για να ενισχυθεί η πρακτικότητα αυτών των μοντέλων, οι σύγχρονες προσπάθειες βελτιστοποίησης συχνά ενσωματώνουν δεδομένα από τον πραγματικό κόσμο μέσω εργαλείων όπως οι χάρτες Google. Το Google Maps προσφέρει λεπτομερείς και ενημερωμένες πληροφορίες σχετικά με τις αποστάσεις, τους χρόνους ταξιδιού και τις συνθήκες κυκλοφορίας μεταξύ διάφορων τοποθεσιών. Αυτή η πλούσια πηγή δεδομένων χρησιμοποιείται ευρέως στις επιχειρήσεις σήμερα για τη βελτίωση της εφοδιαστικής αλυσίδας και του σχεδιασμού διαδρομών. Αξιοποιώντας τους Χάρτες Google, οι εταιρείες μπορούν να δημιουργήσουν ακριβή και αποτελεσματικά σχέδια δρομολόγησης που λαμβάνουν υπόψη τις συνεχώς μεταβαλλόμενες συνθήκες των οδικών δικτύων.

Η παρούσα εργασία επικεντρώνεται στη βελτιστοποίηση της δρομολόγησης του στόλου διανομής σε μια εταιρεία μεταφορών με την αξιοποίηση αυτών των προηγμένων εργαλείων και τεχνικών. Ο πρωταρχικός στόχος είναι η ελαχιστοποίηση της απόστασης που διανύουν τα φορτηγά διανομής, μειώνοντας έτσι το λειτουργικό κόστος και βελτιώνοντας τη συνολική αποδοτικότητα. Σκοπός της εργασίας είναι όχι μόνο να συμβάλλει στην ακαδημαϊκή κατανόηση της βελτιστοποίησης της εφοδιαστικής αλυσίδας, αλλά να προσφέρει επίσης πρακτικές λύσεις που μπορούν εύκολα να εφαρμοστούν σε σενάρια μεταφορών στον πραγματικό κόσμο.

1. Βιβλιογραφική επισκόπηση

Για την πραγματοποίηση της παρούσας εργασίας, προηγήθηκε εκτενής βιβλιογραφική επισκόπηση στο θέμα του προβλήματος του πλανόδιου πωλητή (Traveling Salesman Problem). Η επισκόπηση αυτή επικεντρώθηκε στην ιστορική εξέλιξη του προβλήματος, τις διάφορες μεθόδους και αλγορίθμους που έχουν αναπτυχθεί για την επίλυσή του, καθώς και στις σημαντικές συνεισφορές που έχουν γίνει από την επιστημονική κοινότητα στο πεδίο αυτό.

Παρότι η ακριβής προέλευση του προβλήματος του πλανόδιου πωλητή είναι κάπως ασαφής μία από τις πρώτες αναφορές στη βιβλιογραφία έγινε από τον Karl Menger το 1930. Στο άρθρο του «Beiträge zur Theorie der einfach zusammenhängenden räumlichen Systeme» ή «Contributions to the Theory of Simply Connected Spatial Systems» πραγματεύτηκε το πρόβλημα της εύρεσης της συντομότερης δυνατής διαδρομής που συνδέει ένα σύνολο σημείων (πόλεων) χωρίς να επισκεφτεί κάποιο από αυτά δεύτερη φορά, το οποίο είναι ουσιαστικά το TSP. [1] Αν και η ορολογία έχει εξελιχθεί από την αρχική έκθεση του Menger, το έργο του αναγνωρίζεται ευρέως ως μία από τις πρώτες επιστημονικές διατυπώσεις του προβλήματος.

Οι Dantzig, Fulkerson και Johnson στο άρθρο τους «Solution of a Large-Scale Traveling-Salesman Problem» του 1954 εισήγαγαν τη μέθοδο cutting-plane για την επίλυση του προβλήματος του πλανόδιου πωλητή (TSP).Η μέθοδος αυτή προσεγγίζει προβλήματα βελτιστοποίησης επιλύοντας πρώτα μια εκδοχή του προβλήματος ως γραμμικό πρόγραμμα (LP) χωρίς περιορισμούς ακεραιότητας. Όταν η λύση είναι κλασματική ή άκυρη, προστίθενται πρόσθετοι περιορισμοί για να αποκλειστούν αυτές οι άκυρες λύσεις ενώ διατηρούνται οι έγκυρες. Στη συνέχεια, το πρόγραμμα επιλύεται εκ νέου με αυτούς τους νέους περιορισμούς. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να βρεθεί μια βέλτιστη ακέραια λύση. Οι συγγραφείς εφάρμοσαν τη μέθοδό τους για την επιτυχή επίλυση ενός TSP 49 πόλεων, αποδεικνύοντας την αποτελεσματικότητά της. [2]

Ο Kruskal (1956) παρουσιάζει ένα θεώρημα που δηλώνει ότι σε ένα συνδεδεμένο γράφημα με διακριτά μήκη ακμών το δέντρο που συνδέει όλες τις κορυφές με το ελάχιστο συνολικό μήκος ακμών είναι μοναδικό. Η εργασία του εισάγει απλούστερες μεθόδους για την κατασκευή αυτού του ελάχιστου δέντρου, συμπεριλαμβανομένου του «greedy» algorithm που επιλέγει επανειλημμένα τη συντομότερη ακμή που δεν σχηματίζει βρόχο μέχρι να συνδεθούν όλες οι κορυφές. Επιπλέον, διερευνά τη σχέση μεταξύ αυτών των δέντρων διάσχισης και του προβλήματος του πλανόδιου πωλητή (TSP). [3]

Το 1960 οι Miller, Tucker, και Zemlin στο άρθρο τους με τίτλο «Integer Programming Formulation of Traveling Salesman Problems» προτείνουν μια προσέγγιση ακέραιου γραμμικού προγραμματισμού για τη μοντελοποίηση του TSP. Η διατύπωσή τους περιλαμβάνει καινοτόμους περιορισμούς για την αποφυγή υποδιαδρομών και θέτει τις

βάσεις για την ανάπτυξη πιο προηγμένων αλγορίθμων και μεθόδων για την επίλυση σύνθετων προβλημάτων δρομολόγησης. [4]

Ο Chisman στο άρθρο του με τίτλο «The Clustered Traveling Salesman Problem» που δημοσιεύθηκε το 1975 πραγματεύεται την υποπερίπτωση του TSP όπου ο «πωλητής» πρέπει όχι μόνο να επισκεφθεί κάθε πόλη μία και μόνο μία φορά, αλλά υπάρχουν υποσύνολα (συστάδες) εντός αυτών των πόλεων που πρέπει να τα επισκεφθεί συνεχόμενα. Η μέθοδος του Chisman περιλαμβάνει την τροποποίηση του πίνακα αποστάσεων ώστε να ευνοούνται οι συνδέσεις εντός της συστάδας, την επίλυση του τροποποιημένου προβλήματος με branch-and-bound και, στη συνέχεια, την αξιολόγηση της λύσης με βάση τις αρχικές αποστάσεις. Για μικρότερα προβλήματα, μπορούν να χρησιμοποιηθούν πρόσθετοι περιορισμοί στα ήδη υπάρχοντα μοντέλα ακέραιου γραμμικού προγραμματισμού. [5]

Βασισμένοι στη δουλειά των Miller, Tucker, Zemlin καθώς και των Dantzig, Fulkerson, Johnson οι Gavish και Graves δημοσίευσαν το 1978 το άρθρο τους με τίτλο «The Traveling Salesman Problem and Related Problems» στο οποίο παρουσιάζουν νέες διατυπώσεις γραμμικού ακέραιου προγραμματισμού για το πρόβλημα του πλανόδιου πωλητή (TSP) και διερευνούν τη σχέση τους με προηγούμενες διατυπώσεις. Αυτές οι νέες διατυπώσεις επεκτείνονται για να αντιμετωπίσουν διάφορα προβλήματα προγραμματισμού μεταφορών, όπως το Multi-Travelling Salesman Problem, το Delivery Problem, το School Bus Problem και το Dial-a-Bus Problem. [6]

Με το Multi-Travelling Salesman Problem ασχολήθηκαν και οι Laporte, Nobert στο άρθρο τους «A Cutting Planes Algorithm for the m-Salesmen Problem» που δημοσιεύθηκε το 1980. Σε αυτό εισάγουν έναν νέο cutting planes αλγόριθμο για την επίλυση του m-TSP, χαλαρώνοντας τους περισσότερους περιορισμούς κατά τη διαδικασία επίλυσης. Αναπτύσσουν δύο αλγοριθμικές παραλλαγές: τον ευθύ αλγόριθμο, ο οποίος καθυστερεί την εισαγωγή των περιορισμών εξάλειψης των υποδιαδρομών μέχρι να επιτευχθεί μια ακέραια λύση, και τον αντίστροφο αλγόριθμο, ο οποίος ελέγχει για υποδιαδρομές νωρίτερα στη διαδικασία. [7] Η προσέγγισή τους διαφέρει από τις προηγούμενες μεθόδους, καθώς δεν μετατρέπουν το πρόβλημα σε ένα μεγαλύτερο 1-TSP, αλλά αντιμετωπίζουν απευθείας το m-TSP, γεγονός που αυξάνει την αποδοτικότητα.

Στα βήματα των Laporte και Nobert οι Gavish και Srikanth καταπιάνονται κι αυτοί με το Multi-Travelling Salesman Problem. Η εργασία τους με τίτλο «An optimal solution method for large scale Multiple Traveling Salesmen Problems» του 1986 παρουσιάζει μια μέθοδο βέλτιστης επίλυσης για προβλήματα πολλαπλών πλανόδιων πωλητών μεγάλης κλίμακας. Οι συγγραφείς αναπτύσσουν έναν αλγόριθμο branch-and-bound που χειρίζεται αποτελεσματικά μεγάλες περιπτώσεις του MTSP. Εισάγουν επίσης έναν τρόπο μείωσης του μεγέθους του προβλήματος μέσω βημάτων προεπεξεργασίας που εξαλείφουν τις περιττές μεταβλητές και περιορισμούς. [8] Ο αλγόριθμος δοκιμάζεται σε προβλήματα μεγάλης κλίμακας, αποδεικνύοντας την ικανότητά του να βρίσκει βέλτιστες λύσεις εκεί όπου οι προηγούμενες μέθοδοι δυσκολεύονταν, βελτιώνοντας σημαντικά την υπολογιστική απόδοση και την επεκτασιμότητα.

Στην πρωτοποριακή εργασία τους «Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem» του 1997, οι Dorigo και Gambardella εισήγαγαν το Σύστημα Αποικίας Μυρμηγκιών (Ant Colony System), μια νέα προσέγγιση για την επίλυση του TSP, εμπνευσμένη από τη συμπεριφορά των πραγματικών μυρμηγκιών κατά την αναζήτησης τροφής. Το ACS αξιοποιεί έναν συνεργατικό μηχανισμό μάθησης όπου τα τεχνητά μυρμήγκια δημιουργούν λύσεις ακολουθώντας ίχνη φερομόνης τα οποία ενημερώνονται με βάση την ποιότητα των λύσεων. Αυτή η επαναληπτική διαδικασία επιτρέπει στον αλγόριθμο να εξερευνά αποτελεσματικά το χώρο λύσεων και να συγκλίνει σε λύσεις υψηλής ποιότητας. [9] Η καινοτομία του ACS έγκειται στην εφαρμογή της βελτιστοποίησης που εμπνέεται από τη φύση.

Τέλος, οι Applegate , Bixby , Chvatal και Cook στο άρθρο τους με τίτλο «TSP Cuts Which Do Not Conform to the Template Paradigm» που παρουσιάστηκε το 2001 εισάγουν τη μέθοδο Concorde η οποία επεκτείνει με καινοτόμο τρόπο το πεδίο εφαρμογής των cutting plane αλγορίθμων που χρησιμοποιούνταν στα προγράμματα επίλυσης TSP, συμβάλλοντας σε σημαντικές βελτιώσεις στην επίλυση περιπτώσεων μεγάλης κλίμακας του προβλήματος. [10] Θεωρείται ευρέως ως ο ταχύτερος επιλυτής TSP, για μεγάλες περιπτώσεις, που υπάρχει σήμερα [11].

2. Περιγραφή του προβλήματος

Στο παρόν κεφάλαιο περιγράφεται το πρόβλημα που αναλύεται στην εργασία. Στην αρχή, θα παρατεθούν οι βασικές έννοιες και ορισμοί που είναι απαραίτητοι για την πλήρη κατανόησή του. Στη συνέχεια, θα ακολουθήσει μια αναλυτική περιγραφή της εταιρείας στην οποία πραγματοποιήθηκε η εργασία. Τέλος, θα αναλυθεί ο σκοπός της εργασίας, ο οποίος περιλαμβάνει την επίλυση του προβλήματος και την επίτευξη των στόχων που τέθηκαν πρωταρχικά.

2.1 Βασικές έννοιες

Γραμμικός προγραμματισμός: Ο γραμμικός προγραμματισμός αποτελεί μία μαθηματική τεχνική μοντελοποίησης στην οποία μια γραμμική συνάρτηση μεγιστοποιείται ή ελαχιστοποιείται όταν υπόκειται σε διάφορους, επίσης γραμμικούς, περιορισμούς. Η τεχνική αυτή έχει φανεί χρήσιμη για την καθοδήγηση ποσοτικών αποφάσεων στον επιχειρηματικό σχεδιασμό, στη βιομηχανική διοίκηση και -σε μικρότερο βαθμό- στις κοινωνικές και φυσικές επιστήμες.

Η λύση ενός προβλήματος γραμμικού προγραμματισμού ανάγεται στην εύρεση της βέλτιστης τιμής (μεγαλύτερης ή μικρότερης, ανάλογα με το πρόβλημα) της γραμμικής έκφρασης (που ονομάζεται αντικειμενική συνάρτηση).

$$f = c_1 x_1 + \cdots + c_n x_n$$

Η οποία υπάγεται σε ένα σύνολο περιορισμών που εκφράζονται ως ανισότητες:

$$a_{11} x_1 + \cdots + a_{1n} x_n \leq b_1$$

$$\vdots$$

$$a_{m1} x_1 + \cdots + a_{mn} x_n \leq b_m$$

Τα **a**, **b** και **c** είναι σταθερές που καθορίζονται από τις δυνατότητες, τις ανάγκες, το κόστος, τα κέρδη και άλλες απαιτήσεις και περιορισμούς του προβλήματος. Η βασική παραδοχή κατά την εφαρμογή αυτής της μεθόδου είναι ότι οι διάφορες σχέσεις μεταξύ ζήτησης και διαθεσιμότητας είναι γραμμικές, δηλαδή καμία από τις x_i δεν ανυψώνεται σε δύναμη άλλη από το 1. Για να προκύψει η λύση του προβλήματος αυτού, πρέπει να βρεθεί η λύση του συστήματος γραμμικών ανισοτήτων (δηλαδή το σύνολο των n τιμών των μεταβλητών x_i που ικανοποιεί ταυτόχρονα όλες τις ανισότητες). Στη συνέχεια, η αντικειμενική συνάρτηση υπολογίζεται αντικαθιστώντας τις τιμές των x_i στην εξίσωση που ορίζει την f. [12]

Πρόβλημα πλανόδιου πωλητή (Traveling Salesman Problem ή TSP): Το πρόβλημα του πλανόδιου πωλητή (TSP) είναι ένα κλασικό πρόβλημα βελτιστοποίησης στον τομέα της επιχειρησιακής έρευνας και της επιστήμης των υπολογιστών. Περιλαμβάνει την εύρεση της συντομότερης δυνατής διαδρομής που επιτρέπει σε έναν πωλητή να επισκεφθεί ένα δεδομένο σύνολο πόλεων ακριβώς μία φορά και στη συνέχεια να επιστρέψει στην αρχική πόλη. Ο στόχος είναι η ελαχιστοποίηση της συνολικής απόστασης (ή του κόστους) που διανύει. Το TSP είναι ένα NP-hard πρόβλημα, που σημαίνει ότι καθώς αυξάνεται ο αριθμός των πόλεων, η εύρεση της ακριβούς λύσης γίνεται όλο και πιο δύσκολη και υπολογιστικά απαιτητική. Παρά την απλότητα της διατύπωσής του, το TSP έχει πολυάριθμες πρακτικές εφαρμογές, μεταξύ άλλων στα logistics, την κατασκευή και το σχεδιασμό κυκλωμάτων.

Χάρτες Google (Google maps): Οι Χάρτες Google είναι μια διαδικτυακή υπηρεσία χαρτογράφησης που αναπτύχθηκε από την Google και παρέχει στους χρήστες λεπτομερείς γεωγραφικές πληροφορίες και δυνατότητες πλοήγησης. Προσφέρει ποικιλία υπηρεσιών, όπως απεικόνιση χαρτών, εικόνες σε επίπεδο δρόμου (Street View), ενημερώσεις κυκλοφορίας σε πραγματικό χρόνο, σχεδιασμό διαδρομών για διάφορους τρόπους μεταφοράς (όπως οδήγηση, περπάτημα, ποδηλασία και δημόσια συγκοινωνία) και αναζήτηση τοπικών επιχειρήσεων. Η πλατφόρμα είναι προσβάσιμη μέσω ιστού και εφαρμογών για κινητά, καθιστώντας την ένα ευέλικτο εργαλείο για προσωπική και επαγγελματική χρήση. [13]

API (Application Programming Interface): Οι διεπαφές προγραμματισμού εφαρμογών (API) είναι σύνολα κανόνων και πρωτοκόλλων που επιτρέπουν σε διαφορετικές εφαρμογές λογισμικού να επικοινωνούν μεταξύ τους. Ένα API ορίζει τις μεθόδους και τις μορφές δεδομένων που μπορούν να χρησιμοποιούν οι προγραμματιστές για να αλληλεπιδρούν με εξωτερικά στοιχεία λογισμικού, λειτουργικά συστήματα ή υπηρεσίες. Ουσιαστικά, τα API ενεργούν ως μεσάζοντες που επιτρέπουν στις εφαρμογές να ζητούν και να ανταλλάσσουν δεδομένα ή λειτουργίες, όπως η ανάκτηση πληροφοριών από έναν διακομιστή, η πρόσβαση σε μια βάση δεδομένων ή η χρήση μιας συγκεκριμένης λειτουργίας ενός λειτουργικού συστήματος. Τα APIs είναι θεμελιώδους σημασίας για τη σύγχρονη ανάπτυξη λογισμικού αφού επιτρέπουν την ενσωμάτωση διάφορων υπηρεσιών καθώς και την ενίσχυση της λειτουργικότητας μεταξύ διαφορετικών πλατφορμών. [14]

Python: Η Python είναι μια γλώσσα προγραμματισμού, γνωστή για την απλότητα, την αναγνωσιμότητα και την ευελιξία της. Δημιουργήθηκε από τον Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991. Η σχεδιαστική φιλοσοφία της Python δίνει έμφαση στην αναγνωσιμότητα του κώδικα, με ένα συντακτικό που επιτρέπει στους προγραμματιστές να εκφράζουν έννοιες σε λιγότερες γραμμές κώδικα σε σύγκριση με άλλες γλώσσες όπως η Java ή η C++. Η Python χρησιμοποιείται ευρέως για μια ποικιλία

εφαρμογών, συμπεριλαμβανομένης της ανάπτυξης ιστοσελίδων, της ανάλυσης δεδομένων, της τεχνητής νοημοσύνης, της μηχανικής μάθησης, της αυτοματοποίησης και του επιστημονικού υπολογισμού. Η εκτεταμένη κύρια βιβλιοθήκη της και η ενεργή κοινότητά της την έχουν καταστήσει μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον κόσμο. Η Python είναι επίσης γνωστή για τη συμβατότητά της σε πολλαπλές πλατφόρμες, επιτρέποντας στον κώδικα να εκτελείται σε διάφορα λειτουργικά συστήματα με ελάχιστες τροποποιήσεις. [15]

JSON: Το JSON είναι ένα ακρωνύμιο για το JavaScript Object Notation, είναι μια ανοικτή τυποποιημένη μορφή ανταλλαγής δεδομένων, η οποία είναι ελαφριά και βασισμένη σε κείμενο, με δυνατότητα ανάγνωσης από τον άνθρωπο. Είναι μια μορφή δεδομένων ανεξάρτητη από γλώσσα. Υποστηρίζει σχεδόν κάθε είδους γλώσσα προγραμματισμού, πλαίσιο και βιβλιοθήκη. [16]

2.2 Περιγραφή εταιρείας

Η Εταιρεία Μεταφορών Λυκούδης είναι ένας πάροχος υπηρεσιών μεταφορών και logistics που δραστηριοποιείται κυρίως μεταξύ Κεφαλονιάς και Αθήνας. Η εταιρεία απασχολεί 50 άτομα και διαθέτει δύο μεγάλες αποθήκες, μία στην Αθήνα και μία στο Αργοστόλι, με μία επιπλέον μικρότερη αποθήκη στο Ληξούρι.

Η Λυκούδης Μεταφορική ειδικεύεται στη μεταφορά εμπορευμάτων χρησιμοποιώντας ένα στόλο 25 φορτηγών, ο οποίος περιλαμβάνει τόσο κανονικά φορτηγά όσο και ψυγεία. Η εταιρεία προσφέρει ολοκληρωμένες υπηρεσίες, συμπεριλαμβανομένης της μεταφοράς εμπορευμάτων μεταξύ Κεφαλονιάς και Αθήνας καθώς επίσης τη διανομή των εμπορευμάτων αυτών στους τελικούς προορισμούς τους (last mile logistics). Η εταιρεία είναι γνωστή για τις αξιόπιστες και αποτελεσματικές υπηρεσίες logistics, αξιοποιώντας τον καλά συντηρημένο στόλο της και τις στρατηγικά τοποθετημένες αποθήκες της για την κάλυψη των αναγκών των πελατών της.

2.3 Σκοπός της εργασίας

Ο σκοπός της παρούσας εργασίας είναι η βελτιστοποίηση της δρομολόγησης του στόλου διανομής της παραπάνω εταιρείας, με στόχο τη μείωση της απόστασης ή του χρόνου που χρειάζεται κάθε φορτηγό κατά την παράδοση προϊόντων στους πελάτες. Για την επίτευξη αυτού του στόχου, έχει αναπτυχθεί ένας κώδικας στο πρόγραμμα LINGO που επιλύει το πρόβλημα του πλανόδιου πωλητή (Traveling Salesman Problem). Στη συνέχεια, χρησιμοποιώντας το Google Distance Matrix API μέσω ενός Python script, έχουν δημιουργηθεί οι απαραίτητοι πίνακες αποστάσεων που εισάγονται στον κώδικα LINGO. Η μελέτη επικεντρώνεται στην εφαρμογή αυτών των σύγχρονων τεχνικών για την επίτευξη βελτιστοποιημένων δρομολογίων, αναγνωρίζοντας τη σημασία της αποτελεσματικής διαχείρισης των μεταφορών στη σημερινή αγορά και την ανάγκη για την ανάπτυξη και εφαρμογή προηγμένων μεθόδων που μειώνουν τα λειτουργικά κόστη και βελτιώνουν την εξυπηρέτηση των πελατών.

3 Το μοντέλο

Σε αυτό το κεφάλαιο, παρουσιάζεται η μαθηματική διατύπωση του προβλήματος του πλανόδιου πωλητή (TSP). Η διατύπωση που χρησιμοποιείται βασίζεται στην προσέγγιση που προτάθηκε από τους Miller, Tucker και Zemlin [4]. Παρέχεται μια λεπτομερής επεξήγηση των μεταβλητών, καθώς και των περιορισμών που εξασφαλίζουν ότι κάθε πόλη επισκέπτεται ακριβώς μία φορά και ότι η συνολική απόσταση που διανύεται ελαχιστοποιείται.

3.1 Μεταβλητές, σταθερές και δείκτες

Δείκτες:

- i, j: είναι δείκτες που αντιπροσωπεύουν σημεία (ή κόμβους) στο πρόβλημα του πλανόδιου πωλητή. Το i αναφέρεται σε κόμβο αφετηρίας ενώ το j σε κόμβο προορισμού. Μαζί χρησιμοποιούνται για να ορίσουν ζεύγη σημείων στη διαδρομή, όπου ο πωλητής ταξιδεύει από τον κόμβο i στον κόμβο j.

Σταθερές:

- n: Προσδιορίζει τον αριθμό των κόμβων που καλείται το φορτηγό να επισκεφθεί κατά τη διάρκεια μίας συγκεκριμένης διαδρομής.

- d(i, j): Συμβολίζει το κόστος που συνδέεται με το ταξίδι από τον κόμβο i προς τον κόμβο j. Στην παρούσα εφαρμογή το κόστος ορίζεται είτε ως η απόσταση μεταξύ των 2 σημείων είτε ως ο χρόνος που χρειάζεται το φορτηγό προκειμένου να διανύσει αυτή την απόσταση.

Μεταβλητές:

- x(i, j): Δυαδική μεταβλητή απόφασης που προσδιορίζει το αν το φορτηγό θα μεταβεί από τον κόμβο i στον κόμβο j. Εάν πάρει την τιμή 1 η μετάβαση θα πραγματοποιηθεί ενώ αν πάρει την τιμή 0 δεν θα πραγματοποιηθεί.

- U(i ή j): Αποτελεί βοηθητική ακέραιη μεταβλητή. Αντιπροσωπεύει τη σειρά με την οποία το φορτηγό επισκέπτεται τους κόμβους της διαδρομής και χρησιμοποιείται στους περιορισμούς για την αποφυγή υποδεέστερων διαδρομών.

3.2 Αντικειμενική συνάρτηση και περιορισμοί

Αφού ορίστηκαν τα παραπάνω δεδομένα, είναι πλέον εφικτή η διαμόρφωση του μοντέλου ακέραιου γραμμικού προγραμματισμού που περιγράφει το πρόβλημα του πλανόδιου πωλητή. Η αντικειμενική συνάρτηση που πρέπει να ελαχιστοποιηθεί διατυπώνεται ως εξής:

$$min \sum_{i \neq j} d_{ij} x_{ij} \quad (1)$$

Όπου d_{ij} είναι το κόστος της διαδρομής από τον κόμβο i στον κόμβο j και x_{ij} είναι η μεταβλητή απόφασης που καθορίζει το αν θα πραγματοποιηθεί η συγκεκριμένη διαδρομή. Το γινόμενό τους ουσιαστικά αναπαριστά το κόστος των διαδρομών που τελικά πραγματοποιούνται το οποίο και πρέπει να ελαχιστοποιηθεί.

Οι περιορισμοί που διέπουν το πρόβλημα παρουσιάζονται αναλυτικά παρακάτω:

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1,2, \dots, n \quad (2)$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1,2, \dots, n \quad (3)$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i, j \quad (4)$$

$$U_i - U_j + n x_{ij} \leq n - 1 \qquad i, j = 2 \dots n \quad (5)$$

$$1 \leq U_i \leq n-1 \qquad\qquad i = 2 \dots n \qquad (6)$$

- (2): Εξασφαλίζει ότι για κάθε σημείο i, υπάρχει ακριβώς μία εξερχόμενη διαδρομή από το i προς ένα άλλο σημείο j. Μόλις το φορτηγό επισκεφθεί το σημείο i, πρέπει να εγκαταλείψει το σημείο αυτό για να πάει σε επόμενο. Δεν μπορεί να υπάρχουν πολλαπλές έξοδοι από τον κόμβο i, ούτε μπορεί το φορτηγό να παραμείνει στο i επ' αόριστον (επειδή το άθροισμα των μεταβλητών για την έξοδο από το σημείο i πρέπει να ισούται με 1).

- (3): Αυτός ο περιορισμός εξασφαλίζει ότι το φορτηγό «εισέρχεται» σε κάθε σημείο ακριβώς μία φορά. Μαζί με τον πρώτο περιορισμό, διαβεβαιώνεται ότι το φορτηγό εισέρχεται και εξέρχεται από κάθε κόμβο ακριβώς μία φορά, πράγμα απαραίτητο για να σχηματιστεί ένας έγκυρος κύκλος.

- (4): Ο περιορισμός αυτός ορίζει τη δυαδική φύση της μεταβλητής x_{ij}. Οι μόνες αποδεκτές τιμές για τη συγκεκριμένη μεταβλητή είναι το 0 και το 1.

- (5): Με τον περιορισμό αυτό διασφαλίζεται ότι δεν επιτρέπονται υποδιαδρομές (μικρότεροι βρόχοι που επισκέπτονται μόνο ένα υποσύνολο σημείων) στη λύση και ο τρόπος με τον οποίο λειτουργεί είναι ο εξής:
  - Όταν $x_{ij} = 1$ δηλαδή η μετάβαση από το i στο j εμπεριέχεται στην τελική διαδρομή ο περιορισμός γίνεται $U_i - U_j \leq -1$ αναγκάζοντας έτσι τις μεταβλητές U_j να είναι μεγαλύτερες κατά τουλάχιστον 1 από τις μεταβλητές U_i αποφεύγοντας έτσι τις υποδεέστερες διαδρομές.
  - Όταν $x_{ij} = 0$ δηλαδή η μετάβαση από το i στο j δεν εμπεριέχεται στην τελική διαδρομή ο περιορισμός γίνεται $U_i - U_j \leq n-1$ και οι μεταβλητές U δεν περιορίζονται.

  Τελικά ο περιορισμός αυτός εξασφαλίζει ότι οι μεταβλητές U σχηματίζουν μια ακολουθία που «μετράει» τη σειρά με την οποία το φορτηγό επισκέπτεται τους κόμβους, γεγονός που αποτρέπει το σχηματισμό διαδρομών εντός ενός υποσυνόλου πόλεων. Έτσι, η λύση θα είναι μια ενιαία συνεχής διαδρομή που επισκέπτεται όλα τα σημεία ακριβώς μία φορά.

- (6): Ο περιορισμός αυτός περιορίζει τις τιμές που οι βοηθητικές μεταβλητές U μπορούν να λάβουν, εξασφαλίζοντας ότι βρίσκονται μέσα σε ένα συγκεκριμένο εύρος.

4 Επίλυση στο Lingo

Στο παρόν κεφάλαιο επιλύεται το πρόβλημα του πλανόδιου πωλητή με τη χρήση του προγράμματος Lingo, ενός επιλυτή γραμμικού, και όχι μόνο, προγραμματισμού. Παρακάτω παρουσιάζονται αναλυτικά και με τη χρήση ψευδοκώδικα τα βήματα που ακολουθήθηκαν για τη δόμηση αλλά και την επίλυση του προβλήματος.

Αρχικά έγινε ορισμός των SETS του προβλήματος. Τα SETS ορίζονται ως τα κύρια στοιχεία του μοντέλου, τα οποία στην προκειμένη περίπτωση είναι τα σημεία/κόμβοι (που αναφέρονται ως VERTEX) και οι διαδρομές μεταξύ τους (που αναφέρονται ως ARC). Οι κόμβοι του προβλήματος γίνονται ανάθεση στην μεταβλητή U ενώ για τις διαδρομές μεταξύ τους (ARCS) χρησιμοποιούνται οι μεταβλητές d_{ij} (κόστος της διαδρομής από το i στο j) και x_{ij} (δυαδική μεταβλητή απόφασης για το αν θα πραγματοποιηθεί η διαδρομή από το i στο j).

```
SETS:
    VERTEX: Κόμβοι του προβλήματος, χρήση της μεταβλητής U
    ARC(VERTEX, VERTEX): Διαδρομές μεταξύ ζευγών κόμβων, χρήση των
μεταβλητών D, X;

ENDSETS
```

Πλαίσιο 1: Ψευδοκώδικας για τον ορισμό των SETS στο Lingo

Στη συνέχεια ορίστηκαν τα DATA του προβλήματος. Η ενότητα των DATA παρέχει τις πραγματικές αριθμητικές τιμές για το πρόβλημα, όπως ο αριθμός των κόμβων, ο πίνακας των αποστάσεων και ο πίνακας των χρόνων που χρειάζονται για να καλυφθούν οι αποστάσεις. Οι συγκεκριμένες τιμές εισάγονται από το χρήστη μέσω ενός Python script όπως θα παρουσιαστεί αναλυτικά σε επόμενο κεφάλαιο. Επιπλέον, μέσω της εντολής TEXT οι τιμές που παίρνει η μεταβλητή U αφού ολοκληρωθεί η επίλυση του μοντέλου αποθηκεύονται σε ένα εξωτερικό αρχείο για να χρησιμοποιηθούν στη συνέχεια.

```
DATA:
    N = Συνολικός αριθμός κόμβων όπως ορίζεται από το χρήστη;
    VERTEX = Εύρος από 1 έως N (οι πόλεις χαρακτηρίζονται ως {1, 2,
3, 4});
    D =@FILE(Φόρτωση πίνακα αποστάσεων ή χρόνων από εξωτερικό
αρχείο.);
    @TEXT(Εξαγωγή των τιμών της μεταβλητής U (σειρά των επισκέψεων)
και αποθήκευσή τους σε ένα αρχείο)

    ENDDATA
```

Πλαίσιο 2: Ψευδοκώδικας για τον ορισμό των DATA στο Lingo

Αφού ορίστηκαν τα παραπάνω μένει μόνο η εισαγωγή της αντικειμενικής συνάρτησης και των περιορισμών του προβλήματος όπως αυτά έχουν παρουσιαστεί αναλυτικά στο υποκεφάλαιο 4.2.

```
    !Ορισμός αντικειμενικής συνάρτησης;
    MIN = @ΑΘΡΟΙΣΜΑ D(I, J) * X(I, J) για όλες τις διαδρομές μεταξύ των
κόμβων I και J (όπου I ≠ J).);

    !ΠΕΡΙΟΡΙΣΜΟΙ;

    !ΑΦΙΞΕΙΣ;
    @ΓΙΑ ΚΑΘΕ (ΚΟΜΒΟ(I)) : @ΑΘΡΟΙΣΜΑ X(I, J) για όλα τα J ≠ I ισούται
με 1;

    !ΑΝΑΧΩΡΗΣΕΙΣ;
    @ΓΙΑ ΚΑΘΕ (ΚΟΜΒΟ(J)) : @ΑΘΡΟΙΣΜΑ X(I, J) για όλα τα J ≠ I ισούται
με 1;
    !ΔΥΑΔΙΚΗ ΜΕΤΑΒΛΗΤΗ;
    @ΓΙΑ ΚΑΘΕ (ΔΙΑΔΡΟΜΗ(I,J)): Η Μεταβλητή (X(I,J)) είναι δυαδική
(παίρνει τιμές 0 ή 1);

    !ΑΠΟΚΛΕΙΣΜΟΣ ΥΠΟΔΕΕΣΤΕΡΩΝ ΔΙΑΔΡΟΜΩΝ;
    @ΓΙΑ ΚΑΘΕ (ΔΙΑΔΡΟΜΗ(I,J) | όπου I ≠ 1 και J ≠ 1, ισχύει : U(I) -
U(J) + N*X(I,J) <= N-1);
    @ΓΙΑ ΚΑΘΕ (ΚΟΜΒΟ (I) | όπου I ≠ 1, ισχύει: U(I) >= 1);
    @ΓΙΑ ΚΑΘΕ (ΚΟΜΒΟ (I) | όπου I ≠ 1, ισχύει: U(I) <= N-1);
```

Πλαίσιο 3: Ψευδοκώδικας για των ορισμό αντικειμενικής συνάρτησης και περιορισμών στο Lingo

5 Το πρόγραμμα

Για να δοθεί μια πιο πρακτική προσέγγιση στο TSP, δημιουργήθηκε ένα Python script που υλοποιεί τον παραπάνω κώδικα Lingo. Ο κώδικας αυτός αυτοματοποιεί τη διαδικασία δημιουργίας των πινάκων απόστασης /διάρκειας χρησιμοποιώντας το Google Maps API καθώς και την τροφοδότησή τους στο Lingo για βελτιστοποίηση. Συνδυάζοντας την Python και το Lingo, το πρόγραμμα χειρίζεται αποτελεσματικά δεδομένα του πραγματικού κόσμου και εξάγει τη συντομότερη διαδρομή για ένα δεδομένο σύνολο τοποθεσιών. Επιπλέον, η λύση ενσωματώνεται με τους χάρτες Google για να επιτρέπει την εύκολη απεικόνιση και πλοήγηση, καθιστώντας την πιο φιλική προς το χρήστη και εφαρμόσιμη σε πρακτικά σενάρια όπως η δρομολόγηση φορτηγών.

5.1 Γενική περιγραφή

Το συγκεκριμένο πρόγραμμα Python έχει σχεδιαστεί για την επίλυση του προβλήματος του πλανόδιου πωλητή (TSP) χρησιμοποιώντας το Google Maps API για τη δημιουργία πινάκων απόστασης ή διάρκειας διαδρομής, οι οποίοι στη συνέχεια εισάγονται στο Lingo για την εύρεση της βέλτιστης λύσης. Αρχικά, ο χρήστης παρέχει τον αριθμό των διευθύνσεων και τις ίδιες τις διευθύνσεις χρησιμοποιώντας μια γραφική διεπαφή χρήστη (Graphical User Interface) που έχει κατασκευαστεί χρησιμοποιώντας τη βιβλιοθήκη της Python που ονομάζεται Tkinter. Επιπλέον, ο χρήστης έχει τη δυνατότητα να επιλέξει μεταξύ της βελτιστοποίησης της διαδρομής με βάση την απόσταση ή με βάση τη διάρκεια ταξιδιού. Μόλις δοθούν οι διευθύνσεις, το πρόγραμμα τις στέλνει στο API της Google, ανακτώντας τις αποστάσεις ή τις διάρκειες ταξιδιού μεταξύ κάθε ζεύγους τοποθεσιών. Η απάντηση επεξεργάζεται για τη δημιουργία δύο πινάκων: ένας για τις αποστάσεις και ένας για τις διάρκειες. Αυτοί οι πίνακες αποθηκεύονται σε αρχεία κειμένου για μετέπειτα χρήση στο Lingo.

Στη συνέχεια, το πρόγραμμα εισάγει τον κώδικα που αναλύθηκε στο κεφάλαιο 5 στο Lingo, προσαρμοσμένο στον αριθμό των τοποθεσιών, χρησιμοποιώντας τον αποθηκευμένο πίνακα αποστάσεων ή διάρκειας διαδρομών. Το λογισμικό Lingo τρέχει αυτόματα και το μοντέλο TSP επιλύεται χρησιμοποιώντας τον παρεχόμενο πίνακα ως είσοδο (input). Στη συνέχεια, το πρόγραμμα διαβάζει τα αποτελέσματα από το αρχείο εξόδου, εμφανίζοντας τη βέλτιστη ακολουθία των τοποθεσιών ελαχιστοποιώντας την επιθυμητή παράμετρο. Ο χρήστης έχει τη δυνατότητα να εισάγει αυτή τη βελτιστοποιημένη διαδρομή στο Google Maps για εύκολη πλοήγηση. Εάν ο χρήστης επιλέξει να προχωρήσει, οι διευθύνσεις ανοίγουν στο Google Maps και απεικονίζεται η βέλτιστη διαδρομή.

Στα επόμενα υποκεφάλαια παρουσιάζονται αναλυτικά όλα τα επιμέρους τμήματα του κώδικα και γίνεται χρήση ψευδοκώδικα για την ευρύτερη κατανόησή τους.

5.2 Βιβλιοθήκες που χρησιμοποιήθηκαν

Για την ανάπτυξη αυτού του προγράμματος, ενσωματώθηκαν διάφορες βασικές βιβλιοθήκες της Python. Αυτές οι βιβλιοθήκες επέτρεψαν εργασίες όπως ο χειρισμός αρχείων, η επεξεργασία δεδομένων, η αλληλεπίδραση με εξωτερικά APIs, η δημιουργία διεπαφής χρήστη και η αυτοματοποίηση της χρήσης του διαδικτύου. Αξιοποιώντας αυτά τα εργαλεία, το πρόγραμμα μπόρεσε να χειριστεί αποτελεσματικά τις εισόδους, να αυτοματοποιήσει τις διαδικασίες και να παρουσιάσει τα αποτελέσματα με φιλικό προς τον χρήστη τρόπο. Οι βιβλιοθήκες που χρησιμοποιήθηκαν για το συγκεκριμένο πρόγραμμα παρουσιάζονται αναλυτικά παρακάτω:

- **os**:
  - Περιγραφή: Η βιβλιοθήκη os παρέχει λειτουργίες για την αλληλεπίδραση με το λειτουργικό σύστημα του υπολογιστή. Επιτρέπει τη διαχείριση αρχείων και καταλόγων, καθώς και την αλληλεπίδραση με το περιβάλλον.
  - Ενδεικτική χρήση: Διαχείριση φακέλων (π.χ. os.remove() για διαγραφή αρχείων).
- **re**:
  - Περιγραφή: Η βιβλιοθήκη re παρέχει υποστήριξη για την εργασία με Regular expressions στην Python. Αυτές επιτρέπουν την αντιστοίχιση μοτίβων και τον χειρισμό κειμένου.
  - Ενδεικτική χρήση: Αφαίρεση ψηφίων από μια συμβολοσειρά χρησιμοποιώντας την re.findall().
- **time**:
  - Περιγραφή: Η βιβλιοθήκη time παρέχει λειτουργίες που σχετίζονται με τον χρόνο. Αυτό περιλαμβάνει τη λήψη της τρέχουσας ώρας, την παύση του προγράμματος και τη μετατροπή μορφών χρόνου.
  - Ενδεικτική χρήση: Παύση διεργασιών με την time.sleep() και μέτρηση του χρόνου που απαιτείται από τμήματα του κώδικα.
- **tkinter** (tk):
  - Περιγραφή: Η tkinter είναι η τυπική βιβλιοθήκη GUI (Graphical User Interface) της Python. Επιτρέπει στους προγραμματιστές να δημιουργούν απλές εφαρμογές επικοινωνίας με το χρήστη.

- Ενδεικτική χρήση: Δημιουργία παραθύρων, πλαισίων διαλόγου, κουμπιών και άλλων γραφικών στοιχείων. Στη συγκεκριμένη περίπτωση, χρησιμοποιείται για τη δημιουργία διαλόγων για την εισαγωγή δεδομένων από τον χρήστη και την εμφάνιση μηνυμάτων.

- **urllib.parse**:

  - Περιγραφή: Η ενότητα urllib.parse παρέχει συναρτήσεις για το χειρισμό διευθύνσεων URL, συμπεριλαμβανομένης της ανάλυσης, συνένωσης και κωδικοποίησης στοιχείων URL.

  - Ενδεικτική χρήση: Κωδικοποίηση query strings ώστε να είναι ασφαλή για URL χρησιμοποιώντας την urllib.parse.quote_plus().

- **webbrowser**:

  - Περιγραφή: Η βιβλιοθήκη webbrowser παρέχει μια διεπαφή υψηλού επιπέδου για το άνοιγμα διευθύνσεων URL σε ένα πρόγραμμα περιήγησης ιστού.

  - Ενδεικτική χρήση: Άνοιγμα URL σε έναν συγκεκριμένο web browser χρησιμοποιώντας την webbrowser.open().

- **tkinter.simpledialog** και **tkinter.messagebox**:

  - Περιγραφή: Αποτελούν τμήματα της tkinter και χρησιμοποιούνται για τη δημιουργία διαλόγων και message boxes.

  - Ενδεικτική χρήση: Προτροπή του χρήστη για εισαγωγή δεδομένων ή εμφάνιση μηνυμάτων όπως προειδοποιήσεις ή παράθυρα επιβεβαίωσης. Για παράδειγμα, η messagebox.askquestion() χρησιμοποιείται για να ρωτήσει τον χρήστη αν θέλει να εισάγει διευθύνσεις στους Χάρτες Google.

- **pandas** (pd):

  - Περιγραφή: Η pandas είναι μια ισχυρή βιβλιοθήκη ανάλυσης και επεξεργασίας δεδομένων για την Python. Παρέχει δομές δεδομένων όπως τα DataFrames που χρησιμοποιούνται

  - Ενδεικτική χρήση: Χειρισμός δεδομένων σε πίνακες, εκτέλεση μετασχηματισμών και εξαγωγή δεδομένων σε διάφορες μορφές (π.χ. CSV ή Excel).

- **pyautogui**:

  - Περιγραφή: Η pyautogui επιτρέπει την πρόσβαση στο πληκτρολόγιο και το ποντίκι του υπολογιστή. Συχνά χρησιμοποιείται για αυτοματοποίηση διαδικασιών.

  - Ενδεικτική χρήση: Αυτοματοποίηση εργασιών που απαιτούν είσοδο μέσω πληκτρολογίου ή κλικ του ποντικιού. Σε αυτή την περίπτωση, χρησιμοποιείται για την προσομοίωση του πατήματος Ctrl + U για την εκτέλεση του λογισμικού Lingo.

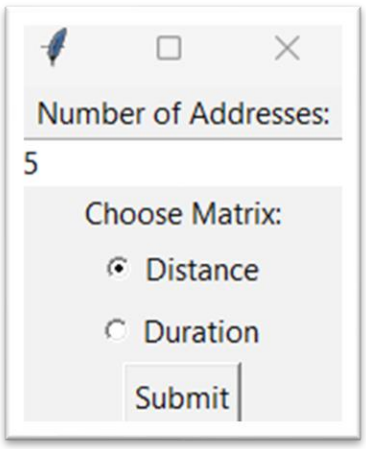
- **requests**:
  - Περιγραφή: Η βιβλιοθήκη requests επιτρέπει τις αιτήσεις σε υπηρεσίες ιστού (HTTP) στην Python. Παρέχει συναρτήσεις για την αποστολή αιτημάτων και το χειρισμό απαντήσεων (π.χ. JSON, XML).
  - Ενδεικτική χρήση: Πραγματοποίηση αιτήσεων GET ή POST σε υπηρεσίες ιστού και API, όπως η ανάκτηση δεδομένων από ένα εξωτερικό API. [17]

5.3 Εισαγωγή δεδομένων

Το πρώτο πράγμα που κάνει το πρόγραμμα είναι να ρυθμίσει τη διεπαφή του Tkinter, επιτρέποντας στον χρήστη να εισάγει βασικά δεδομένα όπως ο αριθμός των διευθύνσεων και να επιλέξει μεταξύ της χρήσης του πίνακα αποστάσεων ή διάρκειας διαδρομών.

Πλαίσιο 4: Ψευδοκώδικας για τη δημιουργία Tkinter box για εισαγωγή αριθμού διευθύνσεων και επιλογή πίνακα

```
# Δημιουργία παραθύρων tkinter
root = tk.Tk() Δημιουργία box και αποθήκευση στην μεταβλητή root
ονομασία.box("TSP Solver")

tk.Label(Προσθήκη ετικέτας "Number of Addresses:")
num_addresses_entry = προσθήκη πλαισίου εισαγωγής
num_addresses_entry.pack() εμφάνιση του πλαισίου εισαγωγής

# Προσθήκη επιλογής ανάμεσα σε πίνακες απόστασης/χρόνου
matrix_choice = tk.StringVar(value="distance_matrix.txt")  # Default to
"distance"

ετικέτα για επιλογή πίνακα(root, text="Choose Matrix:").pack()

εισαγωγή κουμπιού επιλογής για πίνακα απόστασης(root, text="Distance",
variable=matrix_choice, value="distance_matrix.txt").pack()


εισαγωγή κουμπιού επιλογής για πίνακα χρόνου(root, text="Duration",
variable=matrix_choice, value="duration_matrix.txt").pack()



εισαγωγή κουμπιού submit(root, text="Submit",
command=get_api_key_and_num_addresses).pack()
```

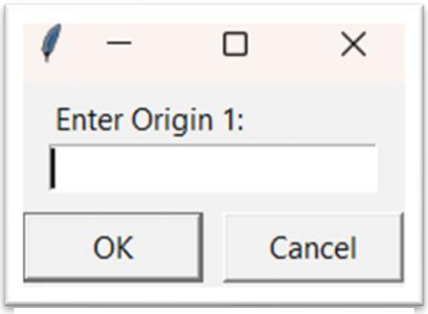


Εικόνα 1: Tkinter box για εισαγωγή αριθμού διευθύνσεων και επιλογή πίνακα

Μόλις ο χρήστης ολοκληρώσει την εισαγωγή αυτών των πληροφοριών, καλείται να ορίσει τις διευθύνσεις που θέλει να συμπεριληφθούν στον υπολογισμό της βέλτιστης διαδρομής. Για να πραγματοποιηθεί αυτό έχουν δημιουργηθεί πλαίσια εισαγωγής όπως αυτό της εικόνας 3, ο αριθμός των οποίων εξαρτάται από τον αριθμό διευθύνσεων που εισήχθη προηγουμένως. Είναι σημαντικό η πρώτη διεύθυνση που εισάγεται να είναι η αφετηρία του προβλήματος.

```
origins = [δημιουργία αριθμημένης λίστας των διευθύνσεων που εισάγει
ο χρήστης ("Input", f"Enter Origin {i + 1}:")for i in
range(num_addresses)]

destinations = origins
```

Πλαίσιο 5: Ψευδοκώδικας για εισαγωγή και αποθήκευση διευθύνσεων



Εικόνα 2: Tkinter box για εισαγωγή διευθύνσεων

Αφού ολοκληρωθεί η εισαγωγή αυτών των πληροφοριών, αποθηκεύονται σε συγκεκριμένες μεταβλητές για μετέπειτα χρήση στο πρόγραμμα. Η μεταβλητή num_addresses καταγράφει τον αριθμό των διευθύνσεων που παρέχει ο χρήστης. Η μεταβλητή matrix_choice_value αποθηκεύει την επιλογή του χρήστη μεταξύ του πίνακα απόστασης ή του πίνακα διάρκειας, που καθορίζεται από τα κουμπιά επιλογής. Τέλος, η μεταβλητή origins συγκρατεί τον κατάλογο των διευθύνσεων που εισάγει ο χρήστης, οι οποίες χρησιμοποιούνται τόσο ως αφετηρία όσο και ως προορισμός για τον υπολογισμό της απόστασης ή του χρόνου και την επίλυση του Προβλήματος Πλανόδιου Πωλητή (TSP). Αυτές οι μεταβλητές είναι ζωτικής σημασίας, καθώς περνούν σε επόμενες συναρτήσεις για να ζητηθούν δεδομένα από το Google Maps API και να δημιουργηθούν οι πίνακες που απαιτούνται για τη βελτιστοποίηση της τελικής διαδρομής.

5.4 Εξαγωγή δεδομένων από την Google

Αφού έχουν πλέον οριστεί ο αριθμός των διευθύνσεων και οι ίδιες οι διευθύνσεις, το πρόγραμμα χρησιμοποιεί το Distance Matrix API της Google με σκοπό να αποκτήσει τα απαραίτητα δεδομένα για την δημιουργία των πινάκων απόστασης και χρόνου μεταξύ των σημείων.

Αυτό επιτυγχάνεται μέσω της συνάρτησης get_distance_matrix. Η συνάρτηση λαμβάνει ως ορίσματα ένα κλειδί API (δηλαδή έναν κωδικό που αποτελεί την άδεια χρήσης για το API), μια λίστα με αφετηρίες (σημεία εκκίνησης), προορισμούς (σημεία τερματισμού) και τον τρόπο ταξιδιού, ο οποίος έχει ως προεπιλογή την οδήγηση. Οι αφετηρίες και οι προορισμοί (οι οποίοι ταυτίζονται και τους έχει εισάγει ο χρήστης προηγουμένως) μορφοποιούνται σε μια συμβολοσειρά που μπορεί να καταλάβει το API. Στη συνέχεια, η συνάρτηση στέλνει ένα αίτημα στο API με αυτές τις λεπτομέρειες. Εάν η αίτηση είναι επιτυχής, επιστρέφει την απάντηση σε μορφή JSON η οποία περιέχει τις αποστάσεις και τις διάρκειες μεταξύ όλων των σημείων. Εάν η αίτηση αποτύχει, εμφανίζει ένα μήνυμα σφάλματος που εξηγεί το πρόβλημα.

Πλαίσιο 6: Ψευδοκώδικας για άντληση δεδομένων από το API

```
συνάρτηση get_distance_matrix(api_key, origins, destinations,
mode="driving"):
    url = "https://maps.googleapis.com/maps/api/distancematrix/json"

    origins_str = Μετατροπή λίστας προελεύσεων σε μία ενιαία
συμβολοσειρά χωρισμένη με '|'

    destinations_str = Μετατροπή λίστας προορισμών σε μία ενιαία
συμβολοσειρά χωρισμένη με '|'

    params = Δημιουργία παραμέτρων που χρειάζεται ως εισαγωγή το API
        'origins': origins_str,
        'destinations': destinations_str,
        'mode': mode,
        'key': api_key


    response = Πραγματοποίηση αίτησης στο API με τη διεύθυνση URL και
τις παραμέτρους

    if η αίτηση είναι επιτυχής (κωδικός κατάστασης 200):
        return την απάντηση σε μορφή JSON
    else:
        raise Δημιουργία σφάλματος με τον κωδικό κατάστασης και το
μήνυμα σφάλματος
```

5.5 Δημιουργία πινάκων

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο το API επιστρέφει τα δεδομένα σε μορφή JSON. Η απόκριση JSON από το Google Maps API παρέχει όλα τα απαραίτητα δεδομένα, αλλά έρχεται σε μορφή που δεν είναι άμεσα χρησιμοποιήσιμη για τον κώδικα Lingo. Για να γεφυρωθεί αυτό το κενό, το πρόγραμμα επεξεργάζεται τα δεδομένα JSON για να δημιουργήσει δύο πίνακες: έναν για τις αποστάσεις και έναν για τις διάρκειες. Κάθε πίνακας οργανώνει τα δεδομένα με δομημένο τρόπο, καθιστώντας τα συμβατά με τις απαιτήσεις εισόδου για το Lingo. Για λόγους σαφήνειας, θα παρουσιαστεί ένα παράδειγμα του τρόπου με τον οποίο διαμορφώνεται η απάντηση JSON καθώς και του πώς θα έπρεπε να είναι τα δεδομένα ώστε να μπορούν να χρησιμοποιηθούν στο Lingo.

Έστω ότι υπάρχουν 3 πόλεις (Α,Β και Γ) με αποστάσεις και χρόνους ταξιδιού μεταξύ τους όπως δίνονται παρακάτω:

Πόλη Α σε Πόλη Β: 10.5 km, 15 λεπτά

Πόλη Α σε Πόλη Γ: 20.2 km, 30 λεπτά

Πόλη Β σε Πόλη Α: 10.5 km, 15 λεπτά

Πόλη Β σε Πόλη Γ: 12.3 km, 20 λεπτά

Πόλη Γ σε Πόλη Α: 20.2 km, 30 λεπτά

Πόλη Γ σε Πόλη Β: 12.3 km, 20 λεπτά

Εάν ορίζαμε τις 3 αυτές πόλεις ως δεδομένα στο πρόγραμμα το αποτέλεσμα που θα παρείχε το API θα είχε τη μορφή που παρουσιάζεται στο πλαίσιο 7.


```
"destination_addresses": ["City A", "City B", "City C"],
  "origin_addresses": ["City A", "City B", "City C"],
  "rows": [
    {
      "elements": [
        {"distance": {"text": "0.0 km", "value": 0}, "duration":
{"text": "0 mins", "value": 0}, "status": "OK"},
        {"distance": {"text": "10.5 km", "value": 10500}, "duration":
{"text": "15 mins", "value": 900}, "status": "OK"},
        {"distance": {"text": "20.2 km", "value": 20200}, "duration":
{"text": "30 mins", "value": 1800}, "status": "OK"}
      ]
    },
    {
      " elements ": [
        {"distance": {"text": "10.5 km", "value": 10500}, "duration":
{"text": "15 mins", "value": 900}, "status": "OK"},
        {"distance": {"text": "0.0 km", "value": 0}, "duration":
{"text": "0 mins", "value": 0}, "status": "OK"},
        {"distance": {"text": "12.3 km", "value": 12300}, "duration":
{"text": "20 mins", "value": 1200}, "status": "OK"}
      ]
    },
    {
      " elements ": [
        {"distance": {"text": "20.2 km", "value": 20200}, "duration":
{"text": "30 mins", "value": 1800}, "status": "OK"},
        {"distance": {"text": "12.3 km", "value": 12300}, "duration":
{"text": "20 mins", "value": 1200}, "status": "OK"},
        {"distance": {"text": "0.0 km", "value": 0}, "duration":
{"text": "0 mins", "value": 0}, "status": "OK"}
      ]
    }
  ],
  "status": "ok"
```


Πλαίσιο 7: Δεδομένα που ελήφθησαν από το API σε μορφή JSON

Η δομή των δεδομένων όπως παρέχεται από το API έχει ως εξής:

- **destination_addresses**: Λίστα διευθύνσεων προορισμού. Σε αυτή την περίπτωση, περιλαμβάνει τις διευθύνσεις «Πόλη Α», «Πόλη Β» και «Πόλη Γ».
- **origin_addresses**: Λίστα διευθύνσεων προέλευσης, οι οποίες είναι ίδιες με τις διευθύνσεις προορισμού.
- **rows**: Πρόκειται για έναν κατάλογο αντικειμένων (objects) όπου κάθε αντικείμενο αντιστοιχεί σε μια σειρά δεδομένων για μια συγκεκριμένη διεύθυνση προέλευσης. Κάθε γραμμή περιέχει τις αποστάσεις και τις διάρκειες προς όλες τις διευθύνσεις προορισμού.
    - Κάθε **row** περιέχει μία λίστα με **elements**. Κάθε element περιλαμβάνει:
        - **distance**: Ένα αντικείμενο που περιέχει:
            - **text**: Μία συμβολοσειρά που αναπαριστά την απόσταση μαζί με τις μονάδες μέτρησης (π.χ. "10.5 km").
            - **value**: Το μέγεθος της απόστασης σε μέτρα (π.χ. 10500 μέτρα για "10.5 km").
        - **duration**: Ένα αντικείμενο που περιέχει:
            - **text**: Μία συμβολοσειρά που αναπαριστά την διάρκεια ταξιδιού μαζί με τις μονάδες μέτρησης (π.χ. "15 mins").
            - **value**: Την αξία της διάρκειας ταξιδιού σε δευτερόλεπτα (π.χ. 900 seconds for "15 mins").
        - **status**: Μια κατάσταση που δείχνει το αποτέλεσμα της αίτησης (π.χ. «OK» αν τα δεδομένα ανακτήθηκαν επιτυχώς).

Όπως είναι σαφές η απόκριση του API εμπεριέχει όλα τα δεδομένα που είναι απαραίτητα για να επιλυθεί το TSP. Το πρόβλημα είναι πως τα δεδομένα αυτά δεν είναι σε μορφή που μπορούν να εισαχθούν στο LINGO. Για να λυθεί αυτό θα πρέπει να μετατραπούν σε $n \times n$ πίνακες, όπου n ο αριθμός των διευθύνσεων που εισήγαγε ο χρήστης (σε αυτό το παράδειγμα 3), οι οποίοι περιλαμβάνουν τις διευθύνσεις αφετηρίας στην πρώτη στήλη, τις διευθύνσεις τερματισμού στην πρώτη γραμμή και τις μεταξύ τους αποστάσεις/διάρκειες ταξιδιού στο εσωτερικό τους. Οι πίνακες του παραδείγματος, στη μορφή που θα έπρεπε να είναι ώστε να αξιοποιηθούν, παραθέτονται στα πλαίσια 7 και 8.

	Πόλη Α	Πόλη Β	Πόλη Γ
Πόλη Α	0	15	30
Πόλη Β	15	0	20
Πόλη Γ	30	20	0

Πλαίσιο 8: Πίνακας διαρκειών ταξιδιού για το παράδειγμα των πόλεων Α,Β,Γ

	Πόλη Α	Πόλη Β	Πόλη Γ
Πόλη Α	0	10.5	20.2
Πόλη Β	10.5	0	12.3
Πόλη Γ	20.2	12.3	0

Πλαίσιο 9: Πίνακας αποστάσεων για το παράδειγμα των πόλεων Α,Β,Γ

Προκειμένου το πρόγραμμα να το επιτύχει αυτό πραγματοποιεί μια σειρά από διεργασίες μέσω της συνάρτησης create_distance_duration_matrices. Αρχικά, εξάγει τις διευθύνσεις προέλευσης και προορισμού. Στη συνέχεια, πραγματοποιεί επαναλήψεις πάνω στις σειρές δεδομένων, όπου κάθε σειρά περιέχει τις αποστάσεις και τις διάρκειες μεταξύ κάθε ζεύγους προέλευσης και προορισμού. Για κάθε ζεύγος, εάν η προέλευση και ο προορισμός είναι οι ίδιοι, αποδίδει την τιμή 0 τόσο για την απόσταση όσο και για τη διάρκεια. Εάν διαφέρουν, εξάγει την αριθμητική απόσταση (σε χιλιόμετρα) και τη διάρκεια (σε λεπτά) από την απόκριση του API. Οι τιμές αυτές αποθηκεύονται σε λίστες, οι οποίες στη συνέχεια μετατρέπονται σε Pandas DataFrames για τη δημιουργία των τελικών πινάκων απόστασης και διάρκειας. Οι πίνακες αυτοί επιστρέφονται στη συνέχεια σε μορφή κατάλληλη για εξαγωγή σε αρχείο κειμένου (.txt), το οποίο μπορεί να χρησιμοποιηθεί ως είσοδος για τη βελτιστοποίηση του Lingo.

```
συνάρτηση create_distance_duration_matrices(matrix):
    Εξαγωγή δεδομένων από JSON
    origins = matrix['origin_addresses']
    destinations = matrix['destination_addresses']
    rows = matrix['rows']
    Δημιουργία λιστών για αποθήκευση αποστάσεων και διαρκειών
    distances = []
    durations = []
      for i, από 0 έως μήκος rows-1:
        Αρχικοποίηση λιστών για αποστάσεις και διάρκειες
        distance_row = []
        duration_row = []

        Λήψη των στοιχείων για την τρέχουσα σειρά
        elements = row['elements']

        Επανάληψη πάνω από κάθε στοιχείο της τρέχουσας γραμμής
        for j, από 0 έως lenght(elements)-1:

          if αφαιτηρία[i] ταυτίζεται με προορισμό[j]
                distance_value = 0.0
                duration_value = 0

            elif αν η κατάσταση του στοιχείου είναι 'OK' εξαγωγή
απόστασης και διάρκειας από το στοιχείο:
                distance = element['distance']['text']
                duration = element['duration']['text']

                Εξαγωγή αριθμητικών τιμών
                distance_value = EXTRACT_NUMBER(distance)
                duration_value = EXTRACT_NUMBER(duration)

         Μετατροπή της διάρκειας σε συνολικά λεπτά εάν περιέχει ώρες
                if διάρκεια) > 1 ώρα:
                    duration_value = CONVERT_TO_MINUTES
                 else:
                    duration_value = duration_value[0]
            else:
                distance_value = None
                duration_value = None

            Προσθήκη τιμών στις αντίστοιχες λίστες
            distance_row.append(distance_value)
            duration_row.append(duration_value)
            distances.append(distance_row)
            durations.append(duration_row)
Δημιουργία πλαισίων δεδομένων από τις λίστες απόστασης και διάρκειας
    df_distances = CREATE_DATAFRAME(distances, INDEX=origins,
COLUMNS=destinations)
df_durations = CREATE_DATAFRAME(durations, INDEX=origins,
COLUMNS=destinations)
    return df_distances, df_durations
```

Πλαίσιο 10: Ψευδοκώδικας για τη δημιουργία πινάκων απόστασης και διάρκειας ταξιδιού

Τελευταίο στάδιο της διαδικασίας δημιουργίας των πινάκων είναι η αποθήκευσή τους σε αρχείο κειμένου έτσι ώστε να μπορούν να εισαχθούν στο Lingo. Για να επιτευχθεί αυτό χρησιμοποιείται η συνάρτηση save_matrix_to_txt. Αναλυτικά αυτή η συνάρτηση:

- Ανοίγει το αρχείο στο καθορισμένο file_path για εγγραφή.
- Γράφει μια γραμμή στην κορυφή του αρχείου, χαρακτηρίζοντας τον πίνακα με το όνομα Distance ή Duration, ακολουθούμενο από τη λέξη «matrix», πριν από ένα ! για να υποδηλώσει ένα σχόλιο (π.χ. ! Distance matrix).
- Για κάθε γραμμή του πίνακα, μετατρέπει τις τιμές της γραμμής σε συμβολοσειρές και τις χωρίζει με κενά. Αυτό δημιουργεί μια μορφοποιημένη σειρά αριθμών στο αρχείο.
- Μόλις γραφτούν όλες οι γραμμές, κλείνει το αρχείο για να αποθηκεύσει τις αλλαγές.

```
Συνάρτηση save_matrix_to_txt(matrix, file_path, matrix_name):

άνοιγμα αρχείου στο file_path για εγγραφή
Δημιουργία σχολίου με το όνομα του πίνακα (π.χ. !Distance Matrix)

        for κάθε γραμμή του πίνακα:
            μετατροπή των τιμών κάθε γραμμής σε συμβολοσειρά
            διαχωρισμός των τιμών με κενά
            τοποθέτηση νέας συμβολοσειράς ως νέα γραμμή στο αρχείο

Κλείσιμο αρχείου
```

Πλαίσιο 11: Ψευδοκώδικας για την αποθήκευση των πινάκων ως αρχεία κειμένου

5.6 Χρήση του Lingo

Τώρα που οι πίνακες απόστασης και διάρκειας έχουν δημιουργηθεί σε μορφή που μπορεί να χρησιμοποιηθεί και ο χρήστης έχει καθορίσει τις διευθύνσεις μαζί με την προτίμησή του για βελτιστοποίηση με βάση είτε την απόσταση είτε τη διάρκεια, μπορεί να χρησιμοποιηθεί το Lingo. Το πρώτο βήμα που κάνει το πρόγραμμα είναι να δημιουργήσει τον κώδικα Lingo (όπως περιγράφεται στο παρόν κεφάλαιο), προσαρμοσμένο ώστε να είναι σύμφωνος με τα δεδομένα που εισήγαγε ο χρήστης. Με τον τρόπο αυτό διασφαλίζεται ότι η διαδικασία βελτιστοποίησης είναι προσαρμοσμένη στο συγκεκριμένο πρόβλημα, είτε πρόκειται για ελαχιστοποίηση της συνολικής απόστασης ταξιδιού είτε για ελαχιστοποίηση του χρόνου.

Για να επιτευχθεί αυτό χρησιμοποιείται η συνάρτηση generate_code με ορίσματα τον αριθμό των διευθύνσεων που όρισε ο χρήστης στην αρχή του προγράμματος και το αρχείο κειμένου με τον κατάλληλο πίνακα.

Πλαίσιο 12: Ψευδοκώδικας για τη δημιουργία κώδικα Lingo

```
Συνάρτηση generate_code(αριθμός διευθύνσεων, επιλεγμένος πίνακας):
    lingo_code = f"""
    SETS:

    VERTEX:U;
    ARC(VERTEX, VERTEX):D, X;

    ENDSETS

    DATA:

    N = {αριθμός διευθύνσεων};
    VERTEX = 1..N;
    D =@FILE({επιλεγμένος πίνακας});
    @TEXT(
'C:/Users/andre/PycharmProjects/pythonProject/RESULTS.TXT') = U;

    ENDDATA

    ΑΝΤΙΚΕΙΜΕΝΙΚΗ ΣΥΝΑΡΤΗΣΗ

    ΠΕΡΙΟΡΙΣΜΟΙ

     Ορισμός Μεταβλητής 'file_path' για την αποθήκευση του αρχείου
μοντέλου Lingo («model.lg4») as file:
    'Ανοιγμα του αρχείου 'file_path' και καταγραφή του κώδικα εκεί
```

Αφού έχει δημιουργηθεί ο κώδικας το επόμενο βήμα είναι να ανοίξει το Lingo, να καταγραφεί ο κώδικας και να τρέξει ώστε να παράξει τη βέλτιστη λύση. Για να επιτευχθεί αυτό δημιουργήθηκε η συνάρτηση run_lingo. Αφού δοθεί η εντολή, το πρόγραμμα περιμένει ένα μικρό χρονικό διάστημα κάποιων δευτερολέπτων ώστε να προλάβει ο υπολογιστής να φορτώσει το Lingo χωρίς προβλήματα. Στη συνέχεια χρησιμοποιείται η συντόμευση ctrl+u ώστε να τρέξει και να παράξει τη βέλτιστη λύση. Τέλος ελέγχεται αν κάτι πάει λάθος ώστε να τυπωθεί μήνυμα σφάλματος. Όπως αναφέρθηκε και προηγουμένως στο παρόν κεφάλαιο αφού λυθεί το πρόβλημα στο Lingo αποθηκεύονται σε ένα αρχείο κειμένου οι τιμές τις μεταβλητής U που αναπαριστούν τη σειρά με την οποία πρέπει το φορτηγό να επισκεφθεί τις διευθύνσεις προκειμένου να ελαχιστοποιηθεί η απόσταση ή η διάρκεια ταξιδιού. Η αποθήκευσή τους είναι απαραίτητο βήμα ώστε να μπορέσει στη συνέχεια το πρόγραμμα να τις διαβάσει και να τις χρησιμοποιήσει.

Πλαίσιο 13: Ψευδοκώδικας για άνοιγμα και χρησιμοποίηση του Lingo

```
Συνάρτηση run_lingo(file_path):
    Χρησιμοποίηση του 'os.startfile' για άνοιγμα του αρχείου που
καθορίζεται από το 'file_path'

# Run Lingo
    Κάλεσμα της συνάρτησης run_lingo με όρισμα το file
path("C:/Users/andre/PycharmProjects/pythonProject/model.lg4")

    Παύση του προγράμματος για (7) δευτερόλεπτα

    # χρησιμοποίηση pyaytogui για ενεργοποίηση της συντόμευσης
ctrl+u

    pyautogui.hotkey('ctrl', 'u')

    επέστρεψε origins

    εντοπισμός Exception as e:
    δημιουργία μηνύματος("Error", str(e))
```

5.7 Επεξεργασία αποτελεσμάτων

Οι τιμές της μεταβλητής U αποθηκεύονται τώρα σε ένα αρχείο κειμένου από το LINGO σε μια μορφή όμως που είναι μη ακέραιη (π.χ. 1,000). Για να μπορέσει το πρόγραμμα να επεξεργαστεί αυτές τις τιμές, πρέπει να μετατραπούν σε ακέραιους αριθμούς. Για το λόγο αυτό δημιουργείται η συνάρτηση read_results. Ο τρόπος που λειτουργεί είναι ο εξής:

Η συνάρτηση δημιουργεί μια κενή λίστα που ονομάζεται first_digits για την αποθήκευση του πρώτου ψηφίου κάθε γραμμής που βρίσκεται στο αρχείο. Στη συνέχεια, εισέρχεται σε έναν βρόχο, προσπαθώντας επανειλημμένα να ανοίξει και να διαβάσει από το αρχείο «RESULTS.TXT». Για κάθε μη κενή γραμμή στο αρχείο, ελέγχει αν ο πρώτος χαρακτήρας είναι ψηφίο και τον προσθέτει στη λίστα. Μόλις ολοκληρωθεί η επεξεργασία

όλων των γραμμών, τα ψηφία μετατρέπονται σε ακέραιους και το αρχείο διαγράφεται. Στη συνέχεια, η συνάρτηση επιστρέφει τη λίστα των ακεραίων αριθμών. Εάν το αρχείο δεν βρεθεί, εκτυπώνει ένα μήνυμα σφάλματος και περιμένει για 5 δευτερόλεπτα πριν επαναλάβει την προσπάθεια.

Πλαίσιο 14: Ψευδοκώδικας για αφαίρεση δεκαδικών από αποτελέσματα του Lingo

```
def read_results():
    first_digits = Δημιουργία κενής λίστας
    εισαγωγή σε βρόχο:
        try:
            with άνοιγμα αρχείου("RESULTS.TXT", σε read mode")
                for κάθε γραμμή του αρχείου:
                    αφαίρεση κενών διαστημάτων

                    if η γραμμή δεν είναι κενή
                     πάρε τον πρώτο χαρακτήρα της γραμμής

                     if ο πρώτος χαρακτήρας είναι ψηφίο
                         τοποθέτησή του στη λίστα first_digits

            first_digits = μετατροπή των ψηφίων από string σε
ακέραιους αριθμούς

            διαγραφή αρχείου("RESULTS.TXT")
            επιστροφή first_digits

           Εάν ο φάκελος δεν βρεθεί:
            εκτύπωση("File not found. Retrying in 5 seconds...")

            time.sleep(5)  # αναμονή 5 δευτερολέπτων και
επαναπροσπάθεια
```

Έχοντας καθορίσει τη βέλτιστη ακολουθία για την επίσκεψη των διευθύνσεων και έχοντας αποθηκεύσει τις πληροφορίες αυτές σε εύχρηστη μορφή, το πρόγραμμα πρέπει τώρα να αναδιατάξει τις διευθύνσεις εισόδου σύμφωνα με αυτή την ακολουθία. Το επόμενο βήμα περιλαμβάνει τη μετάφραση των αριθμών ακολουθίας από το αρχείο αποτελεσμάτων σε έναν κατάλογο διευθύνσεων τοποθετημένων με τη σωστή σειρά. Αυτό εξασφαλίζει ότι οι διευθύνσεις παρουσιάζονται στη βέλτιστη διαδρομή που προκύπτει από τη λύση του προβλήματος του πλανόδιου πωλητή (TSP), επιτρέποντας την αποτελεσματική απεικόνιση ή χρήση σε επόμενα βήματα.

Προκειμένου να επιτευχθούν τα παραπάνω το πρόγραμμα επιτελεί της εξής διεργασίες: Αρχικά, ανακτά τον κατάλογο διευθύνσεων προέλευσης από τη συνάρτηση on_submit(). Στη συνέχεια, διαβάζει τους διατεταγμένους αριθμούς ακολουθίας χρησιμοποιώντας τη συνάρτηση read_results(). Αυτοί οι αριθμοί υποδεικνύουν τη βέλτιστη σειρά επίσκεψης για τις διευθύνσεις. Δημιουργείται ένας χάρτης αντιστοίχισης

για τη συσχέτιση κάθε αριθμού ακολουθίας με την αντίστοιχη διεύθυνση ώστε να τοποθετηθούν οι διευθύνσεις στη σωστή σειρά. Στη συνέχεια, ο κώδικας δημιουργεί έναν κατάλογο προορισμών με τη σωστή πλέον σειρά και προσθέτει σε κάθε προορισμό τον αριθμό ακολουθίας του. Αυτός ο μορφοποιημένος κατάλογος ενώνεται σε μια συμβολοσειρά, η οποία εκτυπώνεται για να εμφανίσει τη βέλτιστη διαδρομή με τις διευθύνσεις αριθμημένες και ταξινομημένες.

Πλαίσιο 15: Ψευδοκώδικας για επανατοποθέτηση και εκτύπωση διευθύνσεων στη σωστή σειρά

```
order_numbers = το αποτέλεσμα της συνάρτησης read_results

origin_map = Δημιουργία ενός χάρτη που αντιστοιχίζει κάθε αριθμό από
την order_numbers στην αντίστοιχη διεύθυνση προέλευσης.

sorted_origin_map = επανατοποθέτηση διευθύνσεων με βάση τον αριθμό
από την order_numbers με τον οποίο έχουν αντιστοιχιστεί

destinations_for_google = δημιουργία λίστας με τις διευθύνσεις σε
σωστή σειρά χωρίς να είναι αριθμημένες ώστε να μπορούν στην συνέχεια
να εισαχθούν στο google maps

ordered_destinations = δημιουργία λίστας με τις διευθύνσεις σε σωστή
σειρά και αριθμημένες


ordered_destinations_str = μετατροπή ordered_destinations σε μορφή
string ώστε να μπορεί να γίνει print

print(ordered_destinations_str)
```

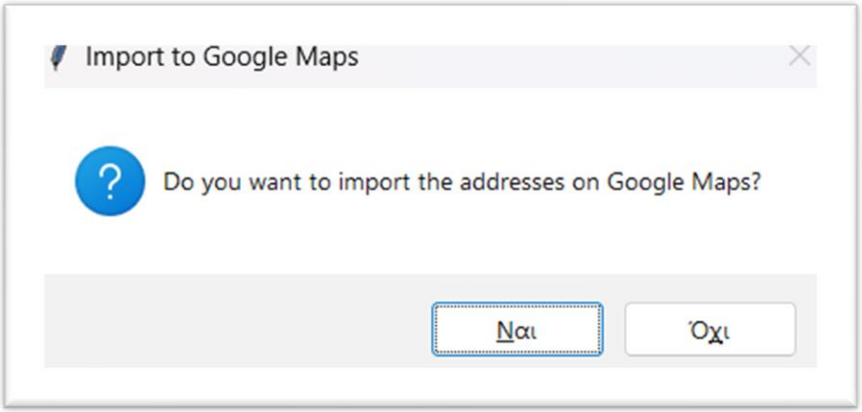
5.8 Εισαγωγή τελικής διαδρομής στο Google Maps

Πλέον όλες οι διευθύνσεις είναι αποθηκευμένες με τέτοια σειρά ώστε να διασφαλίζεται η ελαχιστοποίηση της απόστασης ή της διάρκειας διαδρομής. Το τελευταίο στάδιο που έχει να επιτελέσει το πρόγραμμα είναι η εισαγωγή αυτών των διευθύνσεων στους χάρτες Google ενισχύοντας έτσι την ευκολία με την οποία μπορεί το πρόγραμμα να χρησιμοποιηθεί.

Προκειμένου να επιτευχθεί αυτό δημιουργήθηκε η συνάρτηση open_dialog_for_google_insert η οποία λειτουργεί ως εξής: Η εντολή

messagebox.askquestion() χρησιμοποιείται για την εμφάνιση ενός πλαισίου διαλόγου που ρωτά τον χρήστη αν θέλει να εισαγάγει τις διευθύνσεις στους Χάρτες Google.

- **Αν ο χρήστης απαντήσει ναι:** καλείται η συνάρτηση open_google_maps με όρισμα την ταξινομημένη πλέον λίστα με διευθύνσεις. Η συνάρτηση αυτή ελέγχει πρώτα αν ο αριθμός των διευθύνσεων είναι μικρότερος από 2. Αν είναι, δημιουργεί ένα Σφάλμα επειδή απαιτούνται τουλάχιστον δύο διευθύνσεις για τη δημιουργία της τελικής διαδρομής. Εάν είναι μεγαλύτερος από 2 κατασκευάζει μια διεύθυνση URL στο Google Maps για να παρέχει οδηγίες μεταξύ των συγκεκριμένων διευθύνσεων.
- **Αν ο χρήστης απαντήσει όχι:** Εκτυπώνεται το μήνυμα: "Addresses not imported to Google Maps."



Εικόνα 3: Πλαίσιο διαλόγου για εισαγωγή διευθύνσεων στο Google maps

Πλαίσιο 16: Ψευδοκώδικας για εισαγωγή διευθύνσεων στο Google Maps

```
Συνάρτηση open_dialog_for_google_insert:
    import_choice = δημιουργία πλαισίου διαλόγου("Do you want to
import the addresses on Google Maps?")

    if ο χρήστης επιλέξει yes
        # Code to import addresses to Google Maps
        Συνάρτηση open_google_maps(λίστα διευθύνσεων):
            if διευθύνσεις < 2:
                δημιουργία Error("At least two addresses are required
for directions.")

            # Base URL for Google Maps directions
            base_url = δημιουργία διεύθυνσης url για google maps

            #Encodethe addresses to be URL-friendly
            encoded_addresses = τροποποίηση λίστας διευθύνσεων ώστε
να είναι url friendly

            # Create the full URL with all the addresses in order
            directions_url = εισαγωγή τον τροποποιημένων διευθύνσεων
στο google maps url

            Άνοιγμα url σε browser

        open_google_maps(destinations_for_google)
    else:
        εκτύπωσε("Addresses not imported to Google Maps.")
```

6 Πρακτική εφαρμογή του προγράμματος

Σε αυτό το κεφάλαιο, παρουσιάζεται η εφαρμογή του προγράμματος σε ένα πραγματικό σενάριο, το οποίο αφορά μια διαδρομή φορτηγού διανομής της εταιρείας «Λυκούδης Μεταφορική» στο Αργοστόλι Κεφαλονιάς. Αρχικά, περιγράφεται ο τρόπος με τον οποίο η εταιρεία οργανώνει και διαχειρίζεται τη δρομολόγηση των φορτηγών της, καθώς και η διαδρομή που ακολούθησε ο οδηγός για τη συγκεκριμένη διανομή. Στη συνέχεια, χρησιμοποιώντας το πρόγραμμα, δημιουργείται μία βελτιστοποιημένη διαδρομή, η οποία συγκρίνεται με την αρχική πορεία. Τέλος, παρουσιάζονται τα συμπεράσματα που προκύπτουν από τη χρήση του προγράμματος, αναδεικνύοντας τα οφέλη και τις πιθανές βελτιώσεις που προσφέρει στη διαδικασία δρομολόγησης.

6.1 Περιορισμοί του προγράμματος

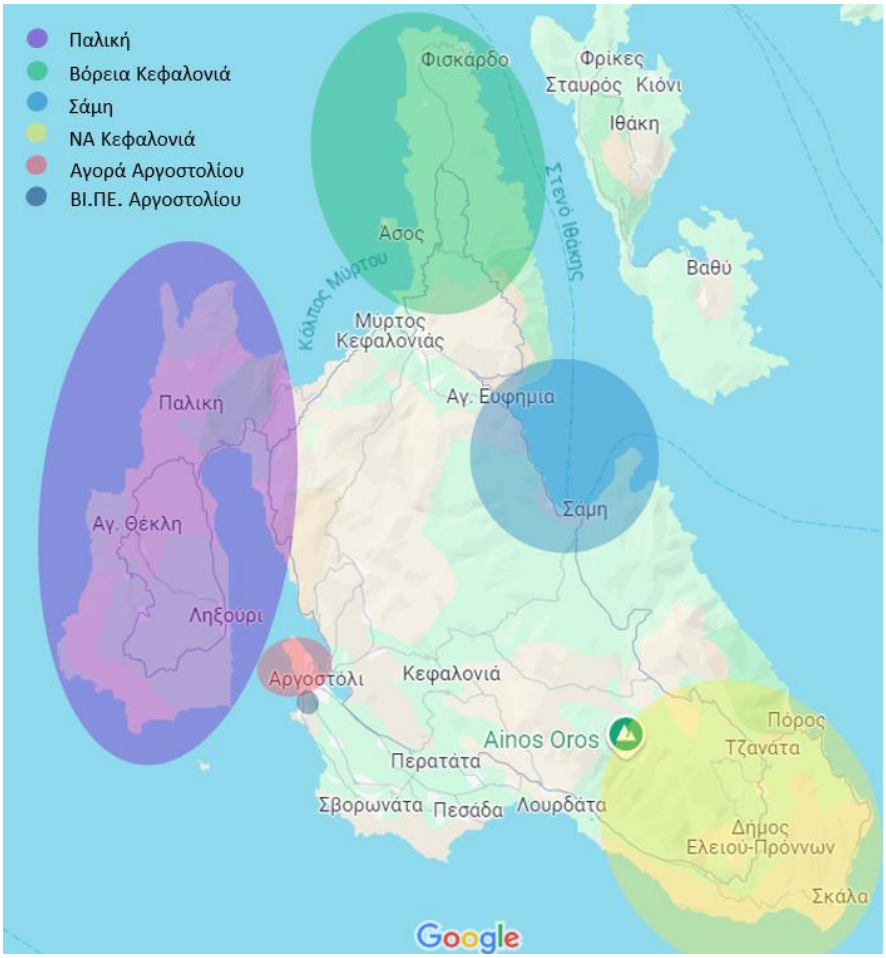
Όπως έχει αναφερθεί και σε προηγούμενα κεφάλαια, το πρόγραμμα χρησιμοποιεί το Google Distance Matrix API. Το συγκεκριμένο API, όπως και όλες οι αντίστοιχες υπηρεσίες της Google, παρέχεται δωρεάν, αλλά υπόκειται σε ορισμένους περιορισμούς. Στην προκειμένη περίπτωση, επιτρέπεται η δημιουργία πινάκων απόστασης και χρόνου με μέγιστο μέγεθος 10x10 σημείων. Για το λόγο αυτό, ήταν απαραίτητο να επιλεγεί μια διαδρομή που να συμμορφώνεται με αυτόν τον περιορισμό.

Προφανώς, σε μια τέτοια διαδρομή δεν είναι εύκολο να αναδειχθούν πλήρως τα πλεονεκτήματα και η αποτελεσματικότητα του προγράμματος, ειδικά σε έναν γεωγραφικό τόπο όπως η Κεφαλονιά, η οποία δεν διαθέτει πολύπλοκο οδικό δίκτυο ή σύνθετη ρυμοτομία. Παρ’ όλα αυτά, όπως θα φανεί και στη συνέχεια, το πρόγραμμα κατάφερε να βελτιώσει τη διαδρομή σε σημαντικό βαθμό, αποδεικνύοντας τη χρησιμότητά του ακόμη και σε πιο απλοϊκά σενάρια.

6.2 Προσέγγιση της εταιρείας

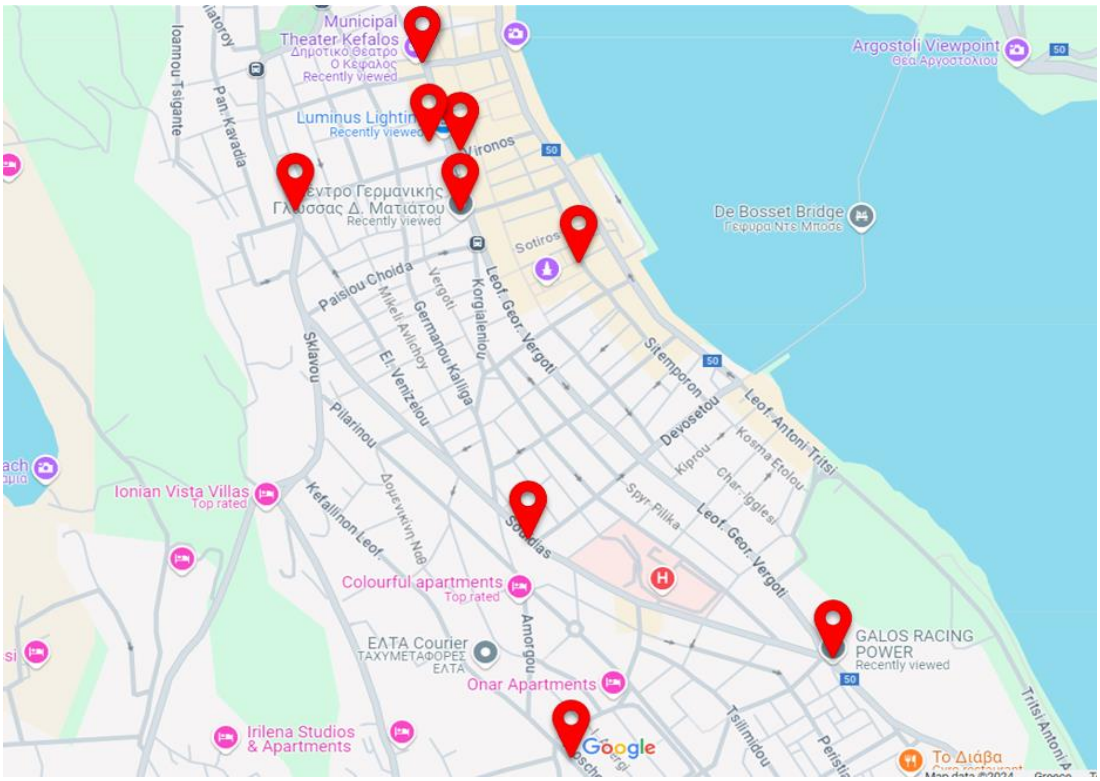
Η προσέγγιση που ακολουθεί η εταιρεία στη δρομολόγηση και τη διανομή των προϊόντων βασίζεται κυρίως στην εμπειρία. Η Κεφαλονιά χωρίζεται σε διαφορετικά γεωγραφικά τμήματα: την Παλική, τη Βόρεια Κεφαλονιά, τη Νοτιοανατολική Κεφαλονιά, τη Σάμη, την αγορά Αργοστολίου (κέντρο της πόλης) και τη Βιομηχανική Περιοχή (ΒΙ. ΠΕ.) Αργοστολίου. Όταν φτάνει τράκτορας από την Αθήνα, τα προϊόντα εκφορτώνονται και ταξινομούνται στην αποθήκη σύμφωνα με τον τελικό προορισμό τους.

Ο καταμερισμός των προϊόντων γίνεται προσεκτικά, ώστε να βρίσκονται σε προκαθορισμένες θέσεις, ανάλογα με το τμήμα του νησιού στο οποίο προορίζονται. Όταν η ζήτηση για κάποιο συγκεκριμένο προορισμό φτάσει σε ένα επιθυμητό επίπεδο —το οποίο καθορίζεται εμπειρικά από τους υπεύθυνους— τότε τα προϊόντα φορτώνονται σε ένα φορτηγό διανομής και πραγματοποιείται η αποστολή τους.



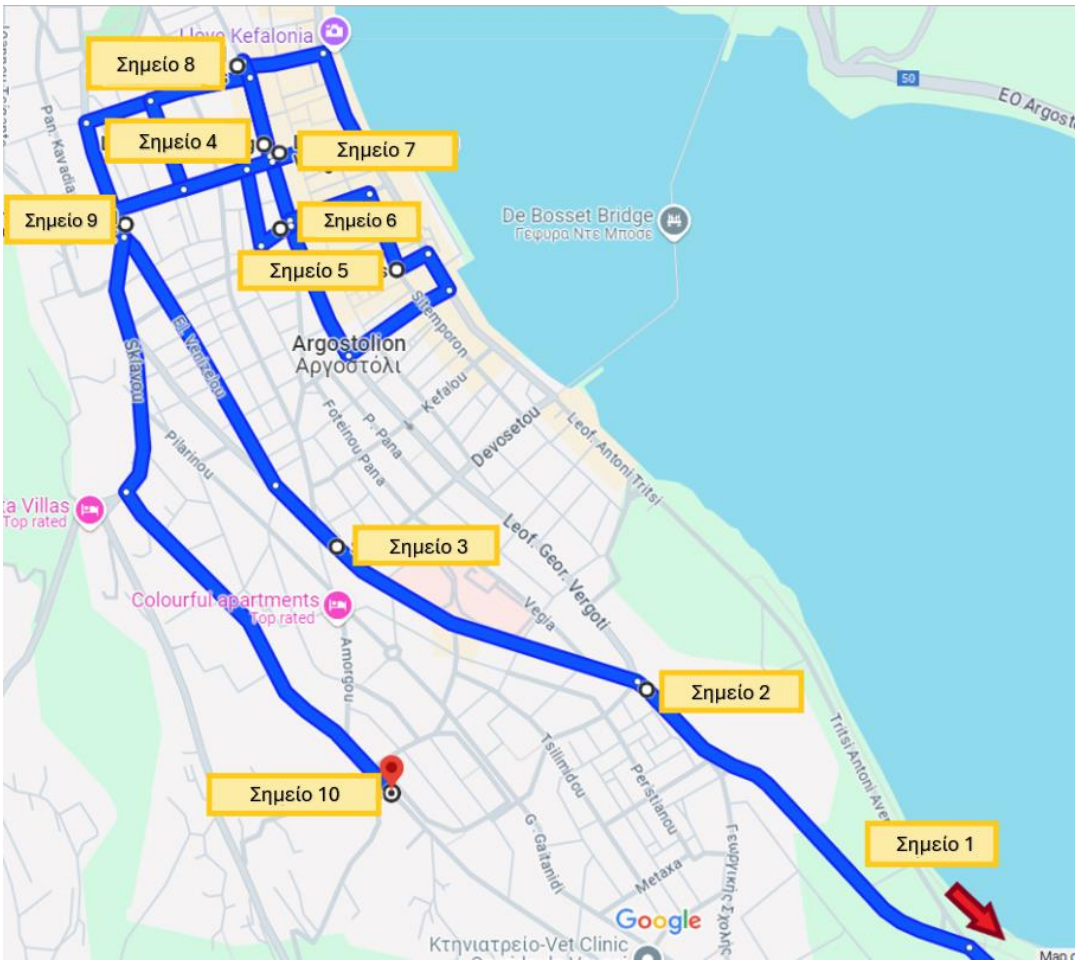
Εικόνα 4: Χάρτης Κεφαλονιάς χωρισμένος σε διαμερίσματα σύμφωνα με την εταιρεία

Στη συγκεκριμένη εφαρμογή του προγράμματος, επικεντρωθήκαμε στην περιοχή της αγοράς του Αργοστολίου. Ο λόγος αυτής της επιλογής είναι ότι οι περιφερειακές διαδρομές στην Κεφαλονιά επιδέχονται ελάχιστες βελτιώσεις λόγω του περιορισμένου οδικού δικτύου. Συγκεκριμένα, μελετήθηκε μία διαδρομή που περιλαμβάνει 9 σημεία μέσα στην αγορά του Αργοστολίου, τα οποία φαίνονται στην παρακάτω εικόνα.



Εικόνα 5: Σημεία στο χάρτη που καλείται να επισκεφθεί το φορτηγό

Η φόρτωση των αντικειμένων έγινε πριν την αναχώρηση του φορτηγού, με βάση την εμπειρία του οδηγού και τη διαδρομή που είχε σχεδιάσει. Ακολούθως, ξεκίνησε η διανομή, και τα σημεία εξυπηρετήθηκαν με τη σειρά που είχε προκαθορίσει ο οδηγός. Η σειρά με την οποία εξυπηρετήθηκαν τα σημεία καθώς και η ακριβής διαδρομή που ακολούθησε το φορτηγό φαίνεται στην παρακάτω εικόνα.



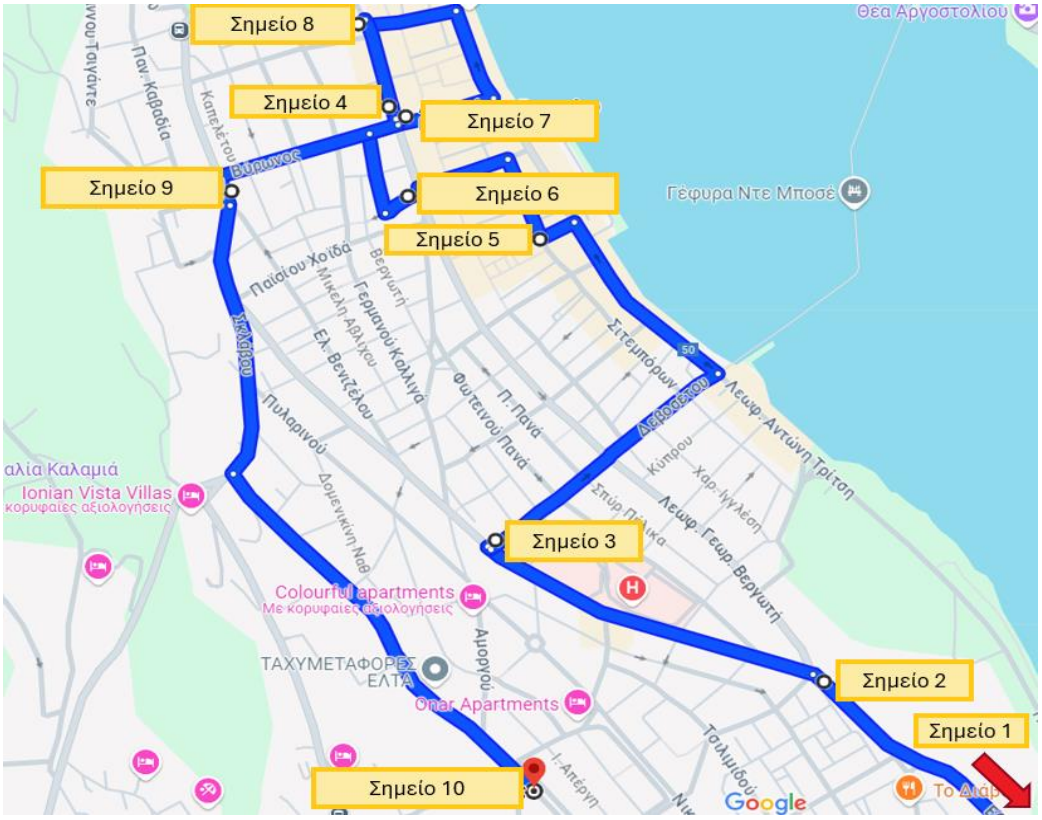
Εικόνα 6: Διαδρομή που ακολούθησε το φορτηγό βάσει εμπειρίας

Η συνολική απόσταση που κάλυψε το φορτηγό, καθώς και ο μέσος χρόνος που απαιτείται για την επίσκεψη όλων των σημείων και την επιστροφή στην αφετηρία, όπως υπολογίστηκαν από τους Χάρτες της Google, είναι 14,1 χιλιόμετρα και 30 λεπτά αντίστοιχα.

6.3 Εφαρμογή του προγράμματος

Σε αυτό το κεφάλαιο, παρουσιάζεται η εφαρμογή του προγράμματος που αναπτύχθηκε για την εύρεση της βέλτιστης διαδρομής διανομής. Η διαδικασία περιλαμβάνει την εισαγωγή των διευθύνσεων των σημείων παράδοσης και τη δημιουργία μιας διαδρομής που ελαχιστοποιεί είτε τη συνολική απόσταση είτε τον απαιτούμενο χρόνο μετακίνησης.

Στη συγκεκριμένη περίπτωση, η βελτιστοποίηση της διαδρομής πραγματοποιήθηκε αρχικά με βάση τη συνολική απόσταση και, στη συνέχεια, με βάση τον συνολικό χρόνο μετακίνησης. Παρόλο που χρησιμοποιήθηκαν διαφορετικά κριτήρια, η διαδρομή που προέκυψε ήταν ακριβώς η ίδια και στις δύο περιπτώσεις. Το αποτέλεσμα αυτό ήταν αναμενόμενο, δεδομένου ότι η κίνηση στην Κεφαλονιά είναι συνήθως περιορισμένη και δεν αποτελεί καθοριστικό παράγοντα για τον σχεδιασμό της βέλτιστης διαδρομής. Η έλλειψη κυκλοφοριακής συμφόρησης και οι απλές οδικές συνθήκες καθιστούν την απόσταση το πρωταρχικό κριτήριο βελτιστοποίησης, με τον χρόνο να επηρεάζεται ελάχιστα. Έτσι, το πρόγραμμα απέδωσε το ίδιο αποτέλεσμα και με τις δύο μεθόδους.



Εικόνα 7:Βέλτιστη διαδρομή όπως προέκυψε από το πρόγραμμα

Η σειρά που πρέπει το φορτηγό να επισκεφθεί τα σημεία ώστε να προκύψει η βέλτιστη διαδρομή σύμφωνα με το πρόγραμμα είναι η εξής:

Σημείο 1→ Σημείο 2→ Σημείο 3→ Σημείο 5 → Σημείο 6 → Σημείο 7 →Σημείο 8 → Σημείο 4 → Σημείο 9 → Σημείο 10→ Σημείο 1

Η απόσταση που χρειάζεται να καλυφθεί και ο χρόνος, όπως υπολογίζονται από την εφαρμογή Google Maps, είναι 12,1 χιλιόμετρα και 23 λεπτά αντίστοιχα.

Όπως γίνεται αντιληπτό, η βελτιστοποίηση της διαδρομής οδήγησε σε μια μείωση περίπου 2 χιλιομέτρων στην απόσταση και 7 λεπτών στον χρόνο (ή 14% στην απόσταση και 23% στο χρόνο). Αν και αυτή η διαφορά μπορεί αρχικά να φαίνεται μικρή, η πραγματική της αξία αναδεικνύεται όταν λάβουμε υπόψη ότι οι διαδρομές αυτές πραγματοποιούνται καθημερινά και, κατά την καλοκαιρινή περίοδο, ακόμη και 2 ή 3 φορές την ημέρα. Σε βάθος χρόνου, η εξοικονόμηση καυσίμου που προκύπτει είναι ιδιαίτερα σημαντική, συμβάλλοντας τόσο στη μείωση του λειτουργικού κόστους όσο και στον περιορισμό του περιβαλλοντικού αποτυπώματος της εταιρείας.

7 Συμπεράσματα

Η εργασία αυτή επικεντρώθηκε στη δημιουργία ενός προγράμματος σε Python που επιλύει το πρόβλημα του Πλανόδιου Πωλητή (TSP) και βελτιστοποιεί τη δρομολόγηση φορτηγών για μια μεταφορική εταιρεία. Ο βασικός στόχος ήταν να κατασκευαστεί ένα εργαλείο που θα μπορούσε να διαχειρίζεται αποτελεσματικά τις καθημερινές διαδρομές διανομής, με στόχο τη μείωση της απόστασης ή/και του χρόνου που απαιτείται για την εξυπηρέτηση των πελατών.

Το πρόγραμμα χρησιμοποιεί αλγοριθμικές τεχνικές για την εύρεση της βέλτιστης διαδρομής, λαμβάνοντας υπόψη τόσο την ελαχιστοποίηση της απόστασης όσο και του χρόνου μετακίνησης. Η πρακτική εφαρμογή του προγράμματος πραγματοποιήθηκε σε μία πραγματική διαδρομή της εταιρείας «Λυκούδης Μεταφορική» στο Αργοστόλι της Κεφαλονιάς. Τα αποτελέσματα έδειξαν ότι το πρόγραμμα ήταν σε θέση να μειώσει τη συνολική απόσταση κατά 14% και τον χρόνο διανομής κατά 23%. Αυτή η φαινομενικά μικρή διαφορά μπορεί να έχει σημαντικό αντίκτυπο μακροπρόθεσμα, ειδικά αν αναλογιστεί κανείς ότι τέτοιες διαδρομές επαναλαμβάνονται πολλές φορές μέσα στη μέρα, ιδιαίτερα κατά τις περιόδους αιχμής.

Τα συμπεράσματα από τη μελέτη δείχνουν ότι η χρήση της τεχνολογίας για την υποστήριξη των διαδικασιών δρομολόγησης δεν αποτελεί απλώς ένα εργαλείο αυτοματοποίησης, αλλά μια λύση που μπορεί να προσφέρει σημαντικά οφέλη σε καθημερινές λειτουργίες. Η βελτιστοποίηση των δρομολογίων οδηγεί σε μείωση του κόστους καυσίμου, μικρότερη φθορά των οχημάτων και πιο άμεση εξυπηρέτηση των πελατών. Παράλληλα, το πρόγραμμα αναδεικνύει την ικανότητά του να προσαρμόζεται σε διαφορετικές γεωγραφικές περιοχές και συνθήκες, προσφέροντας σταθερά αποτελέσματα ακόμα και σε τοπικά δίκτυα με λιγότερο σύνθετη ρυμοτομία.

Συνολικά, η εργασία αυτή απέδειξε την αξία της εφαρμογής αλγοριθμικών μεθόδων στη βελτιστοποίηση διαδρομών και κατέδειξε τη δυνατότητα περαιτέρω βελτίωσης των επιχειρησιακών διαδικασιών, ενισχύοντας τη συνολική αποδοτικότητα της επιχείρησης.

8 Παραρτήματα

8.1 Κώδικας LINGO

```
    SETS:
    VERTEX:U;
    ARC(VERTEX, VERTEX):D, X;

    ENDSETS
    DATA:

    N = 3;
    VERTEX = 1..N;
    D =@FILE(distance_matrix.txt);
    @TEXT( 'C:/Users/andre/PycharmProjects/pythonProject/RESULTS.TXT')
= U;


    ENDDATA

    !OBJ;
    MIN = @SUM(ARC(I,J) | I#NE#J: D(I,J)*X(I,J));

    !ARRIVALS;
    @FOR(VERTEX(I) : @SUM(VERTEX(J)|J#NE#I: X(I,J)) = 1);

    !DEPARTURES;
    @FOR(VERTEX(J) : @SUM(VERTEX(I)|I#NE#J: X(I,J)) = 1);

    !BINARY TYPE;
    @FOR(ARC(I,J): @BIN(X(I,J)));

    !SUBTOUR ELIMINATION;
    @FOR(ARC(I,J) | I#NE#1 #AND# J#NE#1 : U(I) - U(J) + N*X(I,J) <= N-
1);
    @FOR(VERTEX(I) |I#NE#1: U(I) >= 1);
    @FOR(VERTEX(I) |I#NE#1: U(I) <= N-1);
```

Πλαίσιο 17: Κώδικας LINGO

8.2 Κώδικας Python

```
import os
import re
import time
import tkinter as tk
import urllib.parse
import webbrowser
from tkinter import simpledialog, messagebox

import pandas as pd
import pyautogui
import requests


def get_distance_matrix(api_key, origins, destinations,
mode="driving"):
    url = "https://maps.googleapis.com/maps/api/distancematrix/json"

    origins_str = '|'.join(origins)
    destinations_str = '|'.join(destinations)

    params = {
        'origins': origins_str,
        'destinations': destinations_str,
        'mode': mode,
        'key': api_key
    }

    response = requests.get(url, params=params)

    if response.status_code == 200:
        return response.json()
    else:
        raise Exception(f"Error: {response.status_code} -
{response.text}")


def create_distance_duration_matrices(matrix):
    origins = matrix['origin_addresses']
    destinations = matrix['destination_addresses']
    rows = matrix['rows']

    distances = []
    durations = []

    for i, row in enumerate(rows):
        distance_row = []
        duration_row = []
        elements = row['elements']
```

```
for j, element in enumerate(elements):
            if origins[i] == destinations[j]:
                distance_value = 0.0
                duration_value = 0
            elif element['status'] == 'OK':
                distance = element['distance']['text']
                duration = element['duration']['text']

                # Extract numeric values
                distance_value = float(re.findall(r'\d+\.?\d*',
distance.replace(',', ''))[0])  # Keep only the number
                duration_value = re.findall(r'\d+',
duration.replace(',', ''))  # Extract all numbers

                if len(duration_value) > 1:
                    duration_value = int(duration_value[0]) * 60 +
int(
                        duration_value[1])  # Convert hours and
minutes to total minutes
                else:
                    duration_value = int(duration_value[0])
            else:
                distance_value = None
                duration_value = None

            distance_row.append(distance_value)
            duration_row.append(duration_value)
        distances.append(distance_row)
        durations.append(duration_row)

    df_distances = pd.DataFrame(distances, index=origins,
columns=destinations)
    df_durations = pd.DataFrame(durations, index=origins,
columns=destinations)

    return df_distances, df_durations


def save_matrix_to_txt(matrix, file_path, matrix_name):
    with open(file_path, 'w') as f:
        f.write(f"! {matrix_name} matrix;\n")
        for row in matrix.values:
            f.write(" ".join(map(str, row)) + "\n")



def run_lingo(file_path):
    os.startfile(file_path)



def on_submit():
    origins = [simpledialog.askstring("Input", f"Enter Origin {i +
1}:") for i in range(num_addresses)]
```

```
try:
        # Get the distance matrix
        distance_matrix = get_distance_matrix(api_key, origins,
destinations)
        df_distances, df_durations =
create_distance_duration_matrices(distance_matrix)

        # Save the matrices to text files
        save_matrix_to_txt(df_distances, "distance_matrix.txt",
"Distance")
        save_matrix_to_txt(df_durations, "duration_matrix.txt",
"Duration")

        # Run Lingo

run_lingo("C:/Users/andre/PycharmProjects/pythonProject/model.lg4")
        time.sleep(10)

        # Send Ctrl + U to run the LINGO code
        pyautogui.hotkey('ctrl', 'u')
        return origins
    except Exception as e:
        messagebox.showerror("Error", str(e))



def generate_code(num_addresses, file):
    lingo_code = f"""
    SETS:
    VERTEX:U;
    ARC(VERTEX, VERTEX):D, X;

    ENDSETS
    DATA:

    N = {num_addresses};
    VERTEX = 1..N;
    D =@FILE({file});
    @TEXT(
'C:/Users/andre/PycharmProjects/pythonProject/RESULTS.TXT') = U;


    ENDDATA

    !OBJ;
    MIN = @SUM(ARC(I,J) | I#NE#J: D(I,J)*X(I,J));

    !ARRIVALS;
    @FOR(VERTEX(I) : @SUM(VERTEX(J)|J#NE#I: X(I,J)) = 1);

    !DEPARTURES;
    @FOR(VERTEX(J) : @SUM(VERTEX(I)|I#NE#J: X(I,J)) = 1);

    !BINARY TYPE;
    @FOR(ARC(I,J): @BIN(X(I,J)));

    !SUBTOUR ELIMINATION;
    @FOR(ARC(I,J) | I#NE#1 #AND# J#NE#1 : U(I) - U(J) + N*X(I,J) <=
N-1);
    @FOR(VERTEX(I) |I#NE#1: U(I) >= 1);
    @FOR(VERTEX(I) |I#NE#1: U(I) <= N-1);
```

```
"""
    file_path = "model.lg4"
    with open(file_path, "w") as file:
        file.write(lingo_code)

def read_results():
    first_digits = []
    while True:
        try:
            with open("RESULTS.TXT", "r") as file:
                for line in file:
                    line = line.strip()

                    if line:  # Check if the line is not empty
                        first_digit = line[0]

                        if first_digit.isdigit():
                            first_digits.append(first_digit)
            first_digits = [int(digit) for digit in first_digits]
            os.remove("RESULTS.TXT")
            return first_digits
        except FileNotFoundError:
            print("File not found. Retrying in 5 seconds...")
            time.sleep(1)  # Wait for 5 seconds before retrying

def open_dialog_for_google_insert(destinations_for_google):
    import_choice = messagebox.askquestion("Import to Google Maps",
                                           "Do you want to import the
addresses on Google Maps?")

    if import_choice == 'yes':
        # Code to import addresses to Google Maps
        def open_google_maps(addresses):
            if len(addresses) < 2:
                raise ValueError("At least two addresses are required
for directions.")

            # Base URL for Google Maps directions
            base_url = "https://www.google.com/maps/dir/"

            # Encode the addresses to be URL-friendly
            encoded_addresses = [urllib.parse.quote_plus(address) for
address in addresses]

            # Create the full URL with all the addresses in order
            directions_url = base_url + "/".join(encoded_addresses)

            # Open the URL in the default web browser
            webbrowser.open(directions_url)

        open_google_maps(destinations_for_google)
    else:
        print("Addresses not imported to Google Maps.")

def get_api_key_and_num_addresses():
    global num_addresses
    num_addresses = int(num_addresses_entry.get())
    matrix_choice_value = matrix_choice.get()
    generate_code(num_addresses, matrix_choice_value)
    origins = on_submit()
    order_numbers = read_results()
```

```
origin_map = {order_numbers[i]: origins[i] for i in
range(len(order_numbers))}
    sorted_origin_map = {k: origin_map[k] for k in
sorted(origin_map)}
    destinations_for_google = [f"{destination}" for key,destination
in sorted_origin_map.items()]
    ordered_destinations = [f"{key}. {destination}" for key,
destination in sorted_origin_map.items()]
    ordered_destinations_str = "\n".join(ordered_destinations)
    print(ordered_destinations_str)
    open_dialog_for_google_insert(destinations_for_google)

# Setup the Tkinter window
root = tk.Tk()
root.title("TSP Solver")

tk.Label(root, text="Number of Addresses:").pack()
num_addresses_entry = tk.Entry(root)
num_addresses_entry.pack()

# Add a selection for distance or duration
matrix_choice = tk.StringVar(value="distance_matrix.txt")  # Default
to "distance"

tk.Label(root, text="Choose Matrix:").pack()

tk.Radiobutton(root, text="Distance", variable=matrix_choice,
value="distance_matrix.txt").pack()
tk.Radiobutton(root, text="Duration", variable=matrix_choice,
value="duration_matrix.txt").pack()



tk.Button(root, text="Submit",
command=get_api_key_and_num_addresses).pack()

root.mainloop()
```

Πλαίσιο 18: Κώδικας Python

Βιβλιογραφία

[1] K. Menger, «Contributions to the theory of simply connected spatial systems,» 1930.

[2] Dantzig, Fulkerson και Johnson, «Solution of a Large Scale Traveling-Salesman Problem,» 1954.

[3] Kruskal και Joseph, «On the Shortest Spanning Tree of a Graph and the Traveling Salesman Problem,» 1956.

[4] Miller, Tucker και Zemlin, «Integer Programming Formulation of Traveling Salesman Problems,» 1960.

[5] Chrisman και James, «The Clustered Traveling Salesman Problem,» 1975.

[6] Gavish και Graves, «The Traveling Salesman Problem and Related Problems,» 1978.

[7] Laporte και Nobert, «A Cutting Planes Algorithm for the m-Salesmen Problem,» 1980.

[8] Gavish και Srikanth, «An Optimal Solution Method for Large-Scale Multiple Traveling Salesmen Problems,» 1986.

[9] Dorigo και Gambardella, «Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem,» 1997.

[10] Applegate, Bixby, Chvatal και Cook, «TSP Cuts Which Do Not Conform to the Template Paradigm,» 2001.

[11] Goyal, «A Survey on Travelling Salesman Problem,» 2010.

[12] T. E. o. E. Britannica, «Britannica,» [Ηλεκτρονικό]. Available: https://www.britannica.com/science/linear-programming-mathematics.

[13] «Wikipedia,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Google_Maps.

[14] «Apple Support,» Apple, [Ηλεκτρονικό]. Available: https://support.apple.com/el-gr/guide/shortcuts-mac/apd2e30c9d45/mac.

[15] «Python,» [Ηλεκτρονικό]. Available: https://www.python.org/doc/essays/blurb/.

[16] «Oracle,» Oracle, [Ηλεκτρονικό]. Available: https://www.oracle.com/database/what-is-json/.

[17] «docs.python,» [Ηλεκτρονικό]. Available: https://docs.python.org/3/.