ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ VRP ΚΑΙ ΤΟΥ ΣΥΝΔΙΑΣΜΟΥ ΑΛΓΟΡΙΘΜΟΥ ΑΝΑΘΕΣΗΣ ΚΑΙ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ

υπό
ΙΩΑΝΝΗ ΚΑΛΑΚΩΝΑ

Διπλωματική Εργασία

Υπεβλήθη για την εκπλήρωση μέρους των απαιτήσεων για
την απόκτηση του Διπλώματος Μηχανολόγου Μηχανικού

Βόλος, 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ VRP ΚΑΙ ΤΟΥ ΣΥΝΔΙΑΣΜΟΥ ΑΛΓΟΡΙΘΜΟΥ ΑΝΑΘΕΣΗΣ ΚΑΙ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ

υπό
ΙΩΑΝΝΗ ΚΑΛΑΚΩΝΑ

Διπλωματική Εργασία

Υπεβλήθη για την εκπλήρωση μέρους των απαιτήσεων για
την απόκτηση του Διπλώματος Μηχανολόγου Μηχανικού

Βόλος, 2025

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)	Δρ. Σαχαρίδης Γεώργιος Αναπληρωτής Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Δρ. Παντελής Δημήτριος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Τρίτος Εξεταστής	Δρ. Λυμπερόπουλος Γεώργιος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ VRP ΚΑΙ ΤΟΥ ΣΥΝΔΙΑΣΜΟΥ ΑΛΓΟΡΙΘΜΟΥ ΑΝΑΘΕΣΗΣ ΚΑΙ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ

ΙΩΑΝΝΗΣ ΚΑΛΑΚΩΝΑΣ

Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας, 2024

Επιβλέπων Καθηγητής: Κος. Γεώργιος Σαχαρίδης,
Αναπληρωτής Καθηγητής Μεθόδων Επιχειρησιακής Έρευνας στη Βιομηχανική Διοίκηση

Περίληψη

Η διπλωματική εργασία επικεντρώνεται στη συγκριτική μελέτη του αλγορίθμου δρομολόγησης οχημάτων (Vehicle Routing Problem - VRP) και του συνδυασμού του αλγορίθμου ανάθεσης με το πρόβλημα του πλανόδιου πωλητή (Travelling Salesman Problem - TSP) για την επίλυση του προβλήματος δρομολόγησης οχημάτων με περιορισμούς χωρητικότητας (Capacitated Vehicle Routing Problem - CVRP). Περιγράφονται οι βασικές αρχές των VRP και CVRP, οι πρακτικές τους εφαρμογές στη βιομηχανία, και οι δυσκολίες επίλυσης που προκύπτουν λόγω της αυξημένης πολυπλοκότητας των προβλημάτων αυτών.

Η εργασία εξετάζει αναλυτικά τις δύο μεθόδους επίλυσης. Για την υλοποίηση των αλγορίθμων έχει αναπτυχθεί κώδικας σε γλώσσα προγραμματισμού Python, ο οποίος εφαρμόζει τις μεθόδους επίλυσης με τη χρήση της βιβλιοθήκης OR-Tools, που έχει αναπτυχθεί από την Google. Η εργασία παρουσιάζει τον τρόπο αξιοποίησης του OR-Tools για την επίλυση προβλημάτων βελτιστοποίησης, με λεπτομερή περιγραφή της διαδικασίας ανάπτυξης κώδικα για τη διαχείριση του CVRP με τη χρήση του συγκεκριμένου εργαλείου.

Τέλος, περιγράφεται η απόδοση των διαφορετικών αλγορίθμων για την επίλυση του CVRP, μέσα από τη δοκιμή τους σε δέκα διαφορετικά προβλήματα. Τα αποτελέσματα αναδεικνύουν την αποτελεσματικότητα και τις διαφορές μεταξύ των προσεγγίσεων, προσφέροντας μια συγκριτική εικόνα της αποδοτικότητας τους σε ρεαλιστικά σενάρια.

COMPARATIVE STUDY OF THE VRP ALGORITHM AND THE COMBINATION OF THE ASSIGNMENT AND TRAVELING SALESMAN ALGORITHM

IOANNIS KALAKONAS

Department of Mechanical Engineering, University of Thessaly, 2024

Supervisor: George Saharidis
Associate Professor of Methods of Operations Research in Industrial Management

Abstract

The thesis focuses on the comparative study of the Vehicle Routing Problem (VRP) algorithm and the combination of the assignment algorithm with the Traveling Salesman Problem (TSP) to solve the Capacitated Vehicle Routing Problem (CVRP). It describes the fundamental principles of VRP and CVRP, their practical applications in industry, and the challenges of solving these problems due to their increased complexity.

The thesis thoroughly analyzes the two solution methods. A code has been developed in Python to implement the algorithms, applying the solution methods using the OR-Tools library developed by Google. The thesis presents how to utilize OR-Tools for solving optimization problems, with a detailed description of the coding process for managing CVRP using this specific tool.

Finally, it describes the performance of the different algorithms in solving CVRP through testing on ten different problems. The results highlight the effectiveness and differences between the approaches, providing a comparative view of their efficiency in realistic scenarios.

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Εικόνων

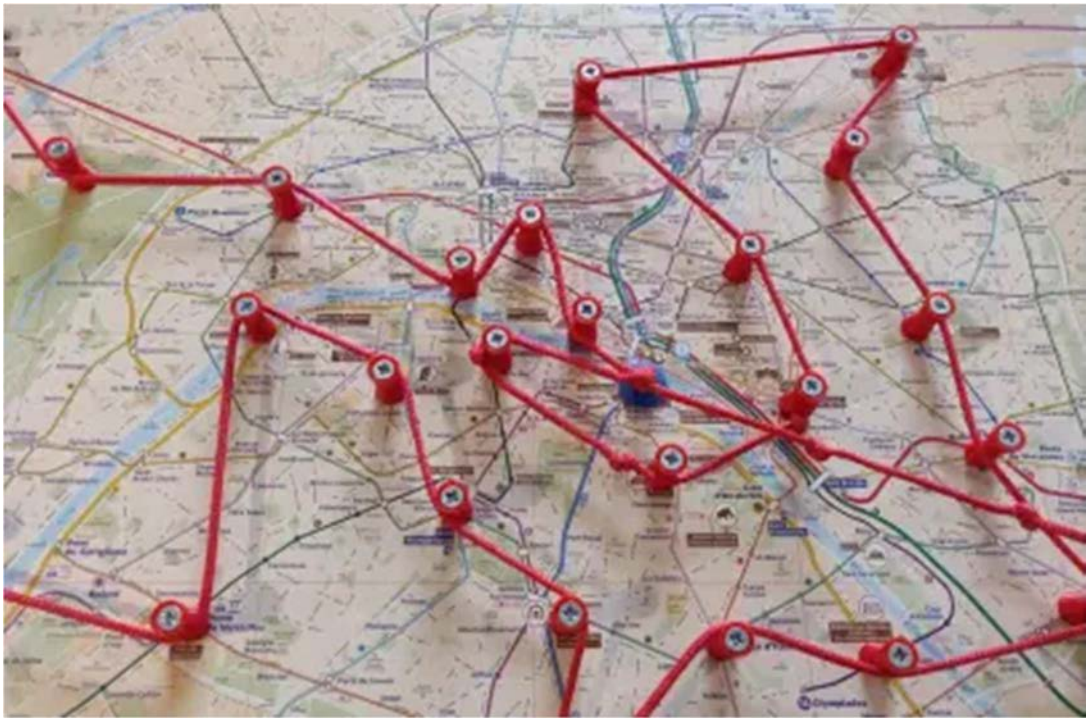
1. Εισαγωγή.

1.1 Επισκόπηση της διατριβής.

Στον σύγχρονο κόσμο, οι επιχειρήσεις και οι βιομηχανίες αντιμετωπίζουν όλο και περισσότερο την πρόκληση της βελτιστοποίησης των logistics σε ένα παγκοσμιοποιημένο περιβάλλον. Μια κρίσιμη πτυχή της διαχείρισης των logistics είναι η δρομολόγηση και ο προγραμματισμός των οχημάτων, η οποία περιλαμβάνει τη μεταφορά αγαθών σε πολλαπλές τοποθεσίες με ταυτόχρονη ελαχιστοποίηση του κόστους, του χρόνου και της χρήσης πόρων. Η πρόκληση αυτή πλαισιώνεται από το πρόβλημα δρομολόγησης οχημάτων (Vehicle Routing Problem - VRP), ένα εξέχον πρόβλημα συνδυαστικής βελτιστοποίησης.

Η παρούσα διατριβή εξετάζει δύο προσεγγίσεις για την επίλυση διαφορετικών μορφών του VRP: τον αλγόριθμο δρομολόγησης οχημάτων και μια μέθοδο που συνδυάζει τον αλγόριθμο ανάθεσης με τον αλγόριθμο πλανόδιου πωλητή. Στόχος είναι να αξιολογηθεί ποια μέθοδος είναι πιο αποτελεσματική όταν εφαρμόζεται στο πρόβλημα δρομολόγησης οχημάτων με χωρητικότητα (Capacitated Vehicle Routing Problem - CVRP), μια διαδεδομένη παραλλαγή του VRP.

Σε αυτή την εισαγωγή, θα διερευνηθούν οι βασικές αρχές του VRP, η ιστορική του εξέλιξη και οι ποικίλες εφαρμογές που έχει σε διάφορες βιομηχανίες. Επιπλέον, θα παρουσιαστεί το CVRP, περιγράφοντας λεπτομερώς πώς σχετίζεται με το κλασικό Πρόβλημα Πλανόδιου Πωλητή (TSP). Θα εμβαθύνθει επίσης η δεύτερη μέθοδος που συγχωνεύει τον αλγόριθμο ανάθεσης με τον αλγόριθμο πλανόδιου πωλητή, θέτοντας τις βάσεις για τη σύγκριση μεταξύ των δύο μεθόδων στα επόμενα κεφάλαια.



Εικόνα 1.1 VRP πρόβλημα

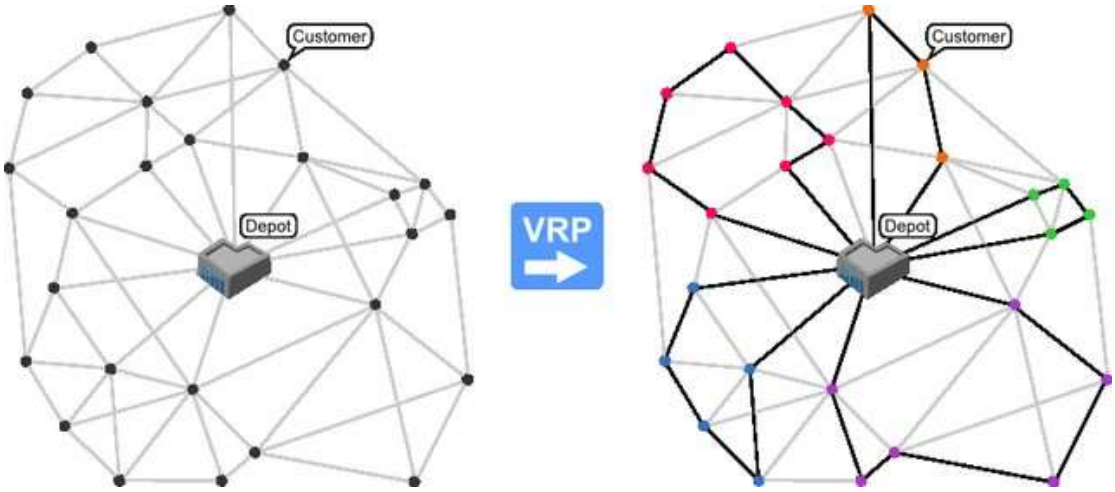
1.2 Ορισμός του VRP.

Το πρόβλημα δρομολόγησης οχημάτων (Vehicle Routing Problem - VRP) αποτελεί ένα λογιστικό πρόβλημα που επικεντρώνεται στον προγραμματισμό δρομολογίων οχημάτων. Σκοπός είναι η ανεύρεση του αποδοτικότερου τρόπου δρομολόγησης ενός στόλου οχημάτων για την εξυπηρέτηση ενός συνόλου πελατών. Μέτρο της αποδοτικότητάς καθίσταται το συνολικό κόστος των διαδρομών, το οποίο μπορεί να περιλαμβάνει παράγοντες όπως το κόστος καυσίμου, τον χρόνο παράδοσης καθώς και τα διάφορα λειτουργικά έξοδα.

Το VRP αποτελεί, στην πραγματικότητα, μια γενίκευση του προβλήματος του πλανόδιου πωλητή (TSP), το οποίο αναζητά την συντομότερη διαδρομή που μπορεί να ακολουθήσει ένας πωλητής για να επισκεφθεί ένα σύνολο πελατών που βρίσκονται σε διαφορετικές πόλεις (ή σημεία). Ο πωλητής του συγκεκριμένου προβλήματος προβλέπεται να περάσει από κάθε πόλη (ή σημείο) ακριβώς μια φορά και να επιστρέψει στην αρχική του θέση μετά την εξυπηρέτηση όλων των πελατών. Αντίστοιχα, το VRP απαιτεί κάθε πελάτης να προσεγγίζεται μονάχα μία φορά για την κάλυψη της παραγγελίας του και κάθε φορτηγό να επιστρέφει πάντα στην κεντρική

αποθήκη (depot) μετά την ολοκλήρωση της προβλεπόμενης διαδρομής του. Επισημαίνεται, ακόμη, ότι ο κάθε πελάτης μπορεί να έχει συγκεκριμένες απαιτήσεις παράδοσης, οι οποίες να επηρεάζουν τον προγραμματισμό των διαδρομών καθώς και τον καταμερισμό των παραγγελιών στα οχήματα.

Η πρόκληση του προβλήματος VRP έγκειται στο σχεδιασμό δρομολογίων που είναι οικονομικά αποδοτικά, ενώ παράλληλα τηρούν περιορισμούς του. Η αποτελεσματική διαχείριση των πόρων απαιτεί στρατηγική σχεδίαση και ορθή αξιοποίηση των διαφορετικών εργαλείων επίλυσης που διατίθενται στην αγορά. Αξίζει να αναφερθεί ότι το VRP ταξινομείται ως NP-hard πρόβλημα, κατηγοριοποίηση που αναφέρεται στην κλάση πολυπλοκότητας του προβλήματος (μη ντετερμινιστική πολυωνυμική δυσκολία), το οποίο υποδηλώνει ότι δεν υπάρχει γνωστός αλγόριθμος που να μπορεί να το λύσει σε πολυωνυμικό χρόνο για όλα τα δυνατά σενάρια. Αυτό σημαίνει ότι καθώς το μέγεθος του προβλήματος (π.χ. ο αριθμός των πελατών ή/και των οχημάτων) αυξάνεται, ο χρόνος επίλυσης και η πολυπλοκότητά του αυξάνονται εκθετικά.



Εικόνα 1.2 Ορισμός του VRP

1.3 Πραγματικές εφαρμογές του VRP.

Το πρόβλημα VRP εφαρμόζεται σε μεγάλο εύρος κλάδων της βιομηχανίας, ιδιαίτερα στους τομείς των μεταφορών, των logistics και της διαχείριση εφοδιαστικής αλυσίδας. Η σημαντικότητα και η χρησιμότητά του αναδεικνύονται σε περιπτώσεις

όπου απαιτείται η εκπλήρωση πολλαπλών παραδόσεων ή/και η παροχή πολλαπλών υπηρεσιών από έναν στόλο οχημάτων.

Εικόνα 1.3 Πραγματικές εφαρμογές του VRP

- Διανομή Προϊόντων και Διαχείριση Εφοδιαστικής Αλυσίδας

Ο κλάδος της εφοδιαστικής αλυσίδας συνιστά έναν από τους κυριότερους τομείς εφαρμογής του VRP. Εταιρείες λιανικής πώλησης, ηλεκτρονικού εμπορίου, logistics καθώς και ταχυδρομικές υπηρεσίες χρησιμοποιούν το VRP για την βελτιστοποίηση των διαδρομών που ακολουθεί ο στόλος των οχημάτων τους.

Ένα χαρακτηριστικό παράδειγμα αποτελεσματικής εφαρμογής του VRP στον συγκεκριμένο κλάδο αποτελεί το πρόγραμμα Amazon Prime της εταιρείας Amazon. Το εν λόγω πρόγραμμα αξιοποιεί εξελιγμένους αλγόριθμους VRP για να μειώσει τα κόστη μεταφοράς, να αυξήσει την αποδοτικότητα του συστήματος δρομολόγησης και να εξασφαλίσει γρήγορες παραδόσεις. Επιπλέον, η εταιρεία έχει βασιστεί στο VRP για την διαχείριση των παραγγελιών τελευταίας διαδρομής, με στόχο την μείωση των εξόδων και την αποτελεσματική και ταχεία εξυπηρέτηση πελατών σε πολυπληθείς περιοχές.

Αξιοσημείωτη εφαρμογή του προβλήματος αποτελεί και το σύστημα ORION (On-Road Integrated Optimization and Navigation) το οποίο αναπτύχθηκε από την εταιρεία UPS βασιζόμενο στις αρχές του VRP. Στόχος του συστήματος είναι να προγραμματίσει τις διαδρομές παράδοσης στις ΗΠΑ με τέτοιο τρόπο ώστε να αποφεύγονται όσο τον δυνατόν περισσότερο οι αριστερές στροφές, οι οποίες συχνά αυξάνουν τους χρόνους αναμονής και κατά συνέπεια την κατανάλωση καυσίμων.

- Δημόσιες υπηρεσίες μεταφορών

Το VRP χρησιμοποιείται, ακόμη, σε πληθώρα δημόσιων υπηρεσιών μεταφορών, βελτιώνοντας την ποιότητα των προσφερόμενων υπηρεσιών τους και περιορίζοντας τόσο το κόστος όσο και το περιβαλλοντικό αποτύπωμα.

Ένα αντιπροσωπευτικό παράδειγμα αποτελεί η υιοθέτηση του VRP από διάφορες αστικές συγκοινωνίες ανά τον κόσμο με στόχο την βελτιστοποίηση των διαδρομών και των χρονοδιαγραμμάτων λεωφορείων και τραμ, ειδικά σε συνθήκες αυξημένης κίνησης, περιορισμένων λωρίδων και ωρών αιχμής. Οι δημόσιες συγκοινωνίες στη Λυών και το Παρίσι αξιοποιούν «έξυπνα» συστήματα ελέγχου της κυκλοφορίας, τα οποία χρησιμοποιούν δεδομένα VRP για να μειώσουν τον χρόνο διαδρομής των επιβατών και την εκπομπή των ρύπων.

Αντίστοιχη χρήση του VRP συναντάται και στα συστήματα δρομολόγησης σχολικών λεωφορείων, ιδιαίτερα σε αγροτικές ή μεγάλες αστικές περιοχές, όπου η μείωση των λειτουργικών εξόδων κρίνεται ύψιστης σημασίας. Αξίζει να αναφερθεί, τέλος, και η εφαρμογή του σε υπηρεσίες παροχής κοινόχρηστων οχημάτων, όπως ποδήλατα και ηλεκτρικά σκούτερ, στις οποίες λειτουργεί ως μέσο εντοπισμού των περιοχών υψηλής ζήτησης με απώτερο σκοπό την στρατηγική διαμόρφωση του δικτύου τοποθέτησης και φόρτισης των οχημάτων.

- Συλλογή και Διαχείριση Απορριμμάτων

Στην αποκομιδή απορριμμάτων, το VRP χρησιμοποιείται για να αναδείξει τον αποτελεσματικότερο τρόπο δρομολόγησης του στόλου των απορριμματοφόρων έτσι

ώστε να εξυπηρετούνται άμεσα όλες οι γειτονιές μιας περιοχής με την χαμηλότερη δυνατή κατανάλωση καυσίμου.

- Υπηρεσίες έκτακτης ανάγκης

Υψηλής σημασίας είναι και η ενσωμάτωση του VRP στις υπηρεσίες έκτακτης ανάγκης, όπως τα ασθενοφόρα και οι πυροσβεστικές υπηρεσίες. Με ορθή εφαρμογή της μεθόδου, τα οχήματα διαμοιράζονται αποτελεσματικότερα στις τοποθεσίες επειγόντων περιστατικών και χρησιμοποιούν ταχύτερες διαδρομές, γεγονός που μπορεί να σώσει ζωές.

Με την αποτελεσματική επίλυση του VRP, οι επιχειρήσεις μπορούν να μειώσουν σημαντικά τα λειτουργικά κόστη, την κατανάλωση καυσίμων και να βελτιώσουν την εμπειρία των πελατών τους, γεγονός που αναδεικνύει τη σημασία του προβλήματος σε πραγματικές εφαρμογές.

1.4 Ιστορική αναδρομή του VRP.

Η ιστορική αναδρομή του προβλήματος VRP δίνει μια αντιπροσωπευτική εικόνα της γενικότερης εξέλιξης των μεθόδων βελτιστοποίησης, οι οποίες αναπτύχθηκαν με την πάροδο του χρόνου, ανταγωνιζόμενες την ολοένα και μεγαλύτερη ανάγκη της βιομηχανίας για καλύτερη διαχείριση των μεταφορών.

Η έννοια του προβλήματος της δρομολόγησης οχημάτων παρουσιάστηκε για πρώτη φορά σε εργασία των George Dantzig και John Ramser το 1959. Η εργασία εστίαζε στο πρόβλημα της δρομολόγησης φορτηγών διανομής καυσίμου σε βενζινάδικα και χρησιμοποιούσε μια απλή αλγοριθμική προσέγγιση για την βελτιστοποίηση του συνόλου των διαδρομών. Ο αλγόριθμος αυτός έθεσε τα θεμέλια για τη μελλοντική έρευνα του προβλήματος VRP καθώς και των ποικίλλων παραλλαγών του.

Κατά τις δεκαετίες του 1960 και 1970, οι υπολογιστικές δυνατότητες βελτιώθηκαν σημαντικά, δίνοντας, έτσι, την ευκαιρία στους ερευνητές να αναπτύξουν ευρετικούς αλγορίθμους για την αντιμετώπιση του VRP. Οι ευρετικές αυτές μέθοδοι προσέφεραν

πρακτικές, σχεδόν βέλτιστες λύσεις, μέσα σε αποδεκτά χρονικά πλαίσια, γεγονός που μετρίασε την αδυναμία ανάδειξης ακριβών λύσεων για περιπτώσεις προβλημάτων μεγάλης κλίμακας.

Η ενίσχυση της πολυπλοκότητας των logistics και των αλυσίδων εφοδιασμού την δεκαετία του 1980 είχε ως αποτέλεσμα τον σχηματισμό διαφόρων παραλλαγών του προβλήματος VRP, που στόχο είχαν να ανταποκριθούν στις ειδικές απαιτήσεις του κλάδου. Μερικά παραδείγματα αυτών είναι το Capacitated VRP (CVRP), το VRP με χρονικά παράθυρα (TWVRP) και το στοχαστικό VRP. Η πρόοδος στην θεωρία των γραφημάτων και η ανάπτυξη υπολογιστικών μεθόδων, όπως οι γενετικοί αλγόριθμοι και οι αλγόριθμοι τεχνητής νοημοσύνης, τις δεκαετίες του 1990 και 2000, εισήγαγαν νέους τρόπους προσέγγισης του προβλήματος που κατέστησαν την επίλυσή του πιο προσιτή για πληθώρα επιχειρήσεων.

Στην σημερινή εποχή, το VRP παραμένει μια δυναμική ερευνητική περιοχή, με ακριβείς και μεταευρετικούς αλγορίθμους να χρησιμοποιούνται για την επίλυση ολοένα και πιο σύνθετων πραγματικών περιπτώσεων του προβλήματος.

1.5 Παραλλαγές του προβλήματος VRP.

Στο παρών κεφάλαιο, θα παρουσιαστούν συνοπτικά οι κυριότερες παραλλαγές του προβλήματος δρομολόγησης οχημάτων, οι επιπρόσθετοι περιορισμοί και οι δυσκολίες που εντάσσουν στο πρόβλημα καθώς και οι τομείς της βιομηχανίας στους οποίους βρίσκει η καθεμία την μεγαλύτερη πρακτική εφαρμογή.

- Πρόβλημα δρομολόγησης οχημάτων με περιορισμένη χωρητικότητα (CVRP)

Το πρόβλημα δρομολόγησης οχημάτων με περιορισμένη χωρητικότητα (CVRP) αποτελεί μια ειδική παραλλαγή του VRP, η οποία θέτει τον επιπρόσθετο περιορισμό της χωρητικότητας των οχημάτων. Στο CVRP, κάθε όχημα έχει ένα μέγιστο φορτίο που μπορεί να μεταφέρει και η συνολική ζήτηση που καλύπτεται σε κάθε διαδρομή δεν μπορεί να υπερβαίνει αυτή τη χωρητικότητα. Ο επιπλέον αυτός περιορισμός αυξάνει την πολυπλοκότητα του προβλήματος, απαιτώντας τόσο τη βελτιστοποίηση της

αποδοτικότητας της διαδρομής όσο και την εξισορρόπηση του φορτίου μεταξύ των οχημάτων.

Όπως και στο VRP, στόχος του CVRP είναι η ελαχιστοποίηση του κόστους μεταφοράς, το οποίο συνήθως μετριέται σε όρους συνολικής απόστασης ή χρόνου ταξιδιού. Ο περιορισμός χωρητικότητας σημαίνει, ωστόσο, ότι η λύση θα πρέπει ταυτόχρονα να διασφαλίζει ότι κανένα όχημα δεν θα υπερβαίνει το όριο φορτίου του. Το CVRP είναι ιδιαίτερα σημαντικό σε κλάδους όπου το ωφέλιμο φορτίο των οχημάτων είναι περιορισμένο, όπως στις μεταφορές εμπορευμάτων και στις υπηρεσίες διανομής.

- Πρόβλημα δρομολόγησης οχημάτων με παράθυρα χρόνου (VRPTW)

Στην συγκεκριμένη παραλλαγή, προστίθεται στο πρόβλημα VRP ο περιορισμός της εξυπηρέτησης των πελατών εντός προκαθορισμένων χρονικών παραθύρων. Αυτό σημαίνει ότι η δρομολόγηση των οχημάτων λαμβάνει υπόψη συγκεκριμένα χρονικά πλαίσια μέσα στα οποία πρέπει να παραλαμβάνονται ή να παραδίδονται τα φορτία από τα αρμόδια οχήματα. Όπως γίνεται αντιληπτό, η υποκατηγορία αυτή βρίσκει ιδιαίτερη εφαρμογή σε υπηρεσίες που απαιτούν ακρίβεια στον χρόνο παράδοσης, όπως είναι οι ταχυμεταφορές.

- Πρόβλημα δρομολόγησης οχημάτων με πολλαπλές αποθήκες (MDVRP)

Η διαφοροποίηση με το πρόβλημα VRP έγκειται, στην συγκεκριμένη περίπτωση, στην ύπαρξη πολλαπλών αποθηκών ή σημείων εκκίνησης των οχημάτων (depots). Στο MDVRP, ο προγραμματισμός των δρομολογίων περιλαμβάνει, κατά συνέπεια, και την επιλογή της αποθήκης από την οποία κρίνεται βέλτιστο να ξεκινήσει το κάθε όχημα για την ελαχιστοποίηση του συνολικού κόστους. Η προσθήκη μιας επιπλέον παραμέτρου απόφασης στο πρόβλημα μπορεί να δυσχεραίνει την επίλυσή του, καθίσταται όπως απαραίτητη για την εφαρμογή του VRP σε εταιρείες που διαθέτουν μεγάλα δίκτυα αποθηκών και κέντρων διανομής.

- Πρόβλημα δρομολόγησης οχημάτων με διαχωρισμό παραδόσεων (SDVRP)

Η παραλλαγή αυτή εισάγει στο πρόβλημα την δυνατότητα εξυπηρέτησης ενός πελάτη από περισσότερα από ένα οχήματα για την ικανοποίηση της παραγγελίας του, εάν

αυτό κρίνεται απαραίτητο. Αποδεικνύεται ιδιαίτερα χρήσιμη όταν ο όγκος των παραγγελιών που λαμβάνει μια εταιρεία είναι μεγάλος ή στην περίπτωση που το πλήθος των διαθέσιμων οχημάτων είναι περιορισμένο.

- Πρόβλημα δρομολόγησης οχημάτων με παραλαβή και παράδοση (VRPPD)

Σε αυτή την επέκταση του προβλήματος, κάθε πελάτης χαρακτηρίζεται, εκτός από την ζήτηση που του αντιστοιχεί, και από το ποσό των εμπορευμάτων που πρέπει να συλλεχθούν από αυτόν. Τα αρμόδια οχήματα προβλέπεται, με άλλα λόγια, να μεταφέρουν εμπορεύματα από και προς διαφορετικές τοποθεσίες, προσαρμόζοντας την διαδρομή τους ανάλογα. Η δυνατότητα αμφίδρομης κίνησης των φορτίων καθιστά το VRPPD ιδιαίτερα χρήσιμο σε υπηρεσίες logistics, στις οποίες τα οχήματα πρέπει να καλύπτουν πολλές στάσεις για παραλαβή και παράδοση.

- Στοχαστικό πρόβλημα δρομολόγησης οχημάτων (SVRP)

Στο στοχαστικό πρόβλημα δρομολόγησης οχημάτων, κάποιες παράμετροι, όπως η ζήτηση των πελατών ή οι χρόνοι εξυπηρέτησης, θεωρούνται άγνωστες κατά τον σχεδιασμό της διαδρομής ή, εναλλακτικά, θεωρείται ότι καθορίζονται τυχαία. Η ενσωμάτωση της έννοιας της τυχαιότητας στο VRP, αν και δυσκολεύει στην επίλυσή του, αποδεικνύεται χρήσιμη σε βιομηχανίες που χαρακτηρίζονται από μεγάλη αβεβαιότητα στις παραδόσεις ή/και στην ζήτηση, όπως οι εταιρείες τροφοδοσίας καταστημάτων λιανικής πώλησης και οι εταιρείες διανομής σε μεγάλα αστικά κέντρα μεταβλητής κυκλοφοριακής συμφόρησης.

- Πρόβλημα δρομολόγησης ηλεκτρικών οχημάτων (EVRP)

Η είσοδος των ηλεκτρικών οχημάτων στην αγορά έδωσε αφορμή για την ανάπτυξη της συγκεκριμένης παραλλαγής, η οποία ουσιαστικά επεκτείνει το πρόβλημα VRP έτσι ώστε να λαμβάνει υπόψη την ανάγκη για φόρτιση των ηλεκτρικών οχημάτων στους ειδικά διαμορφωμένους σταθμούς που θα συναντήσουν κατά την διάρκεια της επιλεγμένης διαδρομής τους.

- Πρόβλημα δρομολόγησης οχημάτων και διαχείρισης αποθεμάτων (IRP)

Το πρόβλημα αυτό αφορά στον συνδυασμό δύο βασικών λειτουργιών της εφοδιαστικής αλυσίδας: τη διαχείριση αποθεμάτων και τη δρομολόγηση οχημάτων.

Μία εταιρεία που εφαρμόζει το IRP, οφείλει να αποφασίσει το πότε και πόσα αποθέματα θα παραδώσει στους πελάτες της, λαμβάνοντας υπόψη τον διαφορετικό ρυθμός εξάντλησης των αποθεμάτων του κάθε πελάτη. Η διαχείριση των αποθεμάτων καθορίζει την συχνότητα των δρομολογίων, ενώ η βελτιστοποίηση των δρομολογίων, επηρεάζει, με την σειρά της, το κόστος και την αποδοτικότητα της διαχείρισης αποθεμάτων. Το IRP εφαρμόζεται, κυρίως, σε βιομηχανίες στις οποίες η συστηματική ανατροφοδότηση προϊόντων κρίνεται μεγάλης σημασίας, όπως είναι τα βενζινάδικα και τα σούπερ μάρκετ.

1.6 Η σύνδεση μεταξύ CVRP και TSP.

Το CVRP συνδέεται στενά με το TSP, το οποίο είναι ένα από τα πιο μελετημένα προβλήματα συνδυαστικής βελτιστοποίησης. Στο TSP, ο στόχος είναι να βρεθεί η συντομότερη δυνατή διαδρομή για έναν πωλητή που θα επισκεφθεί όλες τις πόλεις ακριβώς μία φορά και θα επιστρέψει στην πόλη εκκίνησης.

Το CVRP επεκτείνει το TSP με τη συμμετοχή πολλαπλών οχημάτων, το καθένα με όριο χωρητικότητας. Αντί για την εύρεση μιας διαδρομής για έναν μόνο πωλητή, το CVRP απαιτεί πολλαπλά οχήματα για την εξυπηρέτηση των πελατών, ελαχιστοποιώντας τη συνολική απόσταση που διανύεται και τηρώντας τους περιορισμούς χωρητικότητας. Η στενή σύνδεση μεταξύ CVRP και TSP είναι σημαντική, διότι πολλές τεχνικές που αναπτύχθηκαν για την TSP μπορούν να προσαρμοστούν για την επίλυση του CVRP. Για παράδειγμα, ο αλγόριθμος ανάθεσης και ο αλγόριθμος πλανόδιου πωλητή, οι οποίοι εξετάζονται στην παρούσα διατριβή, βασίζονται στις αρχές του TSP για να αντιμετωπίσουν τις πρόσθετες πολυπλοκότητες που εισάγονται από τις χωρητικότητες των οχημάτων στο CVRP.

1.7 Επισκόπηση της εναλλακτικής μεθόδου: Αλγόριθμος ανάθεσης και Πλανόδιου πωλητή.

Η παρούσα διατριβή διερευνά μια εναλλακτική μέθοδο επίλυσης του CVRP, η οποία συνδυάζει τον αλγόριθμο ανάθεσης με τον αλγόριθμο περιπλανώμενου πωλητή. Η προσέγγιση αυτή αποτελείται από δύο βασικά βήματα: πρώτα, οι πελάτες αντιστοιχίζονται σε οχήματα χρησιμοποιώντας τον αλγόριθμο ανάθεσης και στη συνέχεια, η βέλτιστη διαδρομή για κάθε όχημα υπολογίζεται χρησιμοποιώντας αρχές από το TSP. Ο συνδυασμός αυτών των δύο τεχνικών επιτρέπει τον αποτελεσματικό σχεδιασμό διαδρομών με σεβασμό στον περιορισμό χωρητικότητας των οχημάτων.

1.7.1 Ο αλγόριθμος ανάθεσης.

Ο αλγόριθμος ανάθεσης είναι μια κλασική τεχνική βελτιστοποίησης που χρησιμοποιείται για την ανάθεση εργασιών σε πράκτορες με τρόπο που ελαχιστοποιεί το συνολικό κόστος ή μεγιστοποιεί την αποδοτικότητα. Στο πλαίσιο της δρομολόγησης οχημάτων, ο αλγόριθμος αυτός χρησιμοποιείται για την κατανομή των πελατών στα οχήματα, διασφαλίζοντας ότι κανένα όχημα δεν υπερβαίνει τη χωρητικότητά του και ότι κάθε πελάτης θα εξυπηρετηθεί από ένα και μόνο όχημα.

Η ανάθεση πελατών στα οχήματα μπορεί να αναπαρασταθεί μαθηματικά ως ένα πρόβλημα βελτιστοποίησης, όπου ο στόχος είναι να ελαχιστοποιηθεί το συνολικό κόστος εξυπηρέτησης. Στη περίπτωση του CVRP το κόστος εξυπηρέτησης προκύπτει από την απόσταση που διανύουν τα οχήματα. Επομένως έχει μεγάλη σημασία το πως θα κατανεμηθούν τα οχήματα στους πελάτες. Αυτό επιτυγχάνεται με την επίλυση ενός προβλήματος ανάθεσης, το οποίο περιλαμβάνει τη λήψη αποφάσεων σχετικά με το ποιους πελάτες θα αναλάβει το κάθε όχημα, υπό τους περιορισμούς ότι κανένα όχημα δεν θα μεταφέρει φορτίο που υπερβαίνει τη χωρητικότητά του και κάθε πελάτης θα εξυπηρετείτε από ένα και μόνο ένα όχημα.

Σε ότι αφορά τους περιορισμούς το πρόβλημα ανάθεσης μπορεί να ανταποκριθεί κατάλληλα. Οι περιορισμοί μπορούν να αναπαρασταθούν ευκολά ως γραμμικές ανισότητες γεγονός που βοηθάει ιδιαίτερα στη μοντελοποίηση. Αυτό σημαίνει ότι εφόσον ικανοποιούνται οι περιορισμοί του προβλήματος ανάθεσης, ο αλγόριθμος επιστέφει εφικτή λύση.

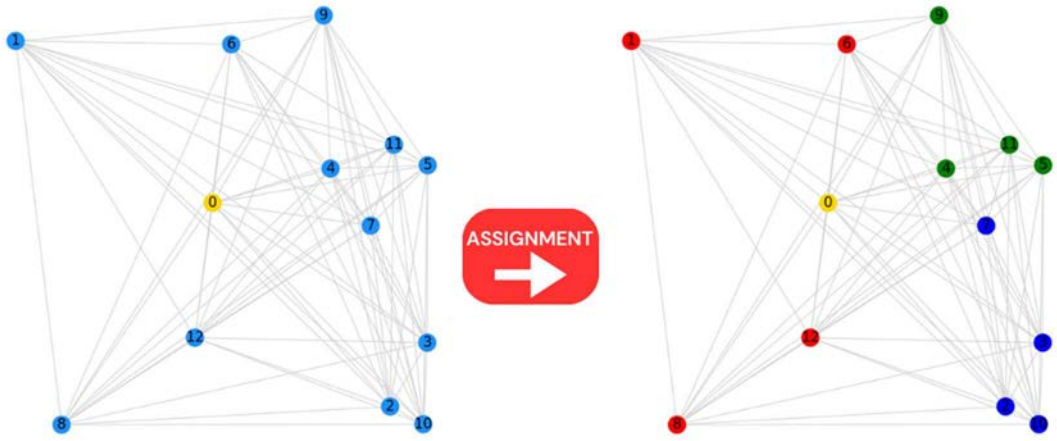
Το ίδιο δεν μπορεί να ειπωθεί για τη βελτιστότητα της λύσης. Όπως σε όλα τα προβλήματα γραμμικού προγραμματισμού, η βέλτιστη λύση του προβλήματος ανάθεσης βρίσκεται από μια γραμμική σχέση (αντικειμενική συνάρτηση) η οποία πρέπει να ελαχιστοποιηθεί ή να μεγιστοποιηθεί. Ο αλγόριθμος ανάθεσης για το CVRP καλείται να ελαχιστοποιήσει την απόσταση που πρέπει να διανύσουν τα οχήματα. Αυτό αποδεικνύεται εξαιρετικά δύσκολο καθώς μέχρι στιγμής δεν έχει βρεθεί κάποια γραμμική σχέση η οποία να ελαχιστοποιεί την απόσταση που θα διανύσουν τα οχήματα, χωρίς να περιγράφει την απόσταση αυτή. Αυτό ακριβώς επιλύει το πρόβλημα CVRP και εδώ πρέπει να υπενθυμίσουμε, ότι το πρόβλημα που προκύπτει είναι εξαιρετικά δύσκολο στην επίλυση του. Συμπεραίνεται, λοιπόν, ότι σε αυτόν τον αλγόριθμο ανάθεσης θα πρέπει η ανάθεση να γίνει με τέτοιο τρόπο ώστε όταν επιλυθούν και οι αλγόριθμοι TSP στο επόμενο σκέλος, να παραχθεί η βέλτιστη λύση του CVRP.

Στη προσπάθεια να αποδειχθεί ότι είναι δυνατό να βρεθούν έστω και εφικτές λύσεις για προβλήματα CVRP μέσω του αλγορίθμου ανάθεσης, έγινε αρχικώς μια παραδοχή. Αυτή ήταν να ‘’αγνοηθεί’’ η αντικειμενική συνάρτηση του προβλήματος ανάθεσης ,δηλαδή να μην οριστούν κόστη ή ποινές για την ανάθεση οχημάτων στους πελάτες. Αυτό, όπως γίνεται αντιληπτό, οδηγεί σε πολύ ακριβείς λύσεις και κατά συνέπεια δεν δύναται να εφαρμοστεί σε πραγματικά προβλήματα. Παρόλα αυτά οι συνθήκες της εφικτότητας, επιβεβαιώνονται και ο αλγόριθμος επιστρέφει εφικτές λύσεις. Στη συνέχεια της διατριβής παρουσιάζεται με μεγαλύτερη λεπτομέρεια και αναλύονται τα αποτελέσματα που επιστρέφει.

Εφόσον αποδείχθηκε ότι ο αλγόριθμος αποδίδει εφικτές λύσεις έπρεπε να αντιμετωπιστεί και το αδύναμο σημείο του, η ποιότητα της λύσης. Αυτό έγινε χρησιμοποιώντας την μεθοδο της συσταδοποίησης. Αρχικά δημιουργείται ένας αριθμός συστάδων ίσος με τον αριθμό των οχημάτων, με στόχο κάθε συστάδα να εξυπηρετηθεί από ένα ξεχωριστό όχημα. Με αυτό το τέχνασμα αποφεύγονται άσκοπα δρομολόγια, για παράδειγμα δυο γειτονικοί προορισμοί να εξυπηρετούνται από διαφορετικά οχήματα. Ο αλγόριθμος ανάθεσης το πετυχαίνει αυτό, ελαχιστοποιώντας την συνολική απόσταση μεταξύ του κάθε προορισμού με το κέντρο της συστάδας που του ανατέθηκε. Όπως αναφέρθηκε και νωρίτερα κάθε συστάδα

αντιπροσωπεύει ένα όχημα, επομένως όταν γίνεται ανάθεση συστάδας σε προορισμό ισοδυναμεί με ανάθεση οχήματος σε προορισμό. Επιπλέον εξετάζεται μια παραλλαγή αυτής της μεθόδου όπου οι αποστάσεις μεταξύ συστάδων και προορισμών υψώνονται στο τετράγωνο. Πως ακριβός δουλεύουν αυτοί οι αλγόριθμοι εξετάζεται εκτενέστερα σε επόμενα κεφάλαια.

Οι αλγόριθμοι ανάθεσης ποικίλλουν ανάλογα με την πολυπλοκότητα και το μέγεθος του προβλήματος. Για παράδειγμα, σε μικρά προβλήματα με λίγους πελάτες και οχήματα, ο αλγόριθμος ανάθεσης μπορεί να βασιστεί σε ακριβείς αλγορίθμους, όπως είναι οι αλγόριθμοι γραμμικού προγραμματισμού ή ο αλγόριθμος κλάδου και ορίων (branch and bound). Αυτές οι μέθοδοι παρέχουν την απόλυτα βέλτιστη λύση αλλά συχνά είναι δύσκολο να εφαρμοστούν σε μεγάλα προβλήματα, καθώς ο υπολογιστικός χρόνος αυξάνεται εκθετικά με τον αριθμό των πελατών και των οχημάτων.



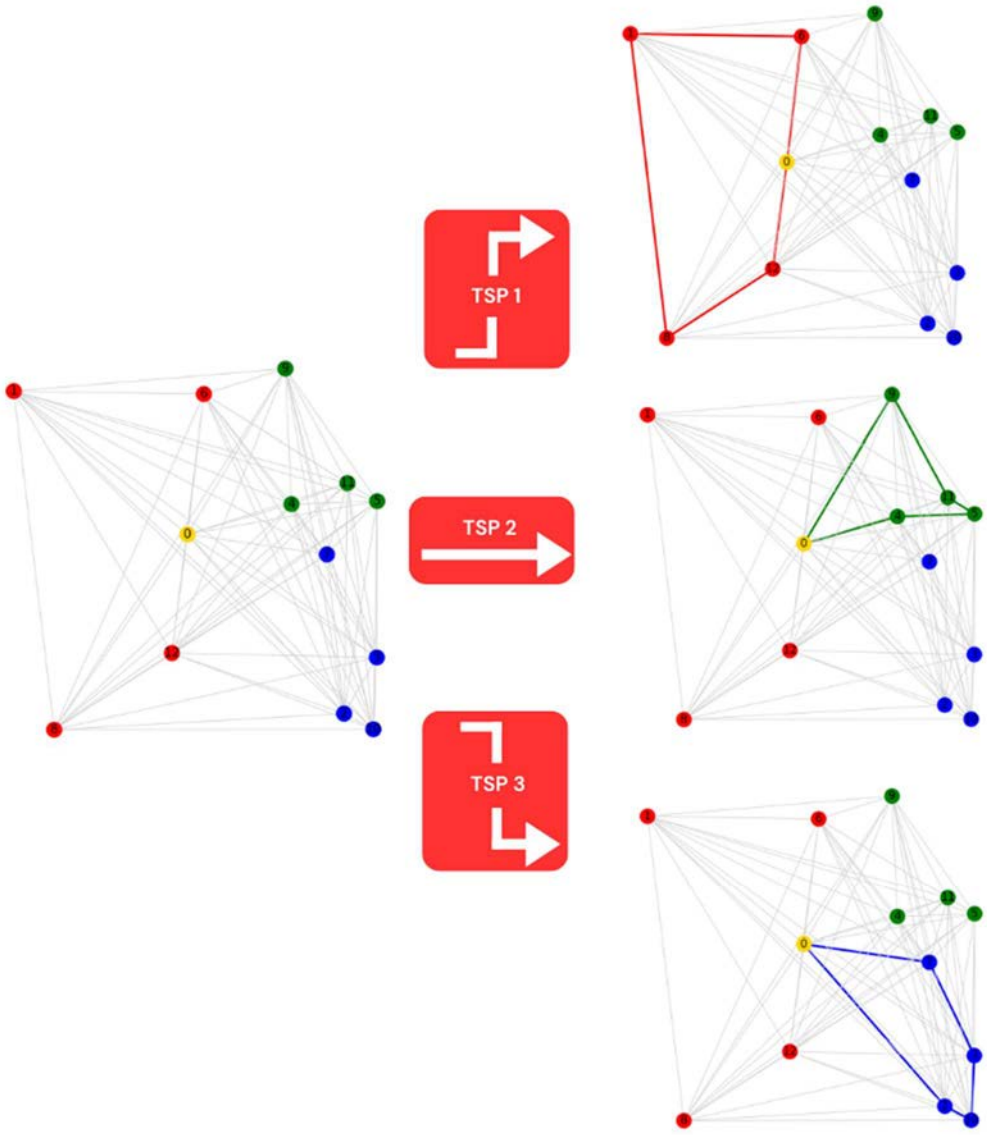
Εικόνα 1.7.1 Αλγόριθμος ανάθεσης

1.7.2 Η προσέγγιση του προβλήματος TSP.

Μόλις οι πελάτες ανατεθούν στα οχήματα, το επόμενο βήμα είναι η εύρεση της συντομότερης δυνατής διαδρομής για κάθε όχημα, η οποία ουσιαστικά αποτελεί επίλυση ενός TSP για κάθε ομάδα πελατών που έχει ανατεθεί. Ο στόχος είναι η

ελαχιστοποίηση της συνολικής απόστασης ταξιδιού για κάθε όχημα, διασφαλίζοντας παράλληλα την αποτελεσματική εξυπηρέτηση όλων των πελατών.

Η λύση του TSP είναι εξαιρετικά δύσκολη από υπολογιστική άποψη, καθώς ανήκει στην κατηγορία των NP-complete προβλημάτων. Αυτό σημαίνει ότι ο χρόνος που απαιτείται για την εξεύρεση της βέλτιστης λύσης αυξάνεται εκθετικά με τον αριθμό των πόλεων. Έτσι, ακόμα και για έναν σχετικά μικρό αριθμό πόλεων, το πλήθος των πιθανών διαδρομών που πρέπει να εξεταστούν, είναι τέτοιο που καθιστά την εύρεση λύσης πρακτικά αδύνατη. Αυτό έρχεται σε παραλληλία με τα προβλήματα VRP και CVRP, καθώς και αυτά είναι υπολογιστικά δυσεπίλυτα και απαιτούν, όπως και το TSP, ιδιαίτερες τεχνικές για την παραγωγή λύσης.



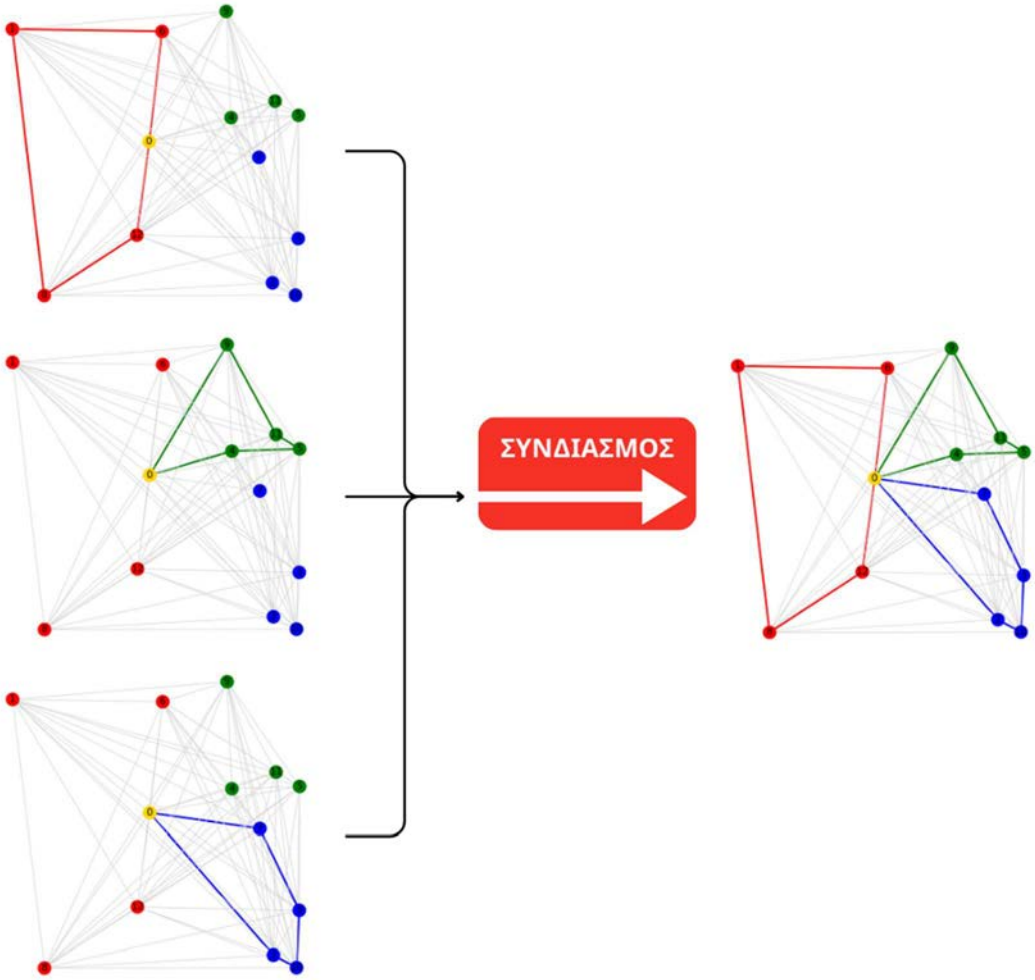
Εικόνα 1.7.2 TSP για κάθε όχημα

Για την προσέγγιση της λύσης για προβλήματα TSP, έχουν αναπτυχθεί διάφορες μέθοδοι, όπως ακριβείς και προσεγγιστικές. Στις ακριβείς μεθόδους περιλαμβάνεται ο αλγόριθμος κλάδου και ορίων (branch and bound) και ο δυναμικός προγραμματισμός. Οι μέθοδοι αυτές παρέχουν τη βέλτιστη λύση αλλά είναι υπολογιστικά απαιτητικές και περιορίζονται σε μικρότερα προβλήματα λόγω της πολυπλοκότητάς τους. Από την άλλη, οι ευρετικές σε συνδυασμό με τις μεταευρετικές μεθόδους, οι οποίες εφαρμόζονται και σε προβλήματα VRP και CVRP, έχουν αποδειχθεί αρκετά αποτελεσματικές στην εύρεση ικανοποιητικών προσεγγιστικών λύσεων.

1.7.3 Συνδυασμός του προβλήματος ανάθεσης με το TSP για την επίλυση του CVRP.

Ο συνδυασμός του προβλήματος ανάθεσης και του TSP παρέχει μια δομημένη προσέγγιση για την επίλυση του CVRP. Πρώτον, ο αλγόριθμος ανάθεσης κατανέμει τους πελάτες σε συγκεκριμένα οχήματα τηρώντας τους περιορισμούς χωρητικότητας. Στη συνέχεια, επιλύεται ένα TSP για κάθε όχημα, βρίσκοντας τη συντομότερη διαδρομή για την επίσκεψη στους πελάτες που του έχουν ανατεθεί. Η γενική λύση του προβλήματος CVRP παράγεται με τον συνδυασμό των λύσεων που έδωσε κάθε ένα TSP. Αυτή η λύση ανταποκρίνεται στους περιορισμούς του αρχικού προβλήματος CVRP.

Η συνδυαστική μέθοδος αποδεικνύεται αρκετά αποτελεσματική. Σε ορισμένες περιπτώσεις μπορεί να ξεπεράσει σε αποδοτικότητα ακόμα και τις παραδοσιακές μεθόδους για την επίλυση CVRP. Στις περισσότερες περιπτώσεις παρέχει λύσεις ελαφρώς χειρότερες. Αυτό την καθιστά μια αρκετά καλή εναλλακτική επιλογή σε προβλήματα αυξημένης πολυπλοκότητας.



Εικόνα 1.7.3 Συνδυασμός TSP

1.8 Υπολογιστικά εργαλεία επίλυσης VRP

Στη συνέχεια, παρουσιάζονται τα κυριότερα υπολογιστικά εργαλεία που είναι διαθέσιμα στην αγορά για την επίλυση του προβλήματος δρομολόγησης οχημάτων. Τα εργαλεία αυτά ποικίλλουν σε επίπεδο αποδοτικότητας, πολυπλοκότητας καθώς και ταχύτητας, ενώ επεκτείνονται από εμπορικά λογισμικά έως και λύσεις ανοιχτού κώδικα. Ακολουθεί η ανάλυση των πιο διαδεδομένων εξ' αυτών και η σύντομη παρουσίαση των αδυναμιών και των πλεονεκτημάτων τους.

- **CPLEX**: Το IBM ILOG CPLEX αποτελεί ένα από τα δημοφιλέστερα εμπορικά λογισμικά επίλυσης προβλημάτων βελτιστοποίησης και ιδιαίτερα προβλημάτων ακεραίου και γραμμικού προγραμματισμού. Υποστηρίζει

πολλούς αλγορίθμους, όπως Simplex, λογαριθμικού φραγμού και διακλάδωσης και δέσμευσης, ενώ προσφέρει υψηλή ακρίβεια, μεγάλη ταχύτητα και ικανότητα διαχείρισης μεγάλων και πολύπλοκων προβλημάτων. Σημαντικό είναι, ακόμη, ότι υποστηρίζει την εύκολη διαμόρφωση πολλαπλών περιορισμών. Για όλους του παραπάνω λόγους χρησιμοποιείται ευρέως στην βιομηχανία για την επίλυση του VRP και των ποικίλλων παραλλαγών του. Στα μειονεκτήματά του είναι ότι η άδεια χρήσης του δεν παρέχεται δωρεάν, γεγονός που το καθιστά λιγότερο ελκυστικό σε ακαδημαϊκές εφαρμογές και μικρές εταιρείες.

- **Gurobi**: Το Gurobi Optimizer αποτελεί ένα εξίσου αποδοτικό λογισμικό βελτιστοποίησης προβλημάτων ακεραίου και γραμμικού προγραμματισμού. Η ικανότητα εντοπισμού ακριβούς λύσης μέσα σε σύντομο χρονικό διάστημα ακόμη και σε περιπτώσεις πολύπλοκων προβλημάτων, το καθιστά ιδανικό εργαλείο για τον κλάδο της διαχείρισης εφοδιαστικής αλυσίδας και για την αντιμετώπιση προβλημάτων δρομολόγησης οχημάτων.

- **OR-Tools**: Το OR-Tools αποτελεί μια βιβλιοθήκη ανοιχτού κώδικα που αναπτύχθηκε από την Google με σκοπό την συγκέντρωση και την παροχή χρήσιμων εργαλείων για την επίλυση διαφόρων προβλημάτων βελτιστοποίησης, συμπεριλαμβανομένου και του VRP. Σε αντίθεση με τα προαναφερθέντα λογισμικά, το OR-Tools επικεντρώνεται κυρίως στην εφαρμογή ευρετικών και μεταευρετηκών μεθόδων, οι οποίες θα μελετηθούν αναλυτικότερα σε μεταγενέστερο κεφάλαιο της διατριβής. Ένα βασικό πλεονέκτημά του είναι η εύκολη ενσωμάτωση του σε εφαρμογές Python καθώς και η δωρεάν χρήση του, το οποίο το καθιστά ιδιαίτερα δημοφιλές σε ακαδημαϊκές και ερευνητικές εφαρμογές. Υστερεί, ωστόσο, σε σύγκριση με τα παραπάνω εργαλεία, στην ταχύτητα απόκρισής του σε ορισμένα προβλήματα μεγάλου μεγέθους και στην ικανότητά του να εξασφαλίσει ακριβείς λύσεις σε όλες τις περιπτώσεις.

Xpress-MP: Το συγκεκριμένο εργαλείο αναπτύχθηκε από την FICO για την βελτιστοποίηση ακεραίων και γραμμικών προβλημάτων και κρίνεται ιδανικό για την επίλυση VRP μεγάλης κλίμακας καθώς αποδίδει ακριβή αποτελέσματα ενώ παρέχει και την δυνατότητα παραλληλισμού. Τα κυριότερα μειονεκτήματά του είναι το υψηλό κόστος του και η περιορισμένη διαθεσιμότητα για ακαδημαϊκές άδειες σε σχέση με τα υπόλοιπα εμπορικά πακέτα που απαντώνται στην αγορά.

2. Επίλυση CVRP.

2.1 Μορφοποίηση CVRP.

Σε αυτή την ενότητα παρουσιάζεται η μορφοποίηση Ροής Οχημάτων με Δυο Δείκτες (Two-index Vehicle Flow Formulation). Πρόκειται για μια από τις κλασσικές μορφοποιήσεις του CVRP. Ορίζεται ένα σύνολο πελατών **C = {1,…,n}**, έτσι ώστε μια θετική ζήτηση σχετίζεται με κάθε πελάτη **i** ∈ **C**. Ο αριθμός των οχημάτων που θα εξυπηρετήσουν τους πελάτες είναι **K**. Κάθε διαδρομή θα πρέπει να έχει ως αφετηρία το κέντρο εξυπηρέτησης, να επισκέπτεται ένα υποσύνολο πελατών και τέλος να επιστρέφει στο κέντρο εξυπηρέτησης. Όλοι οι πελάτες θα πρέπει να επισκέπτονται ακριβώς μια φορά. Κάθε όχημα έχει μια προκαθορισμένη χωρητικότητα **Q**, που περιορίζει τον αριθμό των πελατών που μπορεί να επισκεφθεί πριν επιστρέψει στο κέντρο εξυπηρέτησης. Θεωρείται ότι όλα τα οχήματα έχουν ίση χωρητικότητα. Το πρόβλημα αντιπροσωπεύεται από ένα δίκτυο, όπου **N = C** ∪ **{0, n+1}** είναι το σύνολο των κόμβων που συνδέονται με πελάτες στο **C**, και οι κόμβοι 0 και n+1 συνδέονται με το κέντρο εξυπηρέτησης. Ορίζονται δυο κόμβοι για το ίδιο κέντρο εξυπηρέτησης και επιβάλλεται όλες οι διαδρομές να αναχωρούν από τον κόμβο 0 και να τελειώνουν στον n+1. Το σύνολο **E** αποτελείται από τις ακμές (i, j) για κάθε ζευγάρι κόμβων **i, j** ∈ **N**. Το κόστος σχετίζεται για την διάσχιση κάθε ακμής (i, j) ∈ **E** συμβολίζεται με το $\boldsymbol{c_{ij}}$. Κάθε κόμβος έχει ζήτηση $\boldsymbol{q_i}$ > **0** για κάθε i ∈ **C** και $\boldsymbol{q_0 = q_{n+1} = 0}$**.** Ορίζεται η δυαδική μεταβλητή απόφασης $\boldsymbol{x_{ij}}$, η οποία παίρνει την τιμή 1 αν και μόνο αν υπάρχει μια διαδρομή που πηγαίνει απευθείας από τον πελάτη i στον πελάτη j, για **i, j ∈ N**. Επιπλέον, $\boldsymbol{y_j}$ είναι μια συνεχής μεταβλητή απόφασης που αντιστοιχεί στη συσσωρευμένη ζήτηση στη διαδρομή που επισκέπτεται τον κόμβο **j ∈ N** μέχρι αυτή την επίσκεψη. Με αυτές τις παραμέτρους και τις μεταβλητές απόφασης, η μορφοποίηση ροής οχημάτων με δύο δείκτες του CVRP δίνεται ως εξής:

$$Min \ \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_{ij}, \qquad (2,1)$$

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

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} x_{ih} - \sum_{\substack{j=1 \\ j\neq h}}^{n+1} x_{hj} = 0, \quad h = 1, \dots, n, \qquad (2,3)$$

$$\sum_{j=1}^{n} x_{0j} \leq K, \qquad (2,4)$$

$$y_j \geq y_i + q_j x_{ij} - Q(1 - x_{ij}), \quad i, j = 0, \dots, n+1, \qquad (2,5)$$

$$d_i \leq y_i \leq Q, \quad i = 0, \dots, n+1, \qquad (2,6)$$

$$x_{ij} \in \{0,1\}, \quad i, j = 0, \dots, n+1, \qquad (2,7)$$

Οι περιορισμοί (2.2) διασφαλίζουν ότι όλοι οι πελάτες επισκέπτονται ακριβώς μία φορά. Οι περιορισμοί (2.3) εγγυώνται τη σωστή ροή των οχημάτων μέσω των ακμών, δηλώνοντας ότι αν ένα όχημα φτάσει σε έναν κόμβο $h \in N$, τότε πρέπει να αναχωρήσει από αυτόν τον κόμβο. Ο περιορισμός (2.4) περιορίζει τον μέγιστο αριθμό διαδρομών σε *K*, τον αριθμό των οχημάτων. Οι περιορισμοί (2.5) και (2.6) διασφαλίζουν από κοινού ότι δεν ξεπερνιέται η χωρητικότητα του οχήματος. Η αντικειμενική συνάρτηση, που ορίζεται από τον (2.1), επιβάλλει ότι το συνολικό κόστος ταξιδιού των διαδρομών ελαχιστοποιείται. Οι περιορισμοί (2.5) επίσης αποτρέπουν την εμφάνιση υποπεριηγήσεων στη λύση, δηλαδή κυκλικές διαδρομές που δεν περνούν από το σημείο αναχώρησης (κέντρο εξυπηρέτησης).

Η εύρεση της ακριβής λύσης του σε αποδεκτό χρόνο έχει αποδειχθεί δύσκολη. Παρόλο που το ότι μπορεί να γίνει πιο εύκολη με τεχνάσματα όπως γραμμική χαλάρωση ή πιο εξελιγμένες μορφοποιήσεις, με έναν επαρκή αριθμό κόμβων το πρόβλημα θεωρείτε δισεπίλυτο. Αυτό σημαίνει ότι οι μέθοδοι επίλυσης μετά από ένα επαρκή αριθμό κόμβων δεν είναι ικανές να δώσουν βέλτιστη λύση ή ο χρόνος που απαιτούν για τον υπολογισμό της δεν είναι αποδεκτός. Για τον λόγο αυτό εφαρμόζονται ευρετικές και μεταευρετικές μέθοδοι.

2.2 Γιατί το CVRP είναι υπολογιστικά δυσεπίλυτο.

Στην επιστήμη των υπολογιστών τα προβλήματα μπορούν να διακριθούν σε ‘εύκολα’ ή ‘δύσκολα’ με βάση τον υπολογιστικό χρόνο που απαιτείται για την επίλυση τους. Ο χαρακτηρισμός προβλημάτων σε ‘εύκολα’ και ‘δύσκολα’, είναι θεμελιώδης για την

επιστήμη των υπολογιστών, επειδή βοηθά στην κατανόηση της δυσκολίας επίλυσης και τη φύση διαφορετικών προβλημάτων. Ο διαχωρισμός γίνεται ως εξής, τα 'εύκολα' προβλήματα μπορούν να λυθούν σε χρόνο μικρότερο ή ίσο με το πολυώνυμο $O(n^k)$ όπου $k \in \mathbb{R}$ και n ο αριθμός των μεταβλητών του προβλήματος, αυτό ονομάζεται πολυωνυμικός χρόνος. Τα προβλήματα που εξετάζονται στη διατριβή κατατάσσονται στα δύσκολα, δηλαδή ο χρόνος επίλυσης τους είναι μεγαλύτερος από τον πολυωνυμικό. Στη πραγματικότητα αυτός ο διαχωρισμός είναι πολύ απλοϊκός, στην συνέχεια θα εξηγηθούν οι κλάσης προβλημάτων που έχουν ενδιαφέρων για αυτή τη διατριβή σε μεγαλύτερο βάθος. Υπενθυμίζεται ότι το πρόβλημα CVRP είναι μια παραλλαγή του VRP που ενσωματώνει τον περιορισμό της χωρητικότητας των οχημάτων, αυξάνοντας έτσι την πολυπλοκότητα του ήδη δύσκολου VRP. Το VRP με τη σειρά του είναι γενίκευση του TSP που υπάγεται στην κλάση των NP-complete (Non-deterministic Polynomial-time complete), δυσεπίλυτο πρόβλημα. Τα VRP και CVRP ταξινομούνται ως NP-hard (Non-deterministic Polynomial-time hard), υποδεικνύοντας ότι είναι υπολογιστικά δυσεπίλυτα. Η διαφορά μεταξύ NP-complete και NP-hard αναλύονται σε επόμενες παραγράφους. Ο λόγος για την ταξινόμηση αυτή είναι ότι δεν υπάρχει κανένας γνωστός αλγόριθμος που να μπορεί να επιλύσει μεγάλες περιπτώσεις του CVRP σε πολυωνυμικό χρόνο και ο χρόνος που απαιτείται για την επίλυση του προβλήματος αυξάνεται εκθετικά με το μέγεθος της εισόδου.

Τα προβλήματα NP-hard και NP-complete αποτελούν τις βασικές κατηγορίες στην επιστήμη των υπολογιστών και τη θεωρία υπολογιστικής πολυπλοκότητας, καθώς προσδιορίζουν τη δυσκολία επίλυσης και επαλήθευσης ενός προβλήματος. όμως για τη σωστή κατανόηση τους πρέπει να γίνει αναφορά και στις κλάσης P (Polynomial-time) και NP (Non-deterministic Polynomial-time).

Η κλάση P περιλαμβάνει τα προβλήματα που μπορούν να επιλυθούν από έναν υπολογιστή σε πολυωνυμικό χρόνο, δηλαδή σε χρόνο που αυξάνεται με ένα πολυώνυμο ως προς το μέγεθος της εισόδου. Ένα πρόβλημα λέγεται ότι ανήκει στην κλάση P όταν υπάρχει ένας αλγόριθμος που μπορεί να το επιλύσει σε χρόνο $O(n^k)$, όπου n είναι το μέγεθος της εισόδου και k είναι ένας θετικός ακέραιος. Τα P προβλήματα είναι σημαντικά διότι θεωρούνται "εύκολα" για έναν υπολογιστή, ακόμα και αν ορισμένοι αλγόριθμοι επίλυσής τους μπορεί να είναι περίπλοκοι.

Η κλάση NP περιλαμβάνει τα προβλήματα για τα οποία η ορθότητα μιας λύσης μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο από έναν υπολογιστή. Αυτό σημαίνει ότι, αν μας δοθεί μια πιθανή λύση, μπορούμε να ελέγξουμε εάν αυτή είναι σωστή σε εύλογο χρόνο. Ωστόσο, η εύρεση της λύσης μπορεί να είναι πολύ πιο δύσκολη και χρονοβόρα από την επαλήθευση.

Αν και τα NP προβλήματα μπορούν να επαληθευτούν εύκολα, δεν υπάρχει απόδειξη ότι μπορούν να επιλυθούν εξίσου εύκολα. Δηλαδή, παραμένει ανοιχτό το ερώτημα αν τα NP προβλήματα μπορούν να επιλυθούν σε πολυωνυμικό χρόνο, κάτι που εξετάζει με το περίφημο πρόβλημα "P vs NP". Εάν αποδειχθεί ότι τα P είναι ίσα με τα NP, τότε κάθε πρόβλημα που μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο θα μπορούσε και να επιλυθεί σε πολυωνυμικό χρόνο.

Ένα πρόβλημα ανήκει στην κατηγορία NP-hard εάν είναι τουλάχιστον τόσο δύσκολο όσο οποιοδήποτε πρόβλημα στην κλάση NP, ανεξάρτητα αν ανήκει ή όχι στην κλάση NP. Τα NP-hard προβλήματα δεν απαιτείται να έχουν λύση που μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο, αλλά πρέπει να είναι τουλάχιστον εξίσου περίπλοκα με τα πιο δύσκολα NP προβλήματα.

Αν μπορούσαμε να βρούμε έναν αλγόριθμο για την επίλυση ενός NP-hard προβλήματος σε πολυωνυμικό χρόνο, τότε θα μπορούσαμε να επιλύσουμε και κάθε πρόβλημα στην NP με τον ίδιο αλγόριθμο. Παρ’ όλα αυτά, μέχρι σήμερα, δεν έχει βρεθεί τέτοιος αλγόριθμος, και τα NP-hard προβλήματα θεωρούνται, ως εκ τούτου, εξαιρετικά υπολογιστικά απαιτητικά.

Η κλάση NP-complete περιλαμβάνει τα προβλήματα που είναι και στην NP και ταυτόχρονα NP-hard. Αυτά τα προβλήματα είναι τα πιο δύσκολα εντός της κλάσης NP, καθώς μπορούν να επιβεβαιωθούν σε πολυωνυμικό χρόνο αλλά είναι εξίσου δύσκολα με κάθε άλλο NP πρόβλημα.

Τα NP-complete προβλήματα έχουν την ιδιότητα ότι, αν κάποιος βρει έναν πολυωνυμικό αλγόριθμο για την επίλυση ενός NP-complete προβλήματος, τότε θα μπορούσε να επιλύσει όλα τα προβλήματα στην NP σε πολυωνυμικό χρόνο, δίνοντας απάντηση στο πρόβλημα P vs NP.

Η κατηγοριοποίηση αυτή έχει μεγάλη σημασία στην πληροφορική, καθώς τα P προβλήματα μπορούν να επιλυθούν εύκολα και αποδοτικά, ενώ τα NP, NP-hard και NP-complete προβλήματα συνήθως απαιτούν πιο πολύπλοκες μεθόδους ή προσεγγιστικές λύσεις. Τα NP-hard και NP-complete προβλήματα παρουσιάζουν μεγάλες δυσκολίες για την πλήρη επίλυση, ιδιαίτερα καθώς το μέγεθος των δεδομένων αυξάνεται. Ωστόσο, η έρευνα έχει οδηγήσει σε ανάπτυξη προσεγγιστικών και ευρετικών μεθόδων που παρέχουν λύσεις αρκετά καλές για πρακτικούς σκοπούς, χωρίς να είναι πάντα οι βέλτιστες.

Το πρόβλημα P vs NP παραμένει ένα από τα πιο θεμελιώδη και άλυτα προβλήματα της επιστήμης των υπολογιστών και των μαθηματικών. Αναφέρεται στο ερώτημα αν τα προβλήματα που μπορούν να επαληθευτούν σε πολυωνυμικό χρόνο μπορούν επίσης να επιλυθούν σε πολυωνυμικό χρόνο. Μέχρι στιγμής, οι περισσότεροι επιστήμονες υποθέτουν ότι P δεν είναι ίσο με NP, δηλαδή ότι υπάρχουν προβλήματα που μπορούν να επαληθευτούν σε πολυωνυμικό χρόνο αλλά δεν μπορούν να επιλυθούν σε πολυωνυμικό χρόνο.

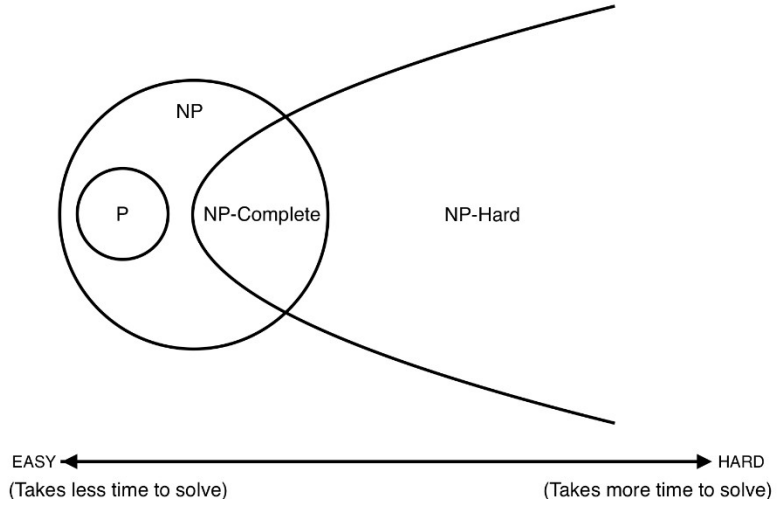
Αν αποδειχθεί ότι P = NP, θα φέρει επανάσταση στην επιστήμη των υπολογιστών, καθώς όλοι οι αλγόριθμοι για τα NP προβλήματα θα μπορούσαν να επιλυθούν σε αποδεκτό χρόνο. Εναλλακτικά, αν αποδειχθεί ότι P ≠ NP, τότε θα επιβεβαιωθεί η ύπαρξη προβλημάτων που δεν μπορούν να επιλυθούν αποδοτικά, παρόλο που η λύση τους μπορεί να επαληθευτεί εύκολα. Για τα προβλήματα που αφορούν την συγκεκριμένη διατριβή αυτό σημαίνει ότι η πόρτα παραμένει ανοιχτή στο να βρεθούν αλγόριθμοι για ακριβείς λύσεις.

Το TSP είναι NP-complete γιατί μπορεί να διατυπωθεί ως πρόβλημα απόφασης: «Υπάρχει μια διαδρομή που να περνά από κάθε πόλη ακριβώς μία φορά και να έχει συνολικό κόστος μικρότερο από ένα δεδομένο όριο;» Η επίλυση του TSP μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο αν δοθεί μια διαδρομή, και είναι επίσης τόσο δύσκολο όσο οποιοδήποτε άλλο NP πρόβλημα, καθιστώντας το NP-complete. Η απόδειξη για την παραπάνω έκφραση ξεφεύγει από τα πλαίσια της διατριβής.

Από την άλλη, τα VRP και CVRP είναι NP-hard. Αν και είναι εξίσου ή πιο περίπλοκα από το TSP, πρόκειται κυρίως για προβλήματα βελτιστοποίησης παρά απόφασης,

όπου ο στόχος είναι η ελαχιστοποίηση της συνολικής απόστασης ή κόστους για πολλαπλά οχήματα με περιορισμούς χωρητικότητας. Επομένως, ενώ δεν απαιτούν μόνο επαλήθευση σε πολυωνυμικό χρόνο, η λύση τους είναι πιο δύσκολη, και δεν εμπίπτουν ακριβώς στην κατηγορία NP-complete.

Αυτή η διάκριση είναι σημαντική, καθώς δείχνει ότι τα TSP, VRP και CVRP απαιτούν πολύπλοκες ευρετικές και μεταευρετικές μεθόδους για αποδοτικές λύσεις.



Εικόνα 2.2 Κλάσης προβλημάτων στην επιστήμη των υπολογιστών.

2.3 Επίλυση CVRP και TSP με ευρετικούς και μεταευρετικούς αλγορίθμους.

Το πρόβλημα του πλανόδιου πωλητή (TSP) και το πρόβλημα δρομολόγησης οχημάτων με χωρητικότητα (CVRP) είναι προβλήματα υπολογιστικά δυσεπίλυτα, πράγμα που σημαίνει ότι η εκθετική αύξηση του αριθμού των πιθανών λύσεων καθιστά ανέφικτη την εύρεση ακριβούς λύσης καθώς αυξάνεται το μέγεθος του προβλήματος. Κατά συνέπεια, χρησιμοποιούνται συχνά ευρετικοί και μεταευρετικοί αλγόριθμοι για τον εντοπισμό λύσεων που είναι σχεδόν βέλτιστες σε εύλογο χρονικό διάστημα. Αυτές οι μέθοδοι εφαρμόζονται συχνά στην πράξη λόγω της ικανότητάς τους να επιτυγχάνουν μια ικανοποιητική ισορροπία μεταξύ της υπολογιστικής αποδοτικότητας και της ποιότητας της λύσης.

Τι είναι οι ευρετικοί και μεταευρετικοί αλγόριθμοι;

Οι ευρετικές μέθοδοι χρησιμοποιούνται σε δύσκολα συνδυαστικά προβλήματα όπου δεν εύκολο να βρεθεί η ακριβής λύση τους. Οι ευρετικές μέθοδοι είναι τεχνικές αναζήτησης βέλτιστων λύσεων που δίνουν «καλές» προσεγγίστηκες λύσεις. Το πλεονέκτημα των ευρετικών λύσεων είναι ότι δίνουν αποτέλεσμα γρηγορά ακόμα και σε περιπτώσεις που δεν είναι δυνατών να βρεθεί ακριβής λύση. Το μειονέκτημα είναι ότι δεν παρέχουν καμία πληροφορία σχετικά με τη ποιότητα της λύσης. Η διαδικασία αναζήτησης ξεκινά από μια τυχαία εφικτή λύση και ακολουθεί επαναληπτική διαδικασία, με στόχο να βρεθεί η επόμενη λύση που θα δώσει καλύτερη τιμή. Σε κάθε επανάληψη εξετάζεται μια περιοχή γύρω από την εκάστοτε λύση και αναζητάτε η επόμενη καλύτερη λύση. Η αναζήτηση τερματίζεται όταν δεν είναι δυνατό να αναδειχθούν καλύτερες λύσεις. Δηλαδή αν δεν μπορεί να βρεθεί καλύτερη τιμή μέσα στην περιοχή που ελέγχεται γύρο από την εκάστοτε λύση, δεν μπορεί να οριστεί καινούριο σημείο και η αναζήτηση σταματάει. Αυτό κάνει την μέθοδο επιρρεπής στο να παγιδεύεται σε τοπικά βέλτιστα. Οι μεταευρετικές μέθοδοι αναπτύχθηκαν για να αποφεύγεται η παγίδευση σε τοπικά βέλτιστα, καθώς επιτρέπουν κινήσεις που θεωρούνται υποδεέστερες από μια απλή ευρετική μέθοδο. Η ιδέα είναι ότι με την προστιθέμενη ευελιξία η μέθοδος θα μπορεί να ξεφεύγει από τοπικά βέλτιστα και κατά συνέπεια θα δίνει καλύτερες λύσεις. Σε αντίθεση με τις ευρετικές μεθόδους δεν υπάρχει διακριτή συνθήκη τερματισμού αναζήτησης και θα πρέπει να ορίζεται ξεχωριστά. Ορισμένες συνθήκες είναι το πλήθος των επαναλήψεων να μην υπερβαίνει ένα συγκεκριμένο αριθμό, η ποιότητα της λύσης να είναι αποδεκτή και ο χρόνος της αναζήτησης να μην υπερβαίνει ένα συγκεκριμένο όριο.

Πλεονεκτήματα ευρετικών και μεταευρετηκών αλγόριθμων;

1. Επεκτασιμότητα: Οι ακριβείς μέθοδοι, όπως ο γραμμικός προγραμματισμός ή η μέθοδος branch-and-bound, καθίστανται υπολογιστικά απαγορευτικές για μεγάλες περιπτώσεις CVRP και TSP καθώς αυξάνεται το μέγεθος του προβλήματος. Τα προβλήματα του πραγματικού κόσμου με εκατοντάδες ή χιλιάδες κόμβους είναι κατάλληλα για ευρετικούς και μεταευρετικούς αλγορίθμους, οι οποίοι παρέχουν κλιμακούμενες λύσεις.
2. Χρονική αποδοτικότητα: Στον τομέα της εφοδιαστικής και της αποστολής οχημάτων, είναι συχνά απαραίτητο να λαμβάνονται αποφάσεις σε

πραγματικό χρόνο ή εντός αυστηρών χρονικών περιορισμών. Οι μεταευρετικοί αλγόριθμοι θυσιάζουν συχνά κάποια ακρίβεια για την ταχύτητα υπολογισμού, προκειμένου να παρέχουν λύσεις σε εύλογο χρονικό διάστημα.

3. Ευελιξία: Οι ευρετικοί και οι μεταευρετικοί αλγόριθμοι είναι προσαρμόσιμοι ώστε να μπορούν να προσαρμόζονται σε διάφορους περιορισμούς του προβλήματος, όπως η χωρητικότητα των οχημάτων, τα χρονικά παράθυρα και τα σενάρια πολλαπλών αποθηκών, γεγονός που τους καθιστά ικανούς να διαχειρίζονται πιο περίπλοκες παραλλαγές των CVRP και TSP.

4. Σχεδόν βέλτιστες λύσεις: Αν και οι ευρετικές μέθοδοι δεν εγγυώνται συνήθως βέλτιστες λύσεις, συχνά παράγουν λύσεις που είναι σχεδόν βέλτιστες, ιδίως όταν βοηθιούνται από ευφυείς μηχανισμούς αναζήτησης όπως αυτοί που συναντώνται στις μεταευρετικές μέθοδοι.

2.3.1 Ευρετικοί αλγόριθμοι για CVRP και TSP.

1. Αλγόριθμος του πλησιέστερου γείτονα (NNA): Αυτός είναι ένας από τους βασικότερους ευρετικούς αλγόριθμους για το TSP. Ο επόμενος προορισμός είναι η πλησιέστερη μη επισκέψιμη πόλη, ξεκινώντας με μια αυθαίρετη πόλη και συνεχίζοντας μέχρι να επισκεφθούμε όλες τις πόλεις. Στη συνέχεια, η περιήγηση επιστρέφει στην αρχική θέση. Αν και η NNA είναι μια γρήγορη μέθοδος υπολογισμού, δεν παράγει πάντα λύσεις υψηλής ποιότητας και μπορεί να οδηγήσει σε μη βέλτιστες περιηγήσεις, ιδίως όταν χρησιμοποιείται σε μεγαλύτερες περιπτώσεις.
2. Αλγόριθμος εξοικονόμησης: Αυτός ο ευρτικός αλγόριθμος χρησιμοποιείται συχνά στο CVRP και ξεκινά με κάθε καταναλωτή να εξυπηρετείται από ένα ξεχωριστό όχημα. Στη συνέχεια, ο αλγόριθμος συγχωνεύει τις διαδρομές με βάση την εξοικονόμηση που επιτυγχάνεται από τον συνδυασμό των διαδρομών, με στόχο την ελαχιστοποίηση της συνολικής διανυόμενης απόστασης, τηρώντας παράλληλα τους περιορισμούς χωρητικότητας των οχημάτων. Αυτή η μεθοδολογία εφαρμόζεται συχνά μέσω του αλγορίθμου Clarke and Wright Savings Algorithm.

3. Ευρετικές μέθοδοι εισαγωγής: Αυτή η ευρετική περιλαμβάνει την αρχική επιλογή ενός υποσυνόλου κόμβων για τον σχηματισμό μιας μερικής περιήγησης και την επακόλουθη εισαγωγή των υπόλοιπων κόμβων στην περιήγηση με βάση συγκεκριμένα κριτήρια, όπως η ελάχιστη αύξηση του κόστους. Αυτή η προσέγγιση είναι πλεονεκτική τόσο για το TSP όσο και για το CVRP, καθώς διευκολύνει μια απλή αλλά αποτελεσματική διαδικασία δημιουργίας λύσεων .

2.3.2 Μεταευρετικοί αλγόριθμοι: TABU SEARCH, GUIDED LOCAL SEARCH

Οι μεταευρετικοί αλγόριθμοι έχουν καταστεί απαραίτητοι για την επίλυση σύνθετων προβλημάτων βελτιστοποίησης, όπου οι κλασικοί αλγόριθμοι συχνά δυσκολεύονται. Ξεχωρίζουν για την εξερεύνηση μεγάλων χώρων αναζήτησης και τον εντοπισμό σχεδόν βέλτιστων λύσεων εντός ενός αποδεκτού χρονικού πλαισίου. Αυτές οι τεχνικές είναι ιδιαίτερα πολύτιμες για την αντιμετώπιση NP-hard προβλημάτων, όπως το TSP και το CVRP. Οι μεταευρετικές μέθοδοι προσφέρουν μηχανισμούς διαφυγής από τα τοπικά βέλτιστα και διερεύνησης του παγκόσμιου χώρου λύσεων σε βάθος. Η παρούσα υποενότητα θα επικεντρωθεί σε δυο ευρέως χρησιμοποιούμενες μεταευρετικές μεθόδους, περιγράφοντας λεπτομερώς τον τρόπο λειτουργίας τους και τις ξεχωριστές στρατηγικές τους.

2.3.2.1 Tabu Search (TS)

Η αναζήτηση Tabu Search, που εισήχθη από τον Fred Glover το 1986, είναι μια μέθοδος βελτιστοποίησης με βάση τη γειτονιά που ενισχύει τη βασική τοπική αναζήτηση με δομές μνήμης. Αυτές οι συνιστώσες μνήμης εμποδίζουν τον αλγόριθμο να επανεξετάσει λύσεις που είχαν εξερευνηθεί προηγουμένως και αποφεύγουν τον εγκλωβισμό σε τοπικά βέλτιστα. Η TS είναι αποτελεσματική για συνδυαστικά προβλήματα, όπως το TSP και το CVRP, λόγω της συστηματικής προσέγγισής της στην αναζήτηση του χώρου λύσεων.

Βασικά στοιχεία :

- Εξερεύνηση γειτονιάς: Το TS αξιολογεί ένα σύνολο γειτονικών λύσεων εφαρμόζοντας μικρές τροποποιήσεις στην τρέχουσα λύση (π.χ., ανταλλαγή δύο πόλεων στο TSP ή ανακατανομή πελατών στο CVRP). Ο αλγόριθμος επιλέγει έναν γείτονα από αυτό το σύνολο για να εξερευνήσει στο επόμενο βήμα.

- Λίστα Tabu: Μια βραχυπρόθεσμη μνήμη, η λίστα tabu καταγράφει λύσεις που επισκέφθηκαν πρόσφατα ή συγκεκριμένες κινήσεις (όπως ανταλλαγές κόμβων) που απαγορεύονται σε μελλοντικές επαναλήψεις. Με τον τρόπο αυτό αποφεύγεται η επανάληψη κακών λύσεων και η λίστα ενημερώνεται δυναμικά κατά τη διάρκεια της διαδικασίας αναζήτησης.

- Κριτήρια αλλαγής: Μερικές φορές, μια κίνηση που θεωρείται «tabu» μπορεί στην πραγματικότητα να οδηγήσει σε μια καλύτερη λύση από την τρέχουσα καλύτερη. Σε τέτοιες περιπτώσεις, το κριτήριο φιλοδοξίας επιτρέπει την αποδοχή λύσεων tabu, εάν οδηγούν σε βελτίωση.

- Εντατικοποίηση και διαφοροποίηση: Το TS εναλλάσσεται μεταξύ της εντατικοποίησης της αναζήτησης σε πολλά υποσχόμενες περιοχές και της διαφοροποίησής της για την εξερεύνηση νέων περιοχών. Η εντατικοποίηση επικεντρώνεται στη βελτίωση των καλών λύσεων, ενώ η διαφοροποίηση ωθεί την αναζήτηση σε ανεξερεύνητα τμήματα του χώρου λύσεων.

Διαδικασία :

1. Αρχικοποίηση: Ο αλγόριθμος ξεκινά με μια εφικτή αρχική λύση.

2. Παραγωγή γειτόνων: Πραγματοποιούνται μικρές αλλαγές στην τρέχουσα λύση για τη δημιουργία ενός συνόλου γειτονικών λύσεων.

3. Περιορισμοί Tabu: Οι κινήσεις που θα οδηγούσαν σε λύσεις στη λίστα tabu αποφεύγονται, εκτός εάν ικανοποιείται το κριτήριο προσδοκίας.

4. Επιλογή καλύτερου γείτονα: Ο καλύτερος επιτρεπόμενος γείτονας (που δεν βρίσκεται στη λίστα tabu ή δεν επιτρέπεται από την επιδίωξη) επιλέγεται ως η νέα τρέχουσα λύση.

5. Ανανέωση της λίστας Tabu: Η πιο πρόσφατη κίνηση προστίθεται στη λίστα tabu για να αποτραπεί η επανάληψή της στο εγγύς μέλλον.

6. Επανάληψη: Ο αλγόριθμος επαναλαμβάνει αυτή τη διαδικασία έως ότου επιτευχθεί ένα προκαθορισμένο κριτήριο διακοπής, όπως ένα χρονικό όριο ή ένας μέγιστος αριθμός επαναλήψεων.

Η TS είναι ιδιαίτερα κατάλληλη για μεγάλα, πολύπλοκα προβλήματα, επειδή εξισορροπεί την τοπική και τη σφαιρική αναζήτηση, εξερευνώντας αποτελεσματικά νέες περιοχές του χώρου λύσεων και αποτρέποντας παράλληλα την ανακύκλωση σε φτωχές λύσεις.

2.3.2.2 GUIDED LOCAL SEARCH (GLS)

Η καθοδηγούμενη τοπική αναζήτηση βελτιώνει τους παραδοσιακούς αλγορίθμους τοπικής αναζήτησης, τιμωρώντας τα χαρακτηριστικά της λύσης που συμβάλλουν σε τοπικά βέλτιστα. Αυτός ο μηχανισμός ποινών τροποποιεί την αντικειμενική συνάρτηση, καθοδηγώντας τον αλγόριθμο να αποφεύγει διαμορφώσεις που έχουν οδηγήσει σε κακές λύσεις στο παρελθόν.

Βασικά στοιχεία :

- Τοπική αναζήτηση: Η GLS λειτουργεί πάνω σε έναν αλγόριθμο τοπικής αναζήτησης, ο οποίος βελτιώνει επαναληπτικά μια λύση κάνοντας μικρές αλλαγές.

- Σύστημα ποινών: Ορισμένα χαρακτηριστικά της λύσης (π.χ. συχνά χρησιμοποιούμενες ακμές στο TSP) τιμωρούνται με βάση το πόσο συχνά εμφανίζονται σε μη βέλτιστες λύσεις. Η ποινή ενθαρρύνει την αναζήτηση να αποφεύγει αυτά τα χαρακτηριστικά σε μελλοντικές επαναλήψεις.

- Τροποποιημένη αντικειμενική συνάρτηση: Η αρχική αντικειμενική συνάρτηση ενισχύεται με όρους ποινής, καθιστώντας λιγότερο πιθανό για τον αλγόριθμο να επανεξετάσει λύσεις που περιέχουν τα τιμωρημένα χαρακτηριστικά.

Διαδικασία :

1. Αρχική λύση: Ξεκινά με μια εφικτή λύση που λαμβάνεται μέσω ενός αλγορίθμου τοπικής αναζήτησης.

2. Προσδιορισμός χαρακτηριστικών: Προσδιορίζονται συγκεκριμένα χαρακτηριστικά της λύσης που οδηγούν σε μη βέλτιστη απόδοση (π.χ. συχνά χρησιμοποιούμενες διαδρομές στο CVRP).

3. Επικαιροποίηση των κυρώσεων: Εφαρμόζονται ποινές σε αυτά τα χαρακτηριστικά και τροποποιούνται ανάλογα την αντικειμενική συνάρτηση.

4. Επανεκκίνηση της τοπικής αναζήτησης: Χρησιμοποιώντας την αντικειμενική συνάρτηση με τις ποινές, η τοπική αναζήτηση επανεκκινείται.

5. Επανάληψη: Η διαδικασία συνεχίζεται με επικαιροποιημένες ποινές έως ότου η αναζήτηση συγκλίνει σε μια λύση υψηλής ποιότητας ή ικανοποιηθεί κάποιο άλλο κριτήριο διακοπής.

Η GLS είναι ιδιαίτερα αποτελεσματική για πολύπλοκα προβλήματα όπως το TSP και το CVRP, επειδή κατευθύνει την αναζήτηση μακριά από προβληματικές περιοχές, διατηρώντας παράλληλα την αποτελεσματικότητα της τοπικής αναζήτησης. Αποτρέπει την αναζήτηση από το να κολλήσει σε τοπικά ελάχιστα βελτιώνοντας συνεχώς την αντικειμενική συνάρτηση.

2.4 Μορφοποίηση Προβλήματος Ανάθεσης.

Το πρόβλημα ανάθεσης είναι ένα από τα κλασσικά προβλήματα συνδυαστικού προγραμματισμού, όπου έχει στόχο να ανατεθούν εργασίες σε παράγοντες που είναι ικανοί να τις εκτελέσουν, με τρόπο που θα βελτιστοποιεί είτε το κόστος είτε την αποδοτικότητα. Στην περίπτωση του προβλήματος CVRP το πρόβλημα ανάθεσης καλείται να αναθέσει τους πελάτες στα οχήματα με τρόπο που να ελαχιστοποιεί την απόσταση που θα καλυφθεί.

$$Min \quad \sum_{i=1}^{m} \sum_{j=i}^{n} c_{ij} y_{ij}$$

$$s.t. \quad \sum_{j=1}^{m} y_{ij} \leq q_i, \; i = 1 \cdots m$$

$$\sum_{i=1}^{n} y_{ij} = 1, \quad j = 1 \cdots n$$

$$y_{i,j} \geq 0, \qquad \forall \text{ i,j}$$

$$y_{i,j} \in \{0,1\}, \qquad \forall \text{ i,j}$$

Όπου:

- Π_i, i={1,2,…m} τα οχήματα και Δ_j, j={1,2,.., n} οι πελάτες.
- y_{ij}=1 αν το όχημα Π_i χρησιμοποιηθεί στον πελάτη Δ_j.
- y_{ij}=0 αν το όχημα Π_i δεν χρησιμοποιηθεί στον πελάτη Δ_j.
- c_{ij} οι συντελεστές κόστους όταν το όχημα Π_i χρησιμοποιηθεί στον πελάτη Δ_j.
- q_i η χωρητικότητα του οχήματος Π_i.

Η επίλυση του προβλήματος ανάθεσης επιδιώκει την εύρεση της βέλτιστης αντιστοίχισης εργασιών σε πόρους, ώστε να ελαχιστοποιηθεί το συνολικό κόστος. Ένας από τους πιο αποδοτικούς αλγόριθμους για την επίλυσή του είναι ο Ουγγρικός Αλγόριθμος, ο οποίος βασίζεται στην τεχνική των «μηδενικών» του πίνακα κόστους. Ο αλγόριθμος αυτός προσδιορίζει μια βέλτιστη λύση, πραγματοποιώντας επαναληπτικά βήματα για να καλύψει κάθε γραμμή ή στήλη που περιέχει μηδενικά, προσθέτοντας ή αφαιρώντας κατάλληλες τιμές κόστους, μέχρις ότου βρεθεί η βέλτιστη αντιστοίχιση.

Για μεγαλύτερα ή πιο πολύπλοκα προβλήματα ανάθεσης, χρησιμοποιείται επίσης η Γραμμική Προγραμματιστική μέθοδος, η οποία διατυπώνει το πρόβλημα ως ένα σύνολο ανισοτήτων και το λύνει μέσω της μεθόδου Simplex ή άλλων αλγορίθμων βελτιστοποίησης. Αν η εύρεση της απόλυτης βέλτιστης λύσης δεν είναι εφικτή, χρησιμοποιούνται ευρετικές μέθοδοι, όπως ο Αλγόριθμος Γενετικών ή οι Μέθοδοι Προσομοιωμένης Ανόπτησης, που επιτρέπουν την εύρεση καλών λύσεων σε μικρότερο χρόνο.

Οι σύγχρονες λύσεις στο πρόβλημα ανάθεσης συχνά περιλαμβάνουν υβριδικές μεθόδους, συνδυάζοντας αλγόριθμους ακριβείς και προσεγγιστικούς, ώστε να επιτύχουν την καλύτερη ισορροπία μεταξύ χρόνου και ακρίβειας για την εφαρμογή τους σε πραγματικές καταστάσεις.

Στα πλαίσια αυτής της διατριβής τα προβλήματα ανάθεσης είναι σχετικά μικρά. Επομένως η επίλυση τους δεν γίνεται με χρήση προσεγγιστικών μεθόδων.

3. OR TOOLS και Περιγραφή Κώδικα.

3.1 OR TOOLS

3.1.1 Σχετικά με το OR TOOLS

Το OR TOOLS είναι λογισμικό ανοιχτού κώδικα, το οποίο επιδιώκει να βρει την καλύτερη λύση προβλημάτων μαθηματικού προγραμματισμού. Το OR TOOLS δημιουργήθηκε το 2011 από την Google. Η ανάπτυξη του λογισμικού έγινε πάνω στη C++, αλλά μπορεί να «τρέξει» και σε Python, Java ή .Net. Σήμερα το OR TOOLS βρίσκει πληθώρα εφαρμογών όπως, Προγραμματισμός, Logistics, Κατανομή Πόρων και Οικονομικά.

3.1.2 Μέρη του OR TOOLS.

Το φάσμα των προβλημάτων μαθηματικού προγραμματισμού που μπορεί να επιλυθεί με το OR TOOLS είναι μεγάλο. Καθώς κάθε κατηγορία μοντέλου μαθηματικού προγραμματισμού χρήζει και διαφορετική μέθοδο επίλυσης, το OR TOOLS κάνει χρήση διαφορετικών δομοστοιχείων επίλυσης (**solver**) για κάθε κατηγορία. Τα προβλήματα για τα οποία συμπεριλαμβάνονται solver είναι:

- **Προγραμματισμός Περιορισμών (Constraint Programming):** Ο Προγραμματισμός Περιορισμών είναι ένα σύνολο τεχνικών για την εύρεση εφικτών λύσεων σε ένα πρόβλημα εκφρασμένο από περιορισμούς.
- **Γραμμική Βελτιστοποίηση (Linear Optimization):** Η γραμμική βελτιστοποίηση (ή γραμμικός προγραμματισμός) είναι η διαδικασία υπολογισμού της καλύτερης λύσης σε ένα πρόβλημα που μοντελοποιείται ως ένα σύνολο γραμμικών σχέσεων.
- **Δρομολόγηση Οχημάτων (Vehicle Routing):** Η δρομολόγηση οχημάτων είναι μία από τις πιο κοινές εργασίες βελτιστοποίησης, όπου ο στόχος είναι να βρεθούν οι καλύτερες διαδρομές για έναν στόλο οχημάτων που

επισκέπτονται ένα σύνολο τοποθεσιών. Συνήθως, "καλύτερες" διαδρομές σημαίνει αυτές με τη μικρότερη συνολική απόσταση ή κόστος.

- **Αλγόριθμοι Γραφημάτων (Graph Algorithms):** Οι αλγόριθμοι γραφημάτων χρησιμοποιούνται για την επίλυση προβλημάτων που περιλαμβάνουν γραφήματα, δηλαδή σύνολα κόμβων συνδεδεμένων με ακμές.

3.1.3 Επίλυση CVRP και TSP με OR TOOLS.

Όπως αναφέρθηκε και νωρίτερα το OR TOOLS έχει ξεχωριστό solver αφιερωμένο για την επίλυση προβλημάτων Δρομολόγησης Οχημάτων. Σε αυτή τη κατηγορία προβλημάτων ανήκουν το TSP και το CVRP. Ο αλγόριθμος που εκτελεί ο solver χρησιμοποιεί ευρετική μέθοδο για να βρει την αρχική λύση και εν συνεχεία μεταευρετική μέθοδο για να διασφαλιστεί η βελτιστότητα. Επειδή τα προβλήματα δρομολόγησης είναι υπολογιστικά δισεπίλυτα καλείται ευρετική μεθοδολογία όπου αναζητά εφικτές λύσεις του προβλήματος, τις οποίες συγκρίνει μεταξύ τους και επιστρέφει την βέλτιστη. Η αχίλλειος πτέρνα του ευρετικού αλγορίθμου είναι ότι συγκλείνει σε τοπικά ελάχιστα. Αυτό συμβαίνει γιατί τα κριτήρια που λαμβάνονται υπόψιν για την επιλογή της επομένης στάσης του οχήματος είναι οι στάσεις που θεωρητικά ήδη έχει πραγματοποιήσει, και η επόμενη στάση που καλείτε να επιλέξει. Αυτό κάνει τον αλγόριθμο μυωπικό, δηλαδή δεν εξετάζει ολικά το πρόβλημα αλλά επιχειρεί να βρει την επόμενη καλύτερη στάση. Για το λόγο αυτό λύση που επιστρέφεται δεν είναι απαραίτητα βέλτιστη. Ο μεταευρετηκός αλγόριθμος «πατάει» στη λύση του ευρετικού και αναζητεί καλύτερες λύσεις λαμβάνοντας υπόψιν ολόκληρο το σύνολο των εφικτών λύσεων. Το πρόβλημα με αυτή την προσέγγιση είναι ότι με έναν επαρκή αριθμό μεταβλητών ο χρόνος που μεσολαβεί για την ολοκλήρωση της αναζήτησης μπορεί να φτάσει και τις μέρες. Με αυτό υπόψιν δίνεται η επιλογή να ορίσει κάποια συνθήκη τερματισμού της αναζήτησης, συνήθως είναι η χρονική διάρκεια της αναζήτησης. Κατά συνέπεια είναι μείζονος σημασίας ο μεταευρετηκός αλγόριθμος να πλησιάσει την βέλτιστη λύση όσο το δυνατόν γρηγορότερα στην αναζήτηση του.

Εκτός από TSP και CVRP προβλήματα ο solver επιλύει και άλλα προβλήματα δρομολόγησης όπως:

- **VRP:** Στο Πρόβλημα Δρομολόγησης Οχημάτων (Vehicle Routing Problem - VRP), ο στόχος είναι να βρεθούν βέλτιστες διαδρομές για πολλαπλά οχήματα που επισκέπτονται ένα σύνολο τοποθεσιών. Όταν υπάρχει μόνο ένα όχημα, το πρόβλημα εκφυλίζεται σε Πρόβλημα του Περιπλανώμενου Πωλητή (Traveling Salesperson Problem - TSP).
- **VRP με Παραλαβές και Διανομές:** Ένα Πρόβλημα Δρομολόγησης Οχημάτων (VRP) στο οποίο κάθε όχημα παραλαμβάνει αντικείμενα από διάφορες τοποθεσίες και τα παραδίδει σε άλλες. Το πρόβλημα είναι να ανατεθούν διαδρομές στα οχήματα για να παραλάβουν και να παραδώσουν όλα τα αντικείμενα, ενώ ελαχιστοποιείται το μήκος της μεγαλύτερης διαδρομής.
- **VRPTW:** Το Πρόβλημα Δρομολόγησης Οχημάτων με Χρονικά Παράθυρα (Vehicle Routing Problem with Time Windows - VRPTW) είναι μια παραλλαγή του κλασικού προβλήματος δρομολόγησης οχημάτων (VRP). Στο VRPTW, κάθε τοποθεσία έχει συγκεκριμένα χρονικά παράθυρα μέσα στα οποία πρέπει να πραγματοποιηθεί η παραλαβή ή η παράδοση των αντικειμένων.

Ο solver χρησιμοποιεί ένα αντικείμενο που ονομάζεται **διάσταση** (dimension) για να παρακολουθεί τις ποσότητες που συσσωρεύονται κατά μήκος της διαδρομής ενός οχήματος, όπως ο χρόνος ταξιδιού, εάν το όχημα κάνει παραλαβές και παραδόσεις, ή το φορτίο που μεταφέρει εάν υπάρχει όριο χωρητικότητας. Αν ένα πρόβλημα δρομολόγησης περιλαμβάνει μια τέτοια ποσότητα, είτε στους περιορισμούς είτε στη αντικειμενική συνάρτηση, πρέπει να οριστεί μια διάσταση που θα την αντιπροσωπεύει.

Τα όρια αναζήτησης τερματίζουν την διαδικασία αναζήτησης αφού φτάσει σε ένα καθορισμένο όριο, όπως η μέγιστη διάρκεια χρόνου ή ο αριθμός των λύσεων που βρέθηκαν. Τα όρια αναζήτησης καθορίζονται με τις παρακάτω μεθόδους:

- **solution_limit:** Όριο στον αριθμό των λύσεων που παράγονται κατά τη διάρκεια της αναζήτησης.

- **time_limit.seconds:** Όριο σε δευτερόλεπτα για τον χρόνο που δαπανάται στην αναζήτηση.
- **lns_time_limit.seconds:** Όριο σε δευτερόλεπτα για τον χρόνο που δαπανάται για την ολοκλήρωση της αναζήτησης για κάθε τοπικό γείτονα.

Οι ευρετικές μέθοδοι για την εύρεση πρώτης λύσης που προσφέρει το OR TOOLS είναι:

- **AUTOMATIC**: Επιτρέπει στον solver να επιλέξει ποια στρατηγική να χρησιμοποιήσει ανάλογα με το μοντέλο που επιλύεται**.**
- **PATH_CHEAPEST_ARC:** Ξεκινώντας από τον κόμβο "εκκίνησης" της διαδρομής, συνδέεται τον με τον κόμβο που παράγει το φθηνότερο τμήμα της διαδρομής, στη συνέχεια επεκτείνετε τη διαδρομή επαναλαμβάνοντας την προσθήκη του τελευταίου κόμβου στη διαδρομή.
- **PATH_MOST_CONSTRAINED_ARC:** Παρόμοιο με το PATH_CHEAPEST_ARC, αλλά οι ακμές αξιολογούνται με έναν συγκριτικό δείκτη που προτιμά τις πιο περιορισμένες ακμές πρώτα.
- **EVALUATOR_STRATEGY:** Παρόμοιο με το PATH_CHEAPEST_ARC, εκτός από το ότι τα κόστη των ακμών αξιολογούνται χρησιμοποιώντας τη συνάρτηση που περνάτε στη μέθοδο **SetFirstSolutionEvaluator()**.
- **SAVINGS:** Αλγόριθμος Savings (Clarke & Wright). Reference Clarke, G. & Wright, J.W. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points" , Operations Research, Vol. 12, 1964, pp. 568-581.
- **SWEEP:** Αλγόριθμος Sweep (Wren & Holliday). Reference Anthony Wren & Alan Holliday Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points Operational Research Quarterly (1970-1977), Vol. 23, No. 3 (Sep., 1972), pp. 333-344.
- **CHRISTOFIDES**: Ο αλγόριθμος Christofides, λειτουργεί σε γενικά μοντέλα δρομολόγησης οχημάτων επεκτείνοντας μια διαδρομή μέχρι να μην μπορούν να προστεθούν άλλοι κόμβοι σε αυτήν. Αναφορά: Christofides, N. "Worst-case analysis of a new heuristic for the traveling salesman problem." In Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC), pp. 56-61. ACM, 1973.

- **ALL_UNPERFORMED**: Καθιστά όλους τους κόμβους ανενεργούς. Βρίσκει λύση μόνο εάν οι κόμβοι είναι προαιρετικοί (είναι στοιχεία μιας περιοριστικής συνθήκης με περιορισμένο κόστος ποινής).
- **BEST_INSERTION**: Χτίζει επαναληπτικά μια λύση εισάγοντας τον φθηνότερο κόμβο στην πιο φθηνή θέση του. Το κόστος της εισαγωγής βασίζεται στη γενική συνάρτηση κόστους του μοντέλου δρομολόγησης λειτουργεί μόνο σε μοντέλα με προαιρετικούς κόμβους (με περιορισμένα κόστη ποινής).
- **PARALLEL_CHEAPEST_INSERTION**: Χτίζει επαναληπτικά μια λύση εισάγοντας τον φθηνότερο κόμβο στη φθηνότερη θέση του. Το κόστος της εισαγωγής βασίζεται στη συνάρτηση κόστους ακμής. Είναι ταχύτερη από την BEST_INSERTION.
- **LOCAL_CHEAPEST_INSERTION:** Χτίζει επαναληπτικά μια λύση εισάγοντας κάθε κόμβο στη φθηνότερη θέση του. Το κόστος της εισαγωγής βασίζεται στη συνάρτηση κόστους ακμής. Διαφέρει από την PARALLEL_CHEAPEST_INSERTION στον κόμβο που επιλέγεται για εισαγωγή. Εδώ, οι κόμβοι εξετάζονται με τη σειρά της δημιουργίας τους. Είναι ταχύτερη από την PARALLEL_CHEAPEST_INSERTION.
- **GLOBAL_CHEAPEST_ARC:** Συνδέει επαναληπτικά δύο κόμβους που παράγουν το φθηνότερο τμήμα διαδρομής.
- **LOCAL_CHEAPEST_ARC:** Επιλέγει τον πρώτο κόμβο με μη δεσμευμένο επόμενο κόμβο και τον συνδέει με τον κόμβο που παράγει το φθηνότερο τμήμα διαδρομής.
- **FIRST_UNBOUND_MIN_VALUE:** Επιλέγει τον πρώτο κόμβο με μη δεσμευμένο επόμενο κόμβο και τον συνδέει με τον πρώτο διαθέσιμο κόμβο.

Οι μεταευρετικές μέθοδοι για την εύρεση πρώτης λύσης που προσφέρει το OR TOOLS είναι:

- **AUTOMATIC**: Επιτρέπει στον solver να επιλέξει τη μεταευρετική μέθοδο.
- **GREEDY_DESCENT:** Αποδέχεται βελτιωτικούς (μειωτικού κόστους) τοπικούς γείτονες μέχρι να επιτευχθεί ένα τοπικό ελάχιστο.

- **GUIDED_LOCAL_SEARCH:** Χρησιμοποιεί guided local search για να ξεφύγει από τοπικά ελάχιστα. Αυτή είναι γενικά η πιο αποδοτική μεταευρετική μέθοδος για δρομολόγηση οχημάτων.
- **SIMULATED_ANNEALING:** Χρησιμοποιεί simulated annealing για να ξεφύγει από τοπικά ελάχιστα.
- **TABU_SEARCH:** Χρησιμοποιεί tabu search για να ξεφύγει από τοπικά ελάχιστα.
- **GENERIC_TABU_SEARCH:** Χρησιμοποιεί tabu search στην αντικειμενική τιμή της λύσης για να ξεφύγει από τοπικά ελάχιστα.

3.1.4 Επίλυση Προβλήματος Ανάθεσης με OR TOOLS.

Τα προβλήματα ανάθεσης ανήκουν στη κατηγορία του ακέραιου προγραμματισμού. Όπως και για τα προβλήματα δρομολόγησης, το OR TOOLS έχει ξεχωριστούς solver αφιερωμένους για την επίλυση προβλημάτων ακέραιου προγραμματισμού. Συγκεκριμένα οι solver απευθύνονται σε προβλήματα που απαιτούν μερικές από τις μεταβλητές να είναι ακέραιοι αριθμοί, δηλαδή Μεικτά Ακέραια Προγράμματα (Mixed Integer Programs - MIP).

Το OR TOOLS παρέχει διάφορους solver για την επίλυση προβλημάτων MIP:

- **MPSolver**: Ένα συνονθύλευμα πολλούς εξωτερικούς solver MIP, οι οποίοι χρησιμοποιούν τις τυπικές τεχνικές branch-and-bound.
- **CP-SAT solver**: Ένας λύτης προγραμματισμού περιορισμών που χρησιμοποιεί μεθόδους ικανοποίησης περιορισμών (SAT).
- **Original CP solver**: Ένας αρχικός λύτης προγραμματισμού περιορισμών.

Για προβλήματα ανάθεσης προτείνονται οι MPSolver και CP-SAT solver. Για τυπικά MIPs που έχουν τόσο ακέραιες όσο και Boolean μεταβλητές, συχνά δεν υπάρχει σαφής διαφορά στην ταχύτητα μεταξύ των δύο solver, οπότε η επιλογή μπορεί να εξαρτηθεί από την προσωπική σας προτίμηση.

3.2 Περιγραφή Κώδικα.

Για την περιγραφή του αλγορίθμου παρουσιάζεται αρχικώς το εκάστοτε απόσπασμα του κώδικα και στην συνέχεια παρατίθεται επεξηγηματικό κείμενο που αποσαφηνίζει την λειτουργία και την χρησιμότητα του αντίστοιχου αποσπάσματος. Επειδή ο κώδικας είναι χωρισμένος σε σκέλη, κάθε σκέλος εξηγείται σε ξεχωριστή υποενότητα.

3.2.1 Εισαγωγή Βιβλιοθηκών.

```
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
from ortools.linear_solver import pywraplp
import pandas as pd
import numpy as np
import random
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib as mpl
```

Η πρώτη ενέργεια του προγράμματος είναι η εισαγωγή των απαραίτητων βιβλιοθηκών. Οι βιβλιοθήκες είναι οι εξής:

i. OR-TOOLS (επίλυση μαθηματικού προγραμματισμού)
ii. Pandas (επεξεργασία “data frame” δεδομένων)
iii. NumPy (μαθηματικό πακέτο)
iv. Random (παράγει τυχαίους αριθμούς)
v. NetworkX (παράγει γραφήματα δικτύων)
vi. Matplotlib (παράγει γραφήματα)

3.2.2 Δημιουργία Τυχαίου Προβλήματος.

```
def data_generation(loc_lb, loc_ub, nv_lb, nv_ub, dem_lb, dem_ub, capacities_equal, rounding,
zero_slack, slack, output) :
    # random problem data.
    data = {}
    # number of locations.
    number_loc = random.randint(loc_lb, loc_ub)
    # number of vehicles.
    data["num_vehicles"] = random.randint(nv_lb, nv_ub)
    # random demands.
    data['demands'] = [0]
    for i in range(number_loc - 1):
```

```
        data['demands'].append(random.randint(dem_lb, dem_ub))
    # Check for capacities equal and zero slack feasibility
    if zero_slack == True:
        not_zs = True
        while not_zs == True :
            if sum(data['demands'])%data['num_vehicles'] == 0:
                not_zs = False
            else :
                data['demands'] = [0]
                for i in range(number_loc - 1):
                    data['demands'].append(random.randint(dem_lb, dem_ub))
    # number of locations without start.
    data['num_demands'] = len(data['demands']) - 1
    # random vehicles capacities.
    data["vehicle_capacities"] = []
    if capacities_equal == True :
        for i in range(data["num_vehicles"]):
            data["vehicle_capacities"].append(round(sum(data['demands'])/data["num_vehicles"]))
        # Feasibility check.
        while sum(data['vehicle_capacities']) < sum(data['demands']):
            data['vehicle_capacities'] = [x + 1 for x in data['vehicle_capacities']]
    if capacities_equal == False :
        random_floats = np.random.dirichlet(np.ones(data["num_vehicles"]))
        for i in range(data["num_vehicles"]):
            data["vehicle_capacities"].append(round(sum(data['demands'])*random_floats[i]))
        data['vehicle_capacities'].sort(reverse = True)
        # Feasibility check.
        while sum(data['vehicle_capacities']) != sum(data['demands']):
            if sum(data['vehicle_capacities']) > sum(data['demands']):
                data['vehicle_capacities'][0] -= 1
            if sum(data['vehicle_capacities']) < sum(data['demands']):
                data['vehicle_capacities'][0] += 1
    # Overslack
    if slack > 0:
        while  sum(data["vehicle_capacities"][:-slack]) < sum(data['demands']):
            if capacities_equal == True :
                data['vehicle_capacities'] = [x + 1 for x in data['vehicle_capacities']]
            if capacities_equal == False :
                data['vehicle_capacities'][random.randint(0,data['num_vehicles']-1)] += 1
                data['vehicle_capacities'].sort(reverse = True)
    # Round vehicle capaciies.
    if rounding == True :
        for i in range(data['num_vehicles']):
            while data['vehicle_capacities'][i]%10 != 0:
                data['vehicle_capacities'][i] += 1
    # random coordinates
    locs = {}
    for i in range(number_loc):
        locs[f'location_{i}'] = (random.randint(0, 10000), random.randint(0, 10000))
    # location distances
    data["distance_matrix"] = []
    Distance_Matrix = pd.DataFrame(columns = [str(i) for i in range(number_loc)])
    for i in range(number_loc):
        iline = []
        for j in range(number_loc):
            iline.append(round(((locs[f'location_{i}'][0]-locs[f'location_{j}'][0])**2 +
                                (locs[f'location_{i}'][1]-locs[f'location_{j}'][1])**2)**0.5))
        Distance_Matrix.loc[len(Distance_Matrix)] = iline
        data["distance_matrix"].append(iline)
    data["depot"] = 0
    # Display Data
```

```
    if output == True :
        print('---------------PROBLEM DATA---------------')
        print('Number of locations = ',number_loc)
        print('Sum demands = ',sum(data['demands']))
        print('Sum capacities = ', sum(data["vehicle_capacities"]))
        print('Number of vehicles = ', data['num_vehicles'])
        print('Vehicles capacities = ', data['vehicle_capacities'])
        print('Number of demands = ', data['num_demands'])
        print('Demand levels = ', data['demands'])
        print('Location Coordinates :')
        print(locs)
        #print(Distance_Matrix)
    return data, locs, number_loc, Distance_Matrix
```

Στη διαδικασία σύγκρισης των αλγορίθμων επίλυσης ο χρήστης καλείται να λύσει ένα σχετικά μεγάλο αριθμό προβλημάτων, ώστε τα αποτελέσματα που θα πάρει να έχουν ακρίβεια. Άρα δημιουργείτε ανάγκη εύρεσης μέσου, που θα δίνει με ευκολία και ταχύτητα, πρόσβαση σε μεγάλο αριθμό προβλημάτων. Για αυτό το λόγο αναπτύχθηκε κώδικας που δημιουργεί τυχαία, θεωρητικά προβλήματα CVRP. Στο απόσπασμα που φαίνεται παραπάνω παρουσιάζεται ο ορισμός της συνάρτησης data_generation η οποία όταν καλείται επιστρέφει τα τυχαία δεδομένα του εκάστοτε προβλήματος. Αυτό δίνει την δυνατότητα στον χρήστη να λύσει με ευκολία ένα μεγάλο αριθμό προβλημάτων.

Οι συνδυασμοί που μπορούν να δημιουργηθούν από την γεννήτρια είναι άπειροι. Αυτό τις περισσότερες φορές είναι αρνητικό για τον χρήστη, καθώς τα προβλήματα που συγκρίνονται ενδεχομένως να είναι τόσο διαφορετικά που να είναι μη συγκρίσιμα. Π.χ. ένα πρόβλημα με 10 πελάτες και ένα πρόβλημα με 1000 πελάτες. (Ακόμα πιο σημαντικό είναι το πρόβλημα να είναι εφικτό. Π.χ. να μην επαρκεί η συνολική χωρητικότητα των οχημάτων στην συνολική ζήτηση των πελατών. Αλλά αυτή η περίπτωση θα εξεταστεί αργότερα). Άρα πρέπει να υπάρχει δυνατότητα ελέγχου των δεδομένων που γεννιόνται. Για αυτό η data_generation ορίζει τι εξής παραμέτρους.

i. **loc_lb**: κατώτατο όριο του αριθμού των τοποθεσιών του προβλήματος (συμπεριλαμβανομένου και του depot).
ii. **loc_ub**: ανώτατο όριο του αριθμού των τοποθεσιών του προβλήματος (συμπεριλαμβανομένου και του depot).
iii. **nv_lb**: κατώτατο όριο του αριθμού των οχημάτων.

iv. **nv_ub**: ανώτατο όριο του αριθμού των οχημάτων.

v. **dem_lb**: κατώτατο όριο του μεγέθους της ζήτησης ανά πελάτη.

vi. **dem_ub**: ανώτατο όριο του μεγέθους της ζήτησης ανά πελάτη.

vii. **capacities_equal**: δυαδική παράμετρος (True ή False), ορίζει αν οι χωρητικότητες των οχημάτων είναι ίσες ή όχι.

viii. **rounding**: δυαδική παράμετρος (True ή False), ορίζει αν οι χωρητικότητες των οχημάτων θα στρογγυλοποιηθούν (πάντα προς τα πάνω) στον πλησιέστερο διαιρέτη του 10.

ix. **slack**: αριθμητική παράμετρος, ορίζει των αριθμό των οχημάτων όπου μπορούν να καθιστούν περιττά (π.χ. αν το πρόβλημα είχε πέντε οχήματα και το slack ήταν δύο, θα πρέπει να υπήρχε εφικτή λύση όπου χρησιμοποιούνται μόνο τα τρία οχήματα και τα υπόλοιπα δύο μένουν αδρανή).

x. **zero_slack**: δυαδική παράμετρος (True ή False), στην περίπτωση που οι χωρητικότητες των οχημάτων είναι ίσες με τις ζητήσεις των πελατών και ταυτόχρονα οι χωρητικότητες είναι ίσες μεταξύ τους, πρέπει να είναι True ώστε να κάνει έλεγχο εφικτότητας.

xi. **output**: δυαδική παράμετρος (True ή False), ορίζει αν θα εμφανίσει στην κονσόλα τα δεδομένα του προβλήματος.

Η data_generation, με βάση τις προαναφερθείσες παραμέτρους, εκτελεί έναν αλγόριθμο όπου θα δώσει δεδομένα τα οποία αντιστοιχούν σε ένα πρόβλημα CVRP. Συγκεκριμένα για να οριστεί ένα πρόβλημα χρειάζονται τα εξής δεδομένα.

i. **Αριθμός τοποθεσιών**: οι τοποθεσίες αντιστοιχούν σε πελάτες, πλην μιας που αντιστοιχεί στο κέντρο εξυπηρέτησης.

ii. **Συντεταγμένες τοποθεσιών**: από τις οποίες εξάγεται ο πίνακας αποστάσεων για κάθε τοποθεσία.

iii. **Ζητήσεις πελατών**: το επίπεδο της ζήτησης για κάθε πελάτη.

iv. **Αριθμός οχημάτων**: ποσά οχήματα είναι διαθέσιμα για να καλύψουν την ζήτηση των πελατών.

v. **Χωρητικότητες οχημάτων**: το επίπεδο συνολικής ζήτησης που μπορεί να καλύψει το κάθε όχημα.

Οι διαδικασίες για να γεννηθούν ο **αριθμός τοποθεσιών** και ο **αριθμός οχημάτων** είναι ίδιες. Καλείται τυχαίος ακέραιος αριθμός με ακραίες τιμές τα αντίστοιχα όρια, για τον αριθμό τοποθεσιών τα loc_lb και loc_ub, και για τον αριθμό οχημάτων τα nv_lb και nv_ub.

Με παρόμοιο τρόπο γεννιούνται και η **ζητήσεις πελατών**. Καλούνται τυχαίοι ακέραιοι αριθμοί εντός των ορίων dem_lb και dem_ub και επισυνάπτονται σε μια λίστα, μέχρι να συμπληρωθεί ο αριθμός των πελατών. Η λίστα είναι αρχικοποιημένη με το πρώτο στοιχείο να είναι μηδέν. Αυτό αντιπροσωπεύει το κέντρο εξυπηρέτησης που δεν έχει ζήτηση, προφανώς. Έτσι η λίστα έχει τόσα στοιχεία όσες τοποθεσίες το πρόβλημα.

Οι **χωρητικότητες οχημάτων** γεννιούνται με δυο διαφορετικούς τρόπους, ανάλογα με το τι έχει οριστεί η παράμετρος **capacities_equal** (ισότητα χωρητικοτήτων). Επίσης επεξεργάζονται περεταίρω για την εξασφάλιση της εφικτότητας και από την στρογγυλοποίηση (rounding) και την πλεονάζουσα χωρητικότητα (slack).

Στη περίπτωση όπου αναζητούνται **ίσες χωρητικότητες** σε όλα τα οχήματα, ορίζεται μια λίστα με μήκος όσο και ο αριθμός των οχημάτων. Τα στοιχεία είναι ίσα μεταξύ τους και είναι το ακέραιο πηλίκο του συνόλου των ζητήσεων και του αριθμού οχημάτων. Εάν αναζητούνται προβλήματα χωρίς στρογγυλοποίηση και χωρίς πλεόνασμα χωρητικοτήτων, πρέπει να εξασφαλιστεί ότι το προαναφερθέντα πηλίκο είναι ακέραιο για να διασφαλιστεί η εφικτότητα του προβλήματος. Τότε πρέπει το **zero_slack** να οριστεί True για να ενεργοποιηθεί η υπορουτίνα που εξασφαλίζει την ακεραιότητα του πηλίκου.

Στη περίπτωση όπου αναζητούνται **άνισες χωρητικότητες**, γεννιέται λίστα με τυχαίους δεκαδικούς αριθμούς και πλήθος όσο και τα οχήματα, όπου το άθροισμα τους είναι ίσο με ένα. Αντιστοιχεί δηλαδή σε ποσοστό ζήτησης που καλείτε να καλύψει το κάθε όχημα.

Πολλές φορές, έχει ενδιαφέρων να διερευνηθούν προβλήματα που υπάρχει, τέτοιο πλεόνασμα χωρητικότητας, όπου η ζήτηση μπορεί να καλυφθεί από μικρότερο αριθμό οχημάτων από ότι αυτά που είναι διαθέσιμα. Για να γεννηθούν τέτοια προβλήματα ορίζεται το **slack** ίσο με τον αριθμό των οχημάτων που είναι δυνητικά

αδρανή. Προσοχή δεν είναι απαραίτητο ότι τα οχήματα θα μείνουν αδρανή μετά την επίλυση του προβλήματος.

Για αισθητικούς λόγους δίνεται η επιλογή να στρογγυλοποιούνται οι χωρητικότητες σε πολλαπλάσια του δέκα. Αυτό γίνεται ορίζοντας το **rounding** ως True. Η Στρογγυλοποίηση γίνετε πάντα προς το μεγαλύτερο πολλαπλάσιο, για να εξασφαλιστεί η εφικτότητα. Σαν αποτέλεσμα με την εφαρμογή της στρογγυλοποίησης, προσδίδεται πλεόνασμα στις χωρητικότητες, όχι όμως σε μεγάλα επίπεδα.

Οι **συντεταγμένες τοποθεσιών** είναι τυχαίες και αποτελούνται από δυο ορίσματα, το οριζόντιο και το κατακόρυφο. Κάθε συντεταγμένη κάθε τοποθεσίας μπορεί να πάρει οποιαδήποτε ακέραια τιμή από το μηδέν έως το δέκα χιλιάδες, έτσι δημιουργείται ένα πεδίο 10,000 επί 10,000 όπου βρίσκονται οι τοποθεσίες. Ο πίνακας αποστάσεων εξάγεται βάση των συντεταγμένων από την σχέση: $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ όπου c_{ij} η απόσταση από το σημείο i στο σημείο j.

3.2.3 Κώδικας Επίλυσης με την Μέθοδο CVRP του OR-TOOLS.

```
    # Instantiate the data problem.
    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(len(data["distance_matrix"]),
data["num_vehicles"],
                                           data["depot"])
    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)
```

Ο παραπάνω κώδικας ορίζει τον διαχειριστή μεταβλητών (**manager**) και μορφοποιεί το μοντέλο δρομολόγησης (**routing**). Οι παράμετροι της μεθόδου **RoutingIndexManager** είναι:

i. Ο **αριθμός γραμμών του πίνακα αποστάσεων**, που είναι ο αριθμός των τοποθεσιών.
ii. Ο **αριθμός των οχημάτων** του προβλήματος.
iii. Ο **κόμβος** που αντιστοιχεί στο κέντρο εξυπηρέτησης.

```
    # Create and register a transit callback.
```

```
    def distance_callback_vrp(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data["distance_matrix"][from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback_vrp)
```

Για να γίνει χρήση του **solver** (δομοστοιχείο του or-tools που λύνει το πρόβλημα) πρέπει να οριστεί η συνάρτηση **distance_callback.** Η συνάρτηση επιστρέφει την απόσταση για κάθε ζευγάρι κόμβων. Μέθοδος **IndexToNode** μεταφράζει τους εσωτερικούς δείκτες του solver σε αριθμούς των προορισμών. Το παραπάνω απόσπασμα ορίζει την **distance_callback** και την καταθέτει στον solver ως **transit_callback_index**.

```
    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
```

Η μέθοδος **SerArcCostEvaluatorOfAllVehicles** καλείτε ώστε να υπολογίσει ο solver το κόστος για την μετακίνηση που προκύπτει για κάθε ζευγάρι τοποθεσιών.

```
    # Add Capacity constraint.
    def demand_callback_vrp(from_index):
        """Returns the demand of the node."""
        # Convert from routing variable Index to demands NodeIndex.
        from_node = manager.IndexToNode(from_index)
        return data["demands"][from_node]

    demand_callback_index = routing.RegisterUnaryTransitCallback(demand_callback_vrp)
    routing.AddDimensionWithVehicleCapacity(
        demand_callback_index,
        0,  # null capacity slack
        data["vehicle_capacities"],
        # vehicle maximum capacities
        True,  # start cumul to zero
        "Capacity",
    )
```

Όπως και το κόστος μετακίνησης, οι χωρητικότητες πρέπει να δηλωθούν με στο μοντέλο με τρόπο που αναγνωρίζει ο solver. Για αυτό χρησιμοποιείται η συνάρτηση **demand_callback_vrp**. Η μέθοδος **AddDimentionWithVehicleCapacity** δημιουργεί μια διάσταση χωρητικοτήτων παρόμοια με αυτή των αποστάσεων.

```
    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
```

```
    search_parameters.first_solution_strategy =
                                (routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
```

Στο απόσπασμα ορίζονται οι παράμετροι αναζήτησης και η ευρετική μέθοδος για να βρεθεί η πρώτη λύση. Η μέθοδος που καλείται είναι η **PATH_CHEAPEST_ARC**.

```
    search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.FromSeconds(time_limit)
```

Η πρώτη λύση δεν είναι απαραίτητα και βέλτιστη. Ο ευρτικός αλγόριθμος μπορεί να παγιδευτεί σε τοπικό βέλτιστο με αποτέλεσμα να αγνοείται μεγάλο μέρος πιθανών λύσεων. Αν το τοπικό ελάχιστο που βρήκε ο αλγόριθμος ως πρώτη λύση δεν είναι και ολικό τότε δεν έχει βρεθεί βέλτιστη λύση. Για αυτό στο τελευταίο απόσπασμα χρησιμοποιείται μια πιο εξελιγμένη μεταευρτική μεθοδολογία που επιτρέπει στον solver να δραπετεύσει από το τοπικό ελάχιστο. Αφού απομακρυνθεί από το τοπικό ελάχιστο ο solver συνεχίζει την αναζήτηση. Η αναζήτηση είναι εξαιρετικά χρονοβόρα καθώς τα προβλήματα CVRP είναι υπολογιστικά δυσεπίλυτα. Για αυτό πρέπει να οριστεί συνθήκη τερματισμού. Στην προκειμένη περίπτωση ορίζεται χρονικό περιθώριο για να ολοκληρωθεί η αναζήτηση, με την μέθοδο **time_limit.FromSeconds()**.

```
        # Solve the problem.
    solution = routing.SolveWithParameters(search_parameters)

    # Print and Save solution on console.
    if solution:
        routes_vrp = get_routes_vrp(solution, routing, manager)
        obj_vrp = solution.ObjectiveValue()
        if output == True :
            print('---------------SOLUTION CVRP---------------')
            print_solution_vrp(data, manager, routing, solution)
            print('Routes :')
            print(routes_vrp)
            print("Solver status: ", routing.status())
            solver_status = routing.status()
```

Τέλος καλείται ο solver να ξεκινήσει την διαδικασία αναζήτησης βέλτιστης λύσης. Ο κώδικας επιστρέφει την λύση στην κονσόλα και την αποθηκεύει στην μνήμη.

```
    # solution formate function
    def print_solution_vrp(data, manager, routing, solution):
        """Prints solution on console."""
        print(f"Objective: {solution.ObjectiveValue()}")
```

```
        total_distance = 0
        total_load = 0
        for vehicle_id in range(data["num_vehicles"]):
            index = routing.Start(vehicle_id)
            plan_output = f"Route for vehicle {vehicle_id}:\n"
            route_distance = 0
            route_load = 0
            while not routing.IsEnd(index):
                node_index = manager.IndexToNode(index)
                route_load += data["demands"][node_index]
                plan_output += f" {node_index} Load({route_load}) -> "
                previous_index = index
                index = solution.Value(routing.NextVar(index))
                route_distance += routing.GetArcCostForVehicle(previous_index, index, vehicle_id)
            plan_output += f" {manager.IndexToNode(index)} Load({route_load})\n"
            plan_output += f"Distance of the route: {route_distance}m\n"
            plan_output += f"Load of the route: {route_load}\n"
            print(plan_output)
            total_distance += route_distance
            total_load += route_load
        print(f"Total distance of all routes: {total_distance}m")
        print(f"Total load of all routes: {total_load}"
```

Η συνάρτηση παραπάνω εκτυπώνει στην κονσόλα την λύση του προβλήματος. Επίσης εμφανίζει το αυξανόμενο φορτίο που αφήνει το όχημα καθώς περνάει από τους σταθμούς.

```
    # routes data
    def get_routes_vrp(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))
                route.append(manager.IndexToNode(index))
            routes.append(route)
        return routes
```

Η συνάρτηση αποθηκεύει τα δρομολόγια όπως υπολογίστηκαν από τον solver.

3.2.4 Κώδικας Επίλυσης CVRP με την Μέθοδο Ανάθεσης και TSP

3.2.4.1 Κώδικας Ανάθεσης.

```
    # Assignment
    # Solver
    # Create the mip solver with the SCIP backend.
    solver = pywraplp.Solver.CreateSolver("SCIP")
```

```
    if not solver:
        print("not solver")
```

Ορίζεται το μοντέλο προβλήματος ανάθεσης και ο MIP solver.

```
    # Variables
    # x[i, j] is an array of 0-1 variables, which will be 1
    # if worker i is assigned to task j.
    x = {}
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            x[i, j] = solver.IntVar(0, 1, "")
```

Δημιουργούνται δυαδικές ακέραιες μεταβλητές για το πρόβλημα. Η μεταβλητές αντιπροσωπεύουν την απόφαση αν θα ανατεθεί το όχημα i στον πελάτη j.

```
    # Constraints
    # Each vehicle does not exceed its capacities.
    for i in range(data["num_vehicles"]):
        solver.Add(solver.Sum([x[i, j]*data["demands"][j+1] for j in range(data["num_demands"])]) <=
                                                                    data["vehicle_capacities"][i])
    # Each demand is assigned to exactly one vehicle.
    for j in range(data["num_demands"]):
        solver.Add(solver.Sum([x[i, j] for i in range(data["num_vehicles"])]) == 1)
```

Ορίζονται οι περιορισμοί του προβλήματος. Η πρώτη σειρά περιορισμών εξασφαλίζει ότι δεν θα υπερβούν οι χωρητικότητες των οχημάτων. Η δεύτερη σειρά περιορισμών εξασφαλίζει ότι κάθε πελάτης θα εξυπηρετηθεί από ένα ακριβώς όχημα.

```
    # Objective Σ x(i,j)
    objective_terms = []
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            objective_terms.append(x[i, j])
    solver.Minimize(solver.Sum(objective_terms))
```

Ορισμός αντικειμενικής συνάρτησης και του προβλήματος ως πρόβλημα ελαχιστοποίησης. Η αντικειμενική συνάρτηση είναι το άθροισμα όλων των μεταβλητών x_{ij}, δηλαδή περιγράφεται από τη σχέση $\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}$ όπου n ο αριθμός οχημάτων και m ο αριθμός πελατών. Το γεγονός ότι η αντικειμενική συνάρτηση αποτελείται μόνο από μεταβλητές απόφασης είναι προβληματικό. Για να καλυφθεί η ζήτηση όλων των πελατών και ταυτόχρονα κάθε πελάτης να εξυπηρετείτε από ένα όχημα πρέπει το άθροισμα των μεταβλητών απόφασης x_{ij} να είναι συγκεκριμένο, δηλαδή ίσο με το m. Το μοντέλο με αύτη την αντικειμενική είναι ικανό να δώσει μόνο

μια εφικτή λύση που συνήθως θα απέχει πολύ από την βέλτιστη. Σε επόμενη ενότητα θα παρουσιαστεί αντικειμενική που δίνει καλύτερες λύσεις.

```
    # Solve
    print(f"Solving with {solver.SolverVersion()}")
    status = solver.Solve()
```

Καλείτε ο solver για την επίλυση του προβλήματος ανάθεσης.

```
    # Print assignment solution.
    if output == True :
        print('---------------SOLUTION FEASIBLE ASSIGNMENT AND TSP---------------
')
        # Print assignment solution.
        if status == pywraplp.Solver.OPTIMAL or status ==
pywraplp.Solver.FEASIBLE:
            print(f"Total cost = {solver.Objective().Value()}\n")
            for i in range(data["num_vehicles"]):
                for j in range(data["num_demands"]):
                    # Test if x[i,j] is 1 (with tolerance for floating point
arithmetic).
                    if x[i, j].solution_value() > 0.5:
                        print(f"Vehicle {i} assigned to location {j+1}.")
        else:
            print("No solution found.")
```

Εμφάνιση των αποτελεσμάτων.

3.2.4.2 Κώδικας TSP.

```
    # Create the data from assignment to use for TSP
    # Separate locations as assigned to vehicles
    routes_tsp_dict = {}
    for i in range(data["num_vehicles"]):
        routes_tsp_dict[f'vehicle{i}'] = [0]
        for j in range(data["num_demands"]):
            # Test if x[i,j] is 1 (with tolerance for floating point arithmetic).
            if x[i, j].solution_value() > 0.5:
                routes_tsp_dict[f"vehicle{i}"].append(j+1)
        routes_tsp_dict[f'vehicle{i}'].append(0)
    # Data
    data_tsp = {}
    for i in range(data["num_vehicles"]):
        data_tsp[f"distance_matrix{i}"] = []
        data_tsp[f"demands{i}"] = []
        data_tsp[f"vehicle{i}_cap"] = data["vehicle_capacities"][i]
        for l in range(len(routes_tsp_dict[f"vehicle{i}"][:-1])):
            data_tsp[f"distance_matrix{i}"].append([])
            m = 0
        for j in routes_tsp_dict[f"vehicle{i}"][:-1]:
```

```
            data_tsp[f"demands{i}"].append(data["demands"][j])
            for k in routes_tsp_dict[f"vehicle{i}"][:-1]:
                data_tsp[f"distance_matrix{i}"][m].append(data["distance_matrix"][j][k])
            m += 1
    data_tsp["num_vehicles"] = 1
    data_tsp["depot"] = 0
```

Σε αυτό το σημείο του αλγορίθμου πρέπει να λυθεί πρόβλημα TSP για κάθε ένα όχημα ξεχωριστά. Κάθε πρόβλημα TSP έχει ξεχωριστά δεδομένα τα οποία πρέπει να αντιπροσωπεύουν μόνο πελάτες που έχουν ανατεθεί στο όχημα. Με απλά πρέπει να χωριστούν τα δεδομένα για κάθε πελάτη με βάση την ανάθεση που έγινε νωρίτερα. Αυτά τα δεδομένα αντλούνται από το απόσπασμα που φαίνεται παραπάνω. Αρχικά αποθηκεύονται σε ξεχωριστές μεταβλητές οι τοποθεσίες που έχουν ανατεθεί σε κάθε όχημα, συμπεριλαμβανομένου και του κέντρου εξυπηρέτη-σης. Με βάση αυτές τις μεταβλητές εξάγει από τον πίνακα αποστάσεων τις αποστάσεις που αφορούν τις τοποθεσίες του TSP προβλήματος. Στη συνέχεια επισυνάπτονται σε ξεχωριστό πίνακα αποστάσεων. Στο TSP πρόβλημα ο αριθμός οχημάτων είναι πάντα μονάδα. Οι χωρητικότητες δεν λαμβάνονται υπόψιν.

```
    # TSP
    # Print solution for each tsp problem
    def print_solution(manager, routing, solution):
        """Prints solution on console."""
        print(f"TSP_{i} Objecive value (distance) : {solution.ObjectiveValue()} ")
        index = routing.Start(0)
        plan_output = f"Route for vehicle {i}:\n"
        route_distance = 0
        route_load = 0
        while not routing.IsEnd(index):
            route_load += data_tsp[f'demands{i}'][manager.IndexToNode(index)]
            plan_output += f" {routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]}
Load({route_load}) ->"
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
        plan_output += f" {routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]}
Load({route_load}) \n"
        plan_output += f"Distance of the route: {route_distance}\n"
        plan_output += f"Load of the route: {route_load}\n"
        print(plan_output)

    # Routes for tsp problems
    def get_routes(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes_tsp = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
```

```
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))
                route.append(routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)])
            routes_tsp.append(route)
        return routes_tsp

    # Run TSP for all vehicles
    obj_tsp = 0
    routes_tsp = []
    for i in range(data["num_vehicles"]):

        # Create the routing index manager.
        manager = pywrapcp.RoutingIndexManager(
            len(data_tsp[f"distance_matrix{i}"]), data_tsp["num_vehicles"], data_tsp["depot"])

        # Create Routing Model.
        routing = pywrapcp.RoutingModel(manager)

        def distance_callback(from_index, to_index):
            """Returns the distance between the two nodes."""
            # Convert from routing variable Index to distance matrix NodeIndex.
            from_node = manager.IndexToNode(from_index)
            to_node = manager.IndexToNode(to_index)
            return data_tsp[f"distance_matrix{i}"][from_node][to_node]
        transit_callback_index = routing.RegisterTransitCallback(distance_callback)
        # Define cost of each arc.
        routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
        # Setting first solution heuristic.
        search_parameters = pywrapcp.DefaultRoutingSearchParameters()
        search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
        search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
        search_parameters.time_limit.FromSeconds(time_limit)
        # Solve the problem.
        solution = routing.SolveWithParameters(search_parameters)

        # Print and Save solution on console.
        if solution:
            if output == True :
                print_solution(manager, routing, solution)
            obj_tsp += solution.ObjectiveValue()
            routes_tsp.append(get_routes(solution, routing, manager)[0])
    if output == True :
        print("Objective value =",obj_tsp)
        print("All routes :\n", routes_tsp)
```

Ο κώδικας επίλυσης του προβλήματος TSP είναι σχεδόν ίδιος με αυτόν του προβλήματος CVRP, που περιεγράφηκε σε προηγούμενη ενότητα. Οι διαφορές είναι ότι απουσιάζουν οι εντολές που αφορούν τους περιορισμούς των χωρητικοτήτων. Επίσης αναζητεί λύσεις για τα δεδομένα που αντλήθηκαν νωρίτερα, όχι από τα δεδομένα όλου του CVRP.

3.2.5 Κώδικας Συσταδοποίησης.

```
    # tsp for one vehicle with unlimited capacity
    # presolution for assignment problem

    def print_solution_TSP(manager, routing, solution):
        """Prints solution on console."""
        print(f"Objective TSP SINGLE VEHICLE: {solution.ObjectiveValue()}")
        index = routing.Start(0)
        plan_output = "Route for vehicle :\n"
        route_distance = 0
        while not routing.IsEnd(index):
            plan_output += f" {manager.IndexToNode(index)} ->"
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
        plan_output += f" {manager.IndexToNode(index)}\n"
        print(plan_output)

    def get_routes_TSP(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))
                route.append(manager.IndexToNode(index))
            routes.append(route)
        return routes

    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(
            len(data["distance_matrix"]), 1, data["depot"])
    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)


    def distance_callback_TSP(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data["distance_matrix"][from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback_TSP)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.FromSeconds(time_limit)
```

```
58
59      # Solve the problem.
60      solution = routing.SolveWithParameters(search_parameters)
61
62      # Print on console and Save solution.
63      if solution:
64          if output == True :
65              print('---------------SOLUTION TSP SINGLE VEHICLE---------------')
66              print_solution_TSP(manager, routing, solution)
67          routes_TSP = get_routes_TSP(solution, routing, manager)
68          obj_TSP = solution.ObjectiveValue()
```

Η πρώτη ενέργεια του κώδικα είναι να λύσει το πρόβλημα θεωρώντας ότι είναι πρόβλημα TSP. Αυτό είναι θεωρητικό πρόβλημα και το αποτέλεσμα της λύσης του δεν είναι εφικτό στην πραγματικότητα. Η λύση του προβλήματος θα λειτουργήσει σαν οδηγός για την συσταδοποίηση. Από την λύση αυτό που λαμβάνεται υπόψιν είναι η διαδρομή του οχήματος. Η επίλυση με συσταδοποίηση μιμείται αυτή τη διαδρομή. Η βελτιστότητα της λύσης νομιμοποιείται από το γεγονός ότι αυτή η διαδρομή είναι βέλτιστη για το πρόβλημα TSP. Στη πραγματικότητα η λύση δεν είναι βέλτιστη αλλά δύναται να δίνει καλύτερη λύση από αυτή της απλής ανάθεσης χωρίς συσταδοποίηση. Στο απόσπασμα παραπάνω γίνεται η επίλυση του TSP. Ακολουθεί την ίδια μεθοδολογία με το κώδικα του TSP που αναλύθηκε σε προηγούμενη ενότητα.

```
1       # Create clusters
2       clusters = {}
3       clusters["cluster_0"] = []
4       load = 0
5       vehicle_index = 0
6       for i in routes_TSP[0][1:-1]:
7           # Add locations to clusters
8           if data['vehicle_capacities'][vehicle_index] >= load + data["demands"][i]:
9                                                   # if next load fits in current vehicle
10              load += data["demands"][i]
11              clusters[f"cluster_{vehicle_index}"].append(i)
12          # if current vehicle is last add remaining locations to last cluster
13          elif data['vehicle_capacities'][vehicle_index] <= load + data["demands"][i]
14                                              and vehicle_index == data['num_vehicles'] - 1 :
15              clusters[f"cluster_{vehicle_index}"].append(i)
16          # if next load doesn't fit in current vehicle
17          else:
18              vehicle_index += 1
19              clusters[f"cluster_{vehicle_index}"] = []
20              clusters[f"cluster_{vehicle_index}"].append(i)
21              load = 0
22              load += data["demands"][i]
```

Στο παραπάνω απόσπασμα ο κώδικας δημιουργεί τις συστάδες. Οι τοποθεσίες προσθέτονται στις συστάδες με τη σειρά που ακολουθεί η διαδρομή που βρέθηκε από το TSP νωρίτερα. Ο αριθμός τοποθεσιών που μπορεί να δεχθεί μια συστάδα

εξαρτάται από τις χωρητικότητες των οχημάτων και από μέγεθος των ζητήσεων. Για να προσθέσει ο αλγόριθμος μια τοποθεσία σε μια συστάδα, ελέγχει τη ζήτηση της τοποθεσίας, τις ζητήσεις των τοποθεσιών που υπάρχουν ήδη μέσα στη συστάδα, και την χωρητικότητα του οχήματος που αντιστοιχεί η συστάδα. Αν το άθροισμα της ζήτησης της νέας τοποθεσίας με τις προηγούμενες τοποθεσίες δεν υπερβαίνει τη χωρητικότητα του οχήματος τότε η τοποθεσία προσθέτετε στη συστάδα. Αν το άθροισμα υπερβαίνει την χωρητικότητα του οχήματος τότε η τοποθεσία προωθείται στην επόμενη συστάδα. Αν το άθροισμα υπερβαίνει την χωρητικότητα του οχήματος και το όχημα είναι το τελευταίο στη λίστα τότε η τοποθεσία προσθέτετε στη συστάδα μαζί με τις επόμενες. Το κέντρο εξυπηρέτησης δεν μπαίνει σε κάποια συστάδα.

```
    # Cluster center coordinates
    cluster_center_coords = {}
    for i in range(len(clusters)):
        cluster_center_coords[f'center_{i}_coords'] = [0,0]
        div = len(clusters[f'cluster_{i}'])
        for j in clusters[f'cluster_{i}']:
            cluster_center_coords[f'center_{i}_coords'][0] += locs[f'location_{j}'][0]/div
            cluster_center_coords[f'center_{i}_coords'][1] += locs[f'location_{j}'][1]/div
```

Στο παραπάνω απόσπασμα ο κώδικας επιστρέφει τα γεωμετρικά κέντρα των συστάδων. Το κάθε κέντρο χαρακτηρίζεται από την οριζόντια μεταβλητή **x,** και την καθετή **y**. Η σχέση για την εύρεση του κέντρου είναι **(x,y) =** $(\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{n}\sum_{i=n}^{n} y_i)$, όπου x_i και y_i οι συντεταγμένες της τοποθεσίας i και n το σύνολο των τοποθεσιών της συστάδας.

```
    # Distance matrix of nodes to clusters
    data_cl = {}
    data_cl['cluster_distance_martix'] = []
    for i in range(len(cluster_center_coords)):
        iline = []
        for j in range(1,number_loc):
            iline.append(round(((cluster_center_coords[f'center_{i}_coords'][0]-
            locs[f'location_{j}'][0])**2 + (cluster_center_coords[f'center_{i}_coords'][1]-
locs[f'location_{j}'][1])**2)**0.5))
        data_cl['cluster_distance_martix'].append(iline)
    if len(clusters) < data['num_vehicles']:
        while len(data_cl['cluster_distance_martix']) != data['num_vehicles']:
            data_cl['cluster_distance_martix'].append([10000000 for i in
range(data['num_demands'])])
```

Στη συνέχεια ο κώδικας δημιουργεί τον πίνακα αποστάσεων μεταξύ των πελατών και των κέντρων συστάδων. Ο πίνακας αποστάσεων εξάγεται από την σχέση: $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ όπου c_{ij} η απόσταση από το κέντρο συστάδας i στον πελάτη j. Στη περίπτωση που οι συστάδες είναι λιγότερες από τα οχήματα η απόσταση της συστάδας από τους πελάτες ορίζεται το μεγάλο M, M = 10000000.

3.2.6 Κώδικας Επίλυσης CVRP με τη Μέθοδο Συστάδων.

Η επίλυση με την μέθοδο συστάδων βασίζεται στη μέθοδο Ανάθεσης και TSP, που είδαμε στην ενότητα 3.2.4. Η ειδοποιός διαφορά τους είναι στην αντικειμενική συνάρτηση του σκέλους της ανάθεσης.

```
    # Objective Σ x(i,j)*distance(i_cluster to j_demand)
    objective_terms = []
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            objective_terms.append(x[i, j]*data_cl['cluster_distance_martix'][i][j])
    solver.Minimize(solver.Sum(objective_terms))
```

Η αντικειμενική συνάρτηση είναι το άθροισμα όλων των μεταβλητών απόφασης x_{ij} επί την απόσταση από τη συστάδα i στον πελάτη j , δηλαδή περιγράφεται από τη σχέση $\boldsymbol{z = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} c_{ij}}$ όπου n ο αριθμός οχημάτων και m ο αριθμός πελατών.

Επίσης δοκιμάζεται και η αντικειμενική με την απόσταση υψωμένη στο τετράγωνο.

```
    # Objective Σ x(i,j)*distance(i_cluster to j_demand)^2
    objective_terms = []
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            objective_terms.append(x[i, j]*(data_cl['cluster_distance_martix'][i][j]**2))
    solver.Minimize(solver.Sum(objective_terms))
```

$\boldsymbol{z = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} c_{ij}^2}$.

4 Αποτελέσματα.

Σε αυτό το κεφάλαιο παρουσιάζονται τα αποτελέσματα, που έδωσαν οι τέσσερίς αλγόριθμοι που συγκρίνονται σε αυτή τη διατριβή, για δέκα διαφορετικά προβλήματα CVRP. Τα προβλήματα που εξετάζονται δημιουργήθηκαν τυχαία χρησιμοποιώντας την γεννήτρια τυχαίων προβλημάτων, που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Κάνοντας χρήση των ορισμάτων της γεννήτριας, τα προβλήματα μπορούν να αποκτήσουν ποικίλα χαρακτηριστικά, δίνοντας έτσι την δυνατότητα αξιολόγησης της αποδοτικότητας των αλγορίθμων σε διαφορετικές συνθήκες. Οι αλγόριθμοι που αναπτύχθηκαν σε αυτή τη διατριβή μπορούν να κάνουν χρήση όλων των μεταευρετηκών μεθόδων που προσφέρει το OR TOOLS. Για την επίδειξη αυτού του χαρακτηριστικού τα προβλήματα λύνονται είτε με Tabu Search είτε με Guided Local Search.

Οι ιδιότητες των προβλημάτων που μπορούν να ελεγχθούν από τα ορίσματα της γεννήτριας, αφορούν την σχέση των χωρητικοτήτων τόσο μεταξύ των οχημάτων όσο και μεταξύ του στόλου και των πελατών. Σχετικά με την πρώτη περίπτωση, ορισμένα προβλήματα παρουσιάζουν ομοιόμορφες χωρητικότητες οχημάτων, ενώ σε άλλα οι χωρητικότητες διαφέρουν από όχημα σε όχημα. Αυτή η διαφοροποίηση προσθέτει έναν επιπλέον παράγοντα πολυπλοκότητας, καθώς οι μέθοδοι καλούνται να λάβουν υπόψη τις διαφορές αυτές για την αποτελεσματικότερη κατανομή των φορτίων. Στα πλαίσια αυτής της διπλωματικής ομοιόμορφα χαρακτηρίζονται τα προβλήματα στα οποία όλα τα οχήματα έχουν ίδιες χωρητικότητες, ενώ ανομοιόμορφα αυτά στα οποία οι χωρητικότητες των οχημάτων διαφέρουν.

Σε ότι αφορά την σχέση μεταξύ χωρητικότητας στόλου και συνολικής ζήτησης, σημειώνεται ότι σε ορισμένα προβλήματα, η συνολική χωρητικότητα των οχημάτων αντιστοιχεί ακριβώς στη συνολική ζήτηση των πελατών, κάτι που καθιστά το πρόβλημα βέλτιστης κατανομής των πόρων ιδιαίτερα αυστηρό. Αντίθετα σε άλλα, η συνολική χωρητικότητα είναι μεγαλύτερη από τη ζήτηση, επιτρέποντας περισσότερους ελιγμούς στην κατανομή των παραδόσεων και μεγαλύτερη ευελιξία στην κατασκευή των διαδρομών. Η ποικιλομορφία αυτή είναι κρίσιμη, καθώς δίνει τη δυνατότητα στις μεθόδους να δοκιμαστούν σε περιπτώσεις όπου η χωρητικότητα των οχημάτων επηρεάζει την πολυπλοκότητα του προβλήματος και τον τρόπο προσέγγισης της βέλτιστης λύσης. Σε πιο ακραίες περιπτώσεις, η συνολική χωρητικότητα των οχημάτων είναι τόσο υψηλή που θα μπορούσε να καλυφθεί η ζήτηση με ένα όχημα λιγότερο, γεγονός που μας επιτρέπει να αξιολογήσουμε την απόδοση των μεθόδων σε περιπτώσεις πλεονασματικών πόρων και να εξετάσουμε το κατά πόσο μπορούν να επιτύχουν την

ελαχιστοποίηση του αριθμού των διαδρομών ή την αξιοποίηση του πλεονασματικού οχήματος για την μείωση του κόστους. Είναι προφανές ότι μπορούν να χαρακτηριστούν δυο είδη προβλημάτων με πλεονάζουσα χωρητικότητα στόλου. Τα προβλήματα, που ανήκουν στη πρώτη περίπτωση όπου οι χωρητικότητες είναι πλεονάζουσες χωρίς να καθίσταται κανένα όχημα περιττό, ονομάζονται (στα πλαίσια της διπλωματικής) στρογγυλοποιημένα. Ο λόγος είναι, ότι το τέχνασμα που χρησιμοποιεί η γεννήτρια για να το πετύχει αυτό είναι η στρογγυλοποίηση προς τα πάνω. Από την άλλη τα προβλήματα με πλεόνασμα, όπου μπορεί κάποιο όχημα να θεωρηθεί περιττό αποκαλούνται (στα πλαίσια της διπλωματικής) πλεονασματικά. Τέλος για λόγους πληρότητας τα προβλήματα που έχουν ίση χωρητικότητα στόλου με συνολική ζήτηση ονομάζονται μη-πλεονασματικά.

Τα προβλήματα περιλαμβάνουν διαφορετικούς αριθμούς οχημάτων και τοποθεσιών. Ο αριθμός των οχημάτων κυμαίνεται από τρία έως τέσσερα, ενώ οι τοποθεσίες παραδόσεων κυμαίνονται από 30 έως 40.

Η μορφή των δεδομένων που αντιστοιχούν στα δέκα προβλήματα που μελετήθηκαν, είναι κωδικοποιημένη και ακολουθεί την ιδιά σειρά για κάθε πρόβλημα. Με τον τρόπο αυτό μπορούν επιβεβαιωθούν στις κατηγορίες που αναφέρθηκαν νωρίτερα με ευκολία. Τα δεδομένα παρουσιάζονται σε μορφή πινάκων.

Στον πρώτο πίνακα ορίζονται:

- **Αριθμός Τοποθεσιών και Πελατών**: Ο αριθμός των τοποθεσιών περιλαμβάνει το κέντρο εξυπηρέτησης και τις τοποθεσίες πελατών. Ο αριθμός πελατών, δηλαδή των σημείων που χρειάζονται εξυπηρέτηση, προσδιορίζεται ξεχωριστά από το κέντρο εξυπηρέτησης με αποτέλεσμα το πλήθος των τοποθεσιών να είναι πάντα μεγαλύτερο από το πλήθος των πελατών κατά μία μονάδα
- **Αριθμός Οχημάτων**: Ορίζει πόσα οχήματα είναι διαθέσιμα για να καλύψουν τη ζήτηση των πελατών.
- **Συνολική Ζήτηση και Χωρητικότητα**: Αυτά τα δεδομένα συνοψίζουν τη ζήτηση όλων των πελατών και τη συνολική διαθέσιμη χωρητικότητα των οχημάτων. Η συνολική ζήτηση είναι το άθροισμα των φορτίων που χρειάζονται οι πελάτες, ενώ η συνολική χωρητικότητα προκύπτει από το άθροισμα των χωρητικοτήτων όλων των οχημάτων. Η σχέση αυτών των ποσοτήτων χαρακτηρίζει τα προβλήματα ως πλεονασματικά, μη πλεονασματικά και στρογγυλοποιημένα.

Στο δεύτερο πίνακα παρουσιάζονται οι:

- **Χωρητικότητες των Οχημάτων**: Ορίζει την μέγιστη χωρητικότητα για κάθε όχημα ξεχωριστά. Οι σχέσεις των χωρητικοτήτων μεταξύ των οχημάτων τους κατηγοριοποιούν το πρόβλημα σε ομοιόμορφο ή ανομοιόμορφο.

Στον τρίτο πίνακα παρουσιάζονται οι:

- **Θέσεις Τοποθεσίων και οι Ζητήσεις**: Για κάθε τοποθεσία, καταγράφεται η θέση σε συντεταγμένες X (οριζόντια) και Y (κάθετη) και η ζήτηση σε μονάδες φορτίου. Η θέση βοηθά στον καθορισμό το πίνακα αποστάσεων μεταξύ τοποθεσιών, ενώ η ζήτηση είναι η ποσότητα φορτίου που χρειάζεται να παραδοθεί στον κάθε πελάτη. Η ζήτηση της πρώτης τοποθεσίας είναι πάντα μηδέν γιατί αναπαριστά το κέντρο εξυπηρέτησης.

Στη συνέχεια παρουσιάζονται οι λύσεις που έδωσαν οι τέσσερίς αλγόριθμοι. Ο εγγενής αλγόριθμος επίλυσης προβλημάτων CVRP του or tools στη συγκεκριμένη διατριβή αναφέρεται απλά ως αλγόριθμος OR. Ο αλγόριθμος απλής ανάθεσης αναφέρεται ως αλγόριθμος FTSP, από το Feasible assignment and TSP, καθώς στο σκέλος της ανάθεσης το μόνο που ελέγχεται είναι η εφικτότητα της ανάθεσης, όπως διαπιστώθηκε σε προηγούμενο κεφάλαιο. Ο αλγόριθμος με συσταδοποίηση αναφέρεται ως CL από το Clustering, ενώ ο αλγόριθμος συσταδοποίησης με τις αποστάσεις υψωμένες στο τετράγωνο ως CL2. Για λόγους πληρότητας παρουσιάζεται επιπλέων η λύση του προβλήματος TSP που χρησιμοποιείται από τις μεθόδους συσταδοποίησης για την δημιουργία συστάδων. Η λύση δεν συγκρίνεται με τις υπόλοιπες λύσεις.

Όπως και για τα δεδομένα των προβλημάτων οι λύσεις των προβλημάτων παρουσιάζονται με τη μορφή πινάκων. Στους πίνακες ποσοτικοποιούνται οι έννοιες που βοηθούν στη κατανόηση των λύσεων και επιτρέπουν την επαλήθευση τους. Οι έννοιες αυτές περιλαμβάνουν:

- **Αντικειμενική Τιμή**: Η αντικειμενική τιμή αντιπροσωπεύει το συνολικό κόστος της λύσης και είναι ο δείκτης απόδοσης που προσπαθούμε να ελαχιστοποιήσουμε. Στο CVRP, το κόστος αυτό συχνά εκφράζεται ως η συνολική απόσταση που διανύουν τα οχήματα ή ως το συνολικό κόστος διανομής.
- **Δρομολόγιο Οχήματος**: Καταγράφεται το δρομολόγιο που ακολουθεί το κάθε όχημα, περιλαμβάνοντας τη σειρά με την οποία επισκέπτεται κάθε τοποθεσία. Έτσι, για κάθε όχημα προσδιορίζεται η σειρά των πελατών που εξυπηρετεί και το πώς οι διαδρομές αυτές συμβάλλουν στην ελαχιστοποίηση του συνολικού κόστους.

- **Απόσταση Δρομολογίου**: Παρουσιάζεται ξεχωριστά για κάθε όχημα, επιτρέποντας τη σύγκριση των διανυόμενων αποστάσεων μεταξύ οχημάτων. Από το άθροισμα όλων των αποστάσεων δρομολογίων προκύπτει η αντικειμενική τιμή.
- **Φορτίο Δρομολογίου**: Το φορτίο καταγράφεται επίσης για κάθε όχημα ξεχωριστά, δείχνοντας πόσο από το διαθέσιμο φορτίο κάθε οχήματος αξιοποιείται σε κάθε δρομολόγιο. Με τον τρόπο αυτό μπορεί να επαληθευτεί η εφικτότητα.

Επιπλέον τα αποτελέσματα αναπαρίστανται και σε μορφή σχήματος. Στα σχήματα διακρίνονται οι τοποθεσίες ως κουκίδες στις οποίες αναγράφεται ο αριθμός τις τοποθεσίας. Οι ακμές που ενώνουν τις κουκίδες αναπαριστούν τα δρομολόγια ενώ το χρώμα τους δηλώνει το πιο όχημα διασχίζει την εκάστοτε ακμή. Τα οχήματα 1,2,3 και 4 αναπαρίστανται με κόκκινο, πράσινο, μπλε και κίτρινο αντίστοιχα.

Για την εξαγωγή των αποτελεσμάτων η συνθήκη τερματισμού αναζήτησης λύσεων που επιλέχθηκε είναι ίδια για όλους τους αλγορίθμους και έχει οριστεί το χρονικό όριο. Η τιμή του χρονικού ορίου έχει οριστεί στα πέντε λεπτά για κάθε αλγόριθμο ή αλλιώς στα τριακόσια δευτερόλεπτα. Κάθε αλγόριθμος έχει την δυνατότητα να εκτελέσει Tabu Search ή Guided Local Search, ανάλογα με το τι έχει επιλεχθεί για το εκάστοτε πρόβλημα, μέχρι να εξαντληθεί το χρονικό περιθώριο. Πρέπει να σημειωθεί ότι για την επίλυση του εκάστοτε προβλήματος οι αλγόριθμοι πρέπει να κάνουν χρήση της ιδίας μεταευρετικής μεθόδου. Παραδείγματος χάριν δεν γίνεται για το ίδιο πρόβλημα ο αλγόριθμος OR να κάνει χρήση Tabu Search και ο αλγόριθμος FTSP να χρησιμοποιεί Guided Local Search. Η ιδιαιτερότητα που προκύπτει είναι ότι για τους αλγορίθμους FTSP, CL και CL2 πρέπει να οριστεί ξεχωριστά το χρονικό όριο για κάθε TSP πρόβλημα που λύνεται από τους αλγόριθμους, έτσι ώστε το συνολικό χρονικό όριο που αφιερώνεται από τους αλγόριθμους να μην ξεπερνά πέντε λεπτά. Για τους αλγόριθμους CL και CL2 σε αυτά τα TSP συμπεριλαμβάνεται και το TSP που χρησιμοποιείται για τη συσταδοποίηση.

Τα πειράματα εκτελέστηκαν από υπολογιστή με τα εξής χαρακτηριστικά:

- Κατασκευαστής: Dell
- Μοντέλο: DESKTOP-EOIOR5P
- Επεξεργαστής: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- Εγκατεστημένη μνήμη: 16,0 GB (15,9 GB χρησιμοποιήσιμη)
- Λειτουργικό: Windows 11 Home 23H2

Τα λογισμικά που χρησιμοποιηθήκαν είναι τα εξής:

- Python: 3.11.5 64bit [MSC v.1916 64 bit (AMD64)]
- IPython: 8.15.0
- Numpy: 1.24.3
- Pandas: 2.0.3
- Networkx: 3.1
- Matplotlib: 3.7.2
- Or tools: 9.9.3963

Πρόβλημα 1

- Ομοιομορφο Στρογγυλοποιημενο. Επιλυση με Tabu Search.

Δεδομενα.

Αριθμός Τοποθεσιών	34
Αριθμός Πελατών	33
Αριθμός Οχημάτων	4
Σύνολο Ζητήσεων	206
Σύνολο Χωρητικοτήτων	240

Πίνακας 4.1.1 Πίνακας δεδομένων Προβλήματος 1

Όχημα	1	2	3	4
Χωρητικότητα Οχήματος	60	60	60	60

Πίνακας 4.1.2 Πίνακας χωρητικοτήτων Προβλήματος 1

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	174	652	11	1	98	214	21	2	483	226	31	8	174	814
2	9	405	143	12	2	889	971	22	6	468	102	32	4	602	171
3	1	545	122	13	8	734	320	23	9	245	496	33	7	578	637
4	8	290	750	14	10	756	187	24	10	717	208	34	8	341	172
5	4	180	786	15	8	709	380	25	7	716	395				
6	8	354	371	16	2	409	154	26	6	841	530				
7	6	454	428	17	4	364	491	27	9	528	667				
8	1	874	944	18	6	85	580	28	8	723	856				
9	7	3	658	19	9	223	714	29	2	652	957				
10	10	880	396	20	6	976	519	30	10	743	498				

Πίνακας 4.1.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 1

Αποτελέσματα.

	Λύση OR	Λύση FTSP
Αντικειμενική τιμή	6376	10046
Δρομολόγιο Οχήματος 1	0->22->16->5->20->31->2->21->1->15->33->10->17-> 0	0->6->5->10->1->15->2->9->7->11->3->4->8->0
Απόσταση Δρομολογίου 1	1774	3396
Φορτίο Δρομολογίου 1	60	59
Δρομολόγιο Οχήματος 2	0->26->32->25->9->19->11->7->27->28->3-> 0	0->18->19->14->12->13->21->20->16->17->0
Απόσταση Δρομολογίου 2	2350	2480
Φορτίο Δρομολογίου 2	59	59
Δρομολόγιο Οχήματος 3	0->29->24->14->12->13->23->6-> 0	0->30->26->28->27->25->24->23->22->0
Απόσταση Δρομολογίου 3	1658	2425
Φορτίο Δρομολογίου 3	59	59
Δρομολόγιο Οχήματος 4	0->8->30->4->18-> 0	0->32->29->31->33->0
Απόσταση Δρομολογίου 4	594	1745
Φορτίο Δρομολογίου 4	28	29

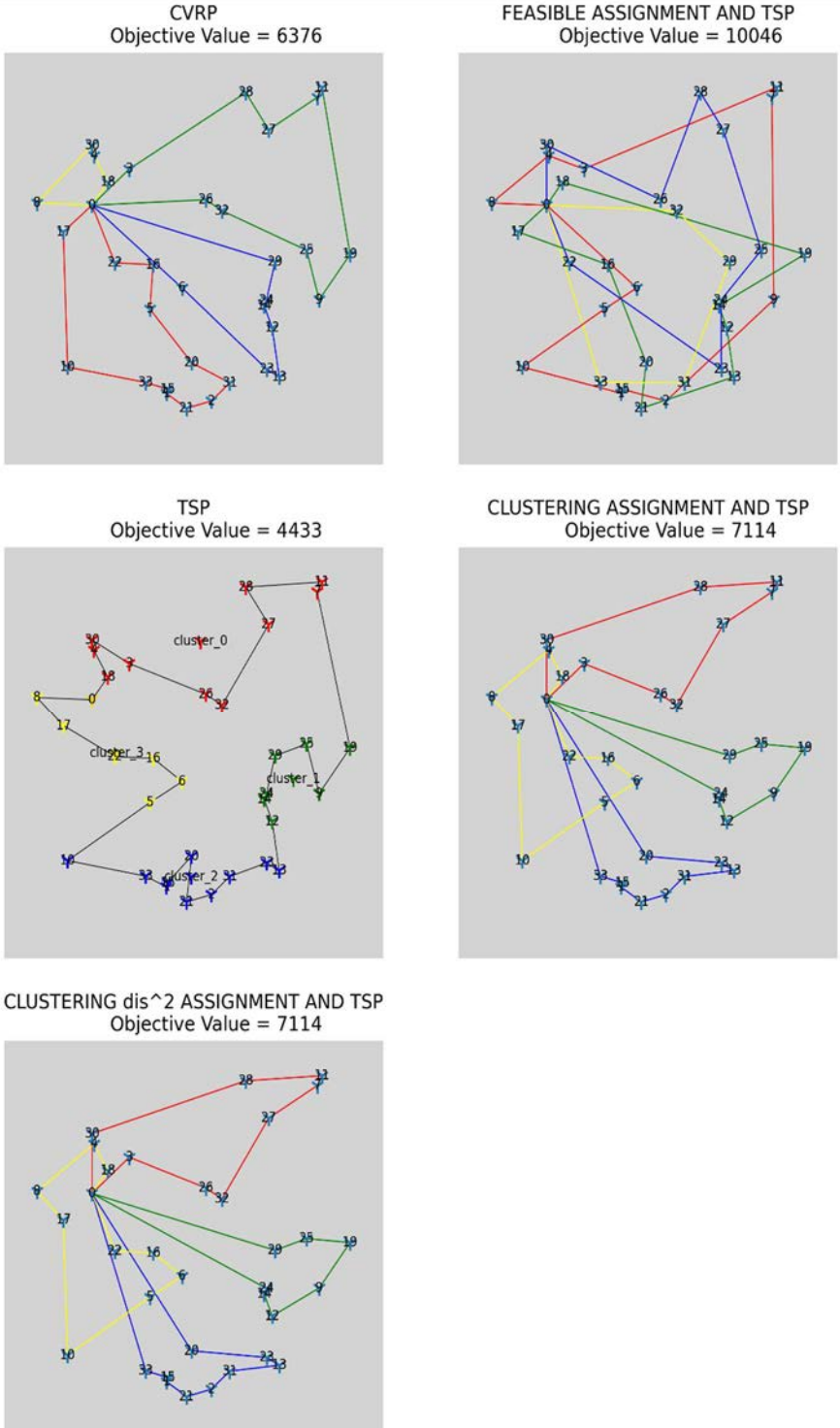
Πίνακας 4.1.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 1

	Λύση TSP
Αντικειμενική τιμή	4433
Δρομολόγιο	0->18->4->30->3->26->32->27->28->11->7->19->9->25->29->24->14->12->13->23->31->2->21->20->15->1->33->10->5->6->16->22->17->8->0

Πίνακας 4.1.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 1

	Λύση CL	Λύση CL2
Αντικειμενική τιμή	7114	7114
Δρομολόγιο Οχήματος 1	0->30->28->11->7->27->32->26->3->0	0->30->28->11->7->27->32->26->3->0
Απόσταση Δρομολογίου 1	1829	1829
Φορτίο Δρομολογίου 1	45	45
Δρομολόγιο Οχήματος 2	0->29->25->19->9->12->14->24->0	0->29->25->19->9->12->14->24->0
Απόσταση Δρομολογίου 2	1830	1830
Φορτίο Δρομολογίου 2	55	55

Δρομολόγιο Οχήματος 3	0->33->15->1->21->2->31->13->23->20->0	0->33->15->1->21->2->31->13->23->20->0
Απόσταση Δρομολογίου 3	1780	1780
Φορτίο Δρομολογίου 3	52	52
Δρομολόγιο Οχήματος 4	0->18->4->8->17->10->5->6->16->22->0	0->18->4->8->17->10->5->6->16->22->0
Απόσταση Δρομολογίου 4	1675	1675
Φορτίο Δρομολογίου 4	54	54



Εικόνα 4.1 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 1

Πρόβλημα 2

- Ομομορφο Μη-Πλεονασματικο. Επιλυση με Guided Local Search.

Δεδομενα

Αριθμός Τοποθεσιών	31
Αριθμός Πελατών	30
Αριθμός Οχημάτων	4
Σύνολο Ζητήσεων	184
Σύνολο Χωρητικοτήτων	184

Πινακας 4.2.1. Πίνακας δεδομένων Προβλήματος 2

Όχημα	1	2	3	4
Χωρητικότητα Οχήματος	46	46	46	46

Πινακας 4.2.2 Πίνακας χωρητικοτήτων Προβλήματος 2

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	328	724	11	2	914	524	21	2	20	40	31	9	213	871
2	7	71	115	12	8	493	472	22	5	744	434				
3	10	167	135	13	2	466	149	23	6	855	823				
4	4	235	342	14	8	603	421	24	2	89	370				
5	9	848	669	15	1	723	492	25	4	278	773				
6	9	627	981	16	9	23	94	26	6	859	618				
7	10	883	623	17	9	56	18	27	9	962	129				
8	5	747	218	18	4	199	736	28	8	600	337				
9	1	25	196	19	4	387	958	29	6	342	3				
10	10	16	871	20	9	135	331	30	6	4	161				

Πινακας 4.2.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 2

Αποτελεσματα

	Λύση OR	Λυση FTSP
Αντικειμενική τιμή	7589	11729
Δρομολόγιο Οχήματος 1	0->18->5->22->4->25->6->10-> 0	0->3->8->1->2->7->14->4->5->0
Απόσταση Δρομολογίου 1	1713	2694
Φορτίο Δρομολογίου 1	46	46
Δρομολόγιο Οχήματος 2	0->11->13->27->12->7->26->21->14-> 0	0->17->9->20->12->11->13->10->6->0
Απόσταση Δρομολογίου 2	2154	3088
Φορτίο Δρομολογίου 2	46	46
Δρομολόγιο Οχήματος 3	0->3->2->19->9->30->24-> 0	0->24->18->22->21->16->15->19->0
Απόσταση Δρομολογίου 3	1748	2764
Φορτίο Δρομολογίου 3	46	46
Δρομολόγιο Οχήματος 4	0->17->23->8->29->1->15->20->16->28-> 0	0->30->23->29->28->27->26->25->0
Απόσταση Δρομολογίου 4	1974	3183
Φορτίο Δρομολογίου 4	46	46

Πινακας 4.2.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 2

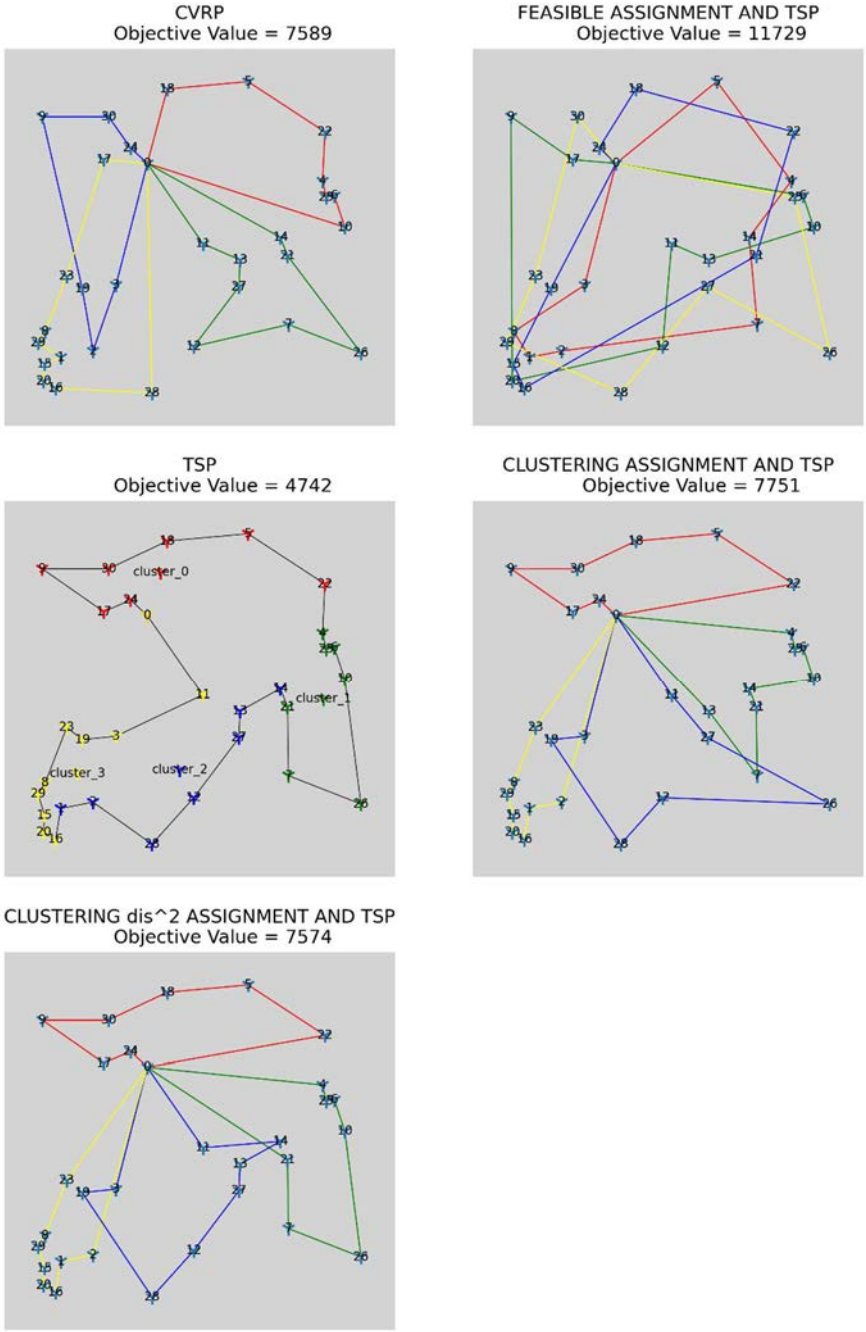
	Λύση TSP
Αντικειμενική τιμή	4742
Δρομολόγιο	0->24->17->9->30->18->5->22->4->25->6->10->26->7->21->14->13->27->12->28->2->1->16->20->15->29->8->23->19->3->11->0

Πινακας 4.2.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 2

	Λύση CL	Λύση CL2
Αντικειμενική τιμή	7751	7574
Δρομολόγιο Οχήματος 1	0->22->5->18->30->9->17->24->0	0->22->5->18->30->9->17->24->0
Απόσταση Δρομολογίου 1	1830	1830
Φορτίο Δρομολογίου 1	46	46
Δρομολόγιο Οχήματος 2	0->4->25->6->10->14->21->7->13->0	0->4->25->6->10->26->7->21->0
Απόσταση Δρομολογίου 2	1834	2058
Φορτίο Δρομολογίου 2	46	46

Δρομολόγιο Οχήματος 3	0->3->19->28->12->26->27->11->0	0->11->14->13->27->12->28->19->3->0
Απόσταση Δρομολογίου 3	2461	2060
Φορτίο Δρομολογίου 3	46	46
Δρομολόγιο Οχήματος 4	0->23->8->29->15->20->16->1->2->0	0->23->8->29->15->20->16->1->2->0
Απόσταση Δρομολογίου 4	1626	1626
Φορτίο Δρομολογίου 4	46	46

Πινακας 4.2.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 2



Εικόνα 4.2 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 2

Πρόβλημα 3

- Ομομορφο Μη-Πλεονασματικο. Επιλυση με Guided Local Search

Δεδομενα

Αριθμός Τοποθεσιών	37
Αριθμός Πελατών	36
Αριθμός Οχημάτων	3
Σύνολο Ζητήσεων	192
Σύνολο Χωρητικοτήτων	192

Πινακας 4.3.1 Πίνακας δεδομένων Προβλήματος 3

Όχημα	1	2	3
Χωρητικότητα Οχήματος	64	64	64

Πινακας 4.3.2 Πίνακας χωρητικοτήτων Προβλήματος 3

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	511	610	11	1	744	1000	21	8	756	931	31	2	66	713
2	7	777	417	12	1	337	89	22	9	490	98	32	4	444	2
3	3	315	370	13	7	172	874	23	7	767	673	33	9	85	444
4	4	649	94	14	6	73	674	24	1	491	844	34	5	218	104
5	6	603	637	15	4	505	366	25	1	586	662	35	8	807	471
6	10	784	645	16	4	972	737	26	3	12	689	36	2	208	903
7	8	926	483	17	8	290	358	27	6	835	371	37	1	290	748
8	8	998	754	18	10	893	18	28	9	454	713				
9	1	136	971	19	2	422	226	29	6	510	766				
10	9	715	706	20	4	964	760	30	8	634	292				

Πινακας 4.3.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 3

Αποτελεσματα

	Λύση OR	Λυση FTSP
Αντικειμενική τιμή	6197	9813
Δρομολόγιο Οχήματος 1	0->4->9->22->5->15->7->19->20->10->23->28-> 0	0->4->5->9->10->7->6->1->3->11->2->13->8->0
Απόσταση Δρομολογίου 1	1470	3616
Φορτίο Δρομολογίου 1	64	64
Δρομολόγιο Οχήματος 2	0->29->21->31->3->17->26->6->34->1-> 0	0->14->16->18->21->17->22->15->19->20->23->12->0
Απόσταση Δρομολογίου 2	2183	3406
Φορτίο Δρομολογίου 2	64	64
Δρομολόγιο Οχήματος 3	0->24->27->36->35->8->12->30->25->13->32->16->2->33->11->18->14-> 0	0->24->34->26->29->31->33->32->25->30->35->36->27->28->0
Απόσταση Δρομολογίου 3	2544	2791
Φορτίο Δρομολογίου 3	64	64

Πινακας 4.3.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 3

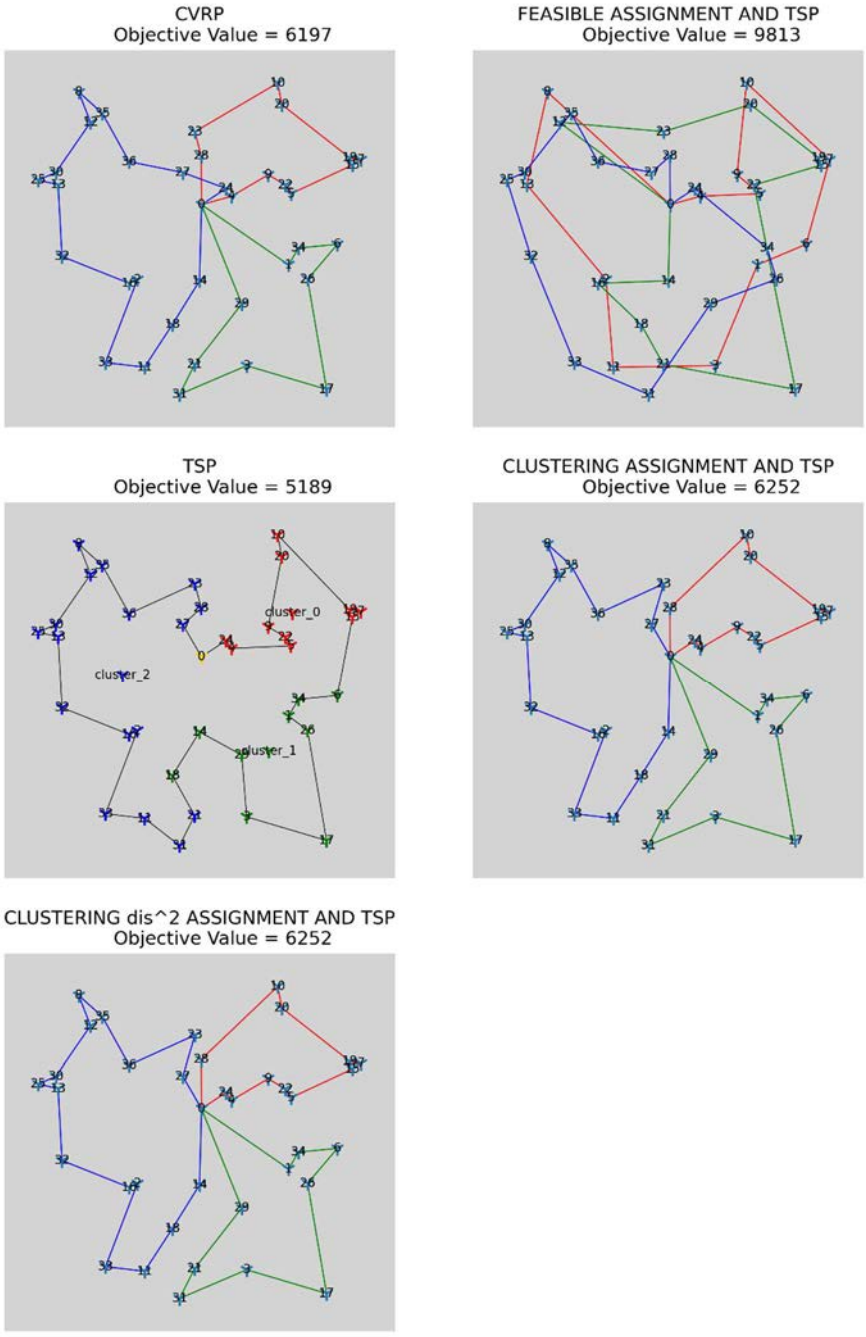
	Λύση TSP
Αντικειμενική τιμή	5189
Δρομολόγιο	0->24->4->5->22->9->20->10->19->7->15->6->34->1->26->17->3->29->14->18->21->31->11->33->2->16->32->13->25->30->12->8->35->36->23->28->27->0

Πινακας 4.3.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 3

	Λύση CL	Λυση CL2
Αντικειμενική τιμή	6252	6252
Δρομολόγιο Οχήματος 1	0->24->4->9->22->5->15->7->19->20->10->28->0	0->24->4->9->22->5->15->7->19->20->10->28->0
Απόσταση Δρομολογίου 1	1449	1449
Φορτίο Δρομολογίου 1	64	64
Δρομολόγιο Οχήματος 2	0->29->21->31->3->17->26->6->34->1->0	0->29->21->31->3->17->26->6->34->1->0
Απόσταση Δρομολογίου 2	2183	2183
Φορτίο Δρομολογίου 2	64	64
Δρομολόγιο Οχήματος 3	0->27->23->36->35->8->12->30->25->13->32->16->2->33->11->18->14->0	0->27->23->36->35->8->12->30->25->13->32->16->2->33->11->18->14->0
Απόσταση Δρομολογίου 3	2620	2620

Φορτίο Δρομολογίου 3	64	64

Πινακας 4.3.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 3



Εικόνα 4.3 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 3

Πρόβλημα 4

- Ομοιόμορφο Πλεονασματικό. Επίλυση με Guided Local Search

Δεδομένα

Αριθμός Τοποθεσιών	30
Αριθμός Πελατών	29
Αριθμός Οχημάτων	4
Σύνολο Ζητήσεων	141
Σύνολο Χωρητικοτήτων	200

Πίνακας 4.4.1 Πίνακας δεδομένων Προβλήματος 4

Όχημα	1	2	3	4
Χωρητικότητα Οχήματος	50	50	50	50

Πίνακας 4.4.2 Πίνακας χωρητικοτήτων Προβλήματος 4

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	693	848	11	4	864	933	21	7	136	478
2	6	103	430	12	3	217	788	22	4	21	318
3	6	288	942	13	1	490	810	23	10	277	75
4	2	910	552	14	5	272	148	24	9	856	258
5	6	68	257	15	7	619	976	25	9	822	107
6	3	599	725	16	8	278	512	26	6	886	997
7	4	860	923	17	2	258	105	27	10	182	436
8	2	145	291	18	4	886	940	28	2	436	740
9	2	259	402	19	3	334	602	29	8	157	876
10	3	288	86	20	2	32	848	30	3	551	90

Πίνακας 4.4.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 4

Αποτελέσματα

	Λύση OR	Λύση FTSP
Αντικειμενική τιμή	5976	9325
Δρομολόγιο Οχήματος 1	0->8->7->4->21->1->20->26->15->18->27-> 0	0->10->6->3->9->16->13->7->4->1->8->11->2->12->5->0
Απόσταση Δρομολογίου 1	1880	3136
Φορτίο Δρομολογίου 1	50	49
Δρομολόγιο Οχήματος 2	0->3->23->24->29->9->22->16->13->5-> 0	0->27->18->15->29->22->21->20->19->14->17->0
Απόσταση Δρομολογίου 2	2271	3352
Φορτίο Δρομολογίου 2	46	50
Δρομολόγιο Οχήματος 3	0-> 0	0->25->23->24->26->28->0
Απόσταση Δρομολογίου 3	0	2837
Φορτίο Δρομολογίου 3	0	42
Δρομολόγιο Οχήματος 4	0->12->11->19->28->2->14->25->17->10->6-> 0	0->0
Απόσταση Δρομολογίου 4	1825	0
Φορτίο Δρομολογίου 4	45	0

Πίνακας 4.4.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 4

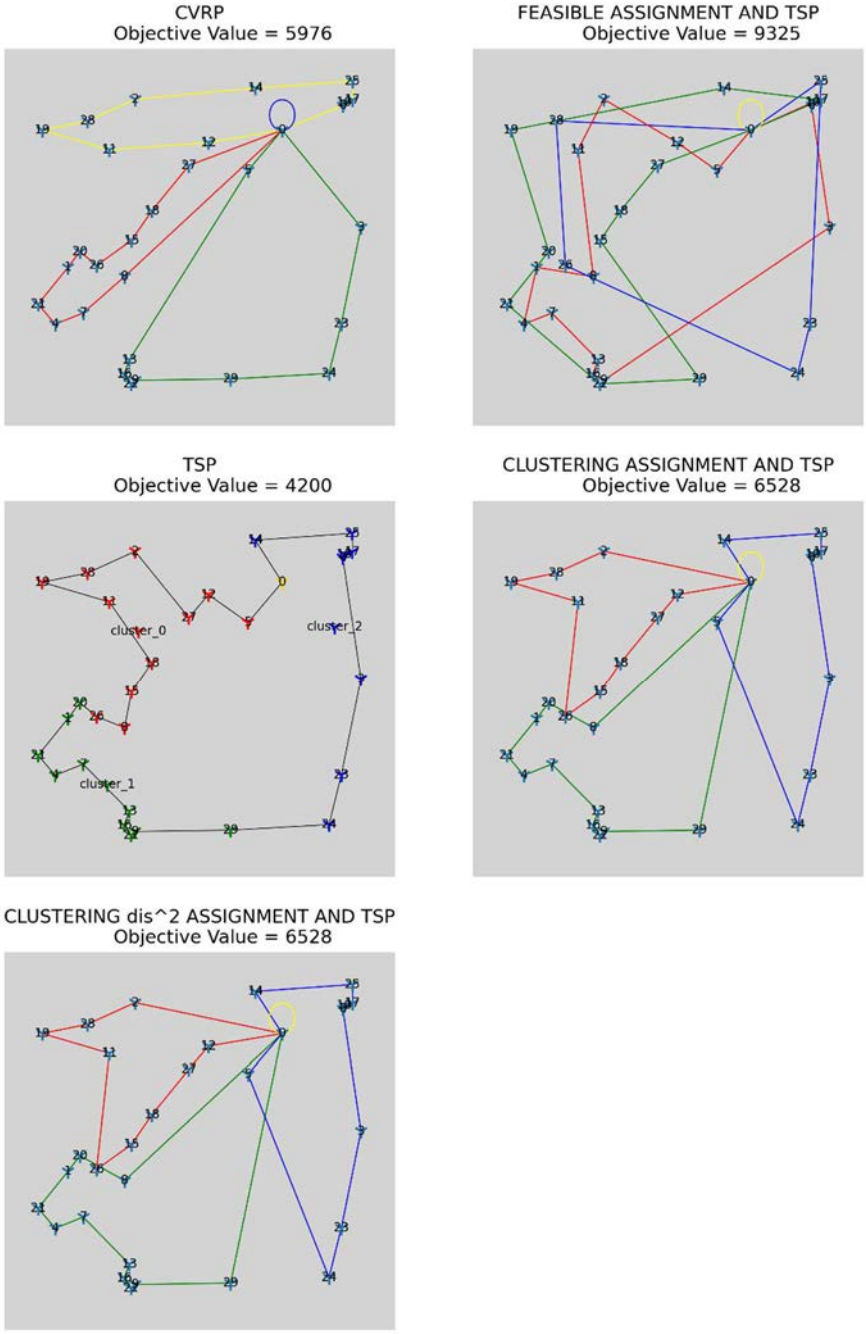
	Λύση TSP
Αντικειμενική τιμή	4200
Δρομολόγιο	0->5->12->27->2->28->19->11->18->15->8->26->20->1->21->4->7->13->16->22->9->29->24->23->3->6->10->17->25->14->0

Πίνακας 4.4.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 4

	Λύση CL	Λύση CL2
Αντικειμενική τιμή	6528	6528
Δρομολόγιο Οχήματος 1	0->12->27->18->15->26->11->19->28->2->0	0->12->27->18->15->26->11->19->28->2->0
Απόσταση Δρομολογίου 1	1934	1934
Φορτίο Δρομολογίου 1	43	43
Δρομολόγιο Οχήματος 2	0->8->20->1->21->4->7->13->16->22->9->29->0	0->8->20->1->21->4->7->13->16->22->9->29->0
Απόσταση Δρομολογίου 2	2447	2447
Φορτίο Δρομολογίου 2	50	50

Δρομολόγιο Οχήματος 3	0->14->25->17->10->6->3->23->24->5->0	0->14->25->17->10->6->3->23->24->5->0
Απόσταση Δρομολογίου 3	2147	2147
Φορτίο Δρομολογίου 3	48	48
Δρομολόγιο Οχήματος 4	0->0	0->0
Απόσταση Δρομολογίου 4	0	0
Φορτίο Δρομολογίου 4	0	0

Πίνακας 4.4.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 4



Εικόνα 4.4 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 4

Πρόβλημα 5

- Ομοιόμορφο Μη-Πλεονασματικό. Επίλυση με Tabu Search

Δεδομενα

Αριθμός Τοποθεσιών	39
Αριθμός Πελατών	38
Αριθμός Οχημάτων	4
Σύνολο Ζητήσεων	228
Σύνολο Χωρητικοτήτων	228

Πινακας 4.5.1 Πίνακας δεδομένων Προβλήματος 5

Όχημα	1	2	3	4
Χωρητικότητα Οχήματος	57	57	57	57

Πινακας 4.5.2 Πίνακας χωρητικοτήτων Προβλήματος 5

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	289	73	11	8	205	165	21	4	611	807	31	3	494	259
2	1	562	580	12	6	404	25	22	2	196	34	32	3	437	961
3	7	117	125	13	9	6	877	23	8	775	222	33	2	770	696
4	5	219	676	14	5	798	173	24	10	874	499	34	10	757	642
5	9	515	126	15	7	373	765	25	6	913	729	35	10	425	870
6	9	733	497	16	4	604	384	26	6	65	931	36	2	88	404
7	4	136	243	17	2	170	697	27	9	428	553	37	7	665	15
8	10	131	882	18	4	817	992	28	4	755	699	38	8	91	172
9	1	591	322	19	6	536	529	29	3	663	886	39	10	227	332
10	10	594	647	20	10	323	623	30	4	521	29				

Πινακας 4.5.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 5

Αποτελεσματα

	Λυση CVRP	Λυση FTSP
Αντικειμενική τιμή	8341	11331
Δρομολόγιο Οχήματος 1	0->21->2->37->6->38->4->36->29->11-> 0	0->10->35->3->14->31->32->27->9->15->22->29->0
Απόσταση Δρομολογίου 1	1415	2741
Φορτίο Δρομολογίου 1	57	57
Δρομολόγιο Οχήματος 2	0->10->26->14->20->28->17->9->1->18->15->8-> 0	0->8->18->1->5->23->24->17->20->25->19->0
Απόσταση Δρομολογίου 2	2472	3296
Φορτίο Δρομολογίου 2	57	57
Δρομολόγιο Οχήματος 3	0->30->5->33->27->32->24->23->22->13-> 0	0->21->37->38->16->34->28->33->26->30->0
Απόσταση Δρομολογίου 3	2080	2580
Φορτίο Δρομολογίου 3	57	57
Δρομολόγιο Οχήματος 4	0->35->3->16->12->25->7->31->34->19-> 0	0->11->4->36->13->7->12->6->2->0
Απόσταση Δρομολογίου 4	2374	2714
Φορτίο Δρομολογίου 4	57	57

Πινακας 4.5.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 5

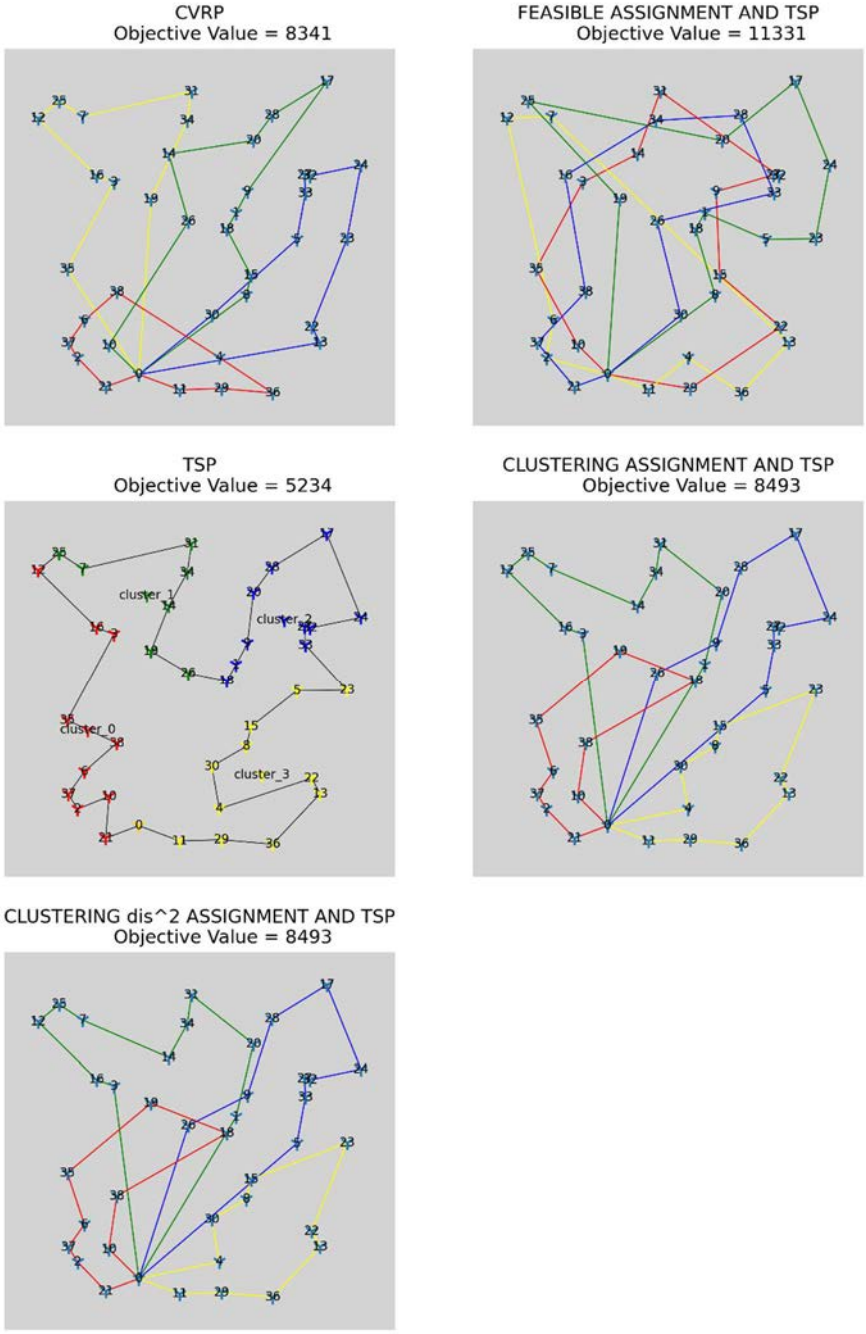
	Λύση TSP
Αντικειμενική τιμή	5234
Δρομολόγιο	0->21->10->2->37->6->38->35->3->16->12->25->7->31->34->14->19->26->18->1->9->20->28->17->24->32->27->33->23->5->15->8->30->4->22->13->36->29->11->0

Πινακας 4.5.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 5

	Λύση CL	Λυση CL2
Αντικειμενική τιμή	8493	8493
Δρομολόγιο Οχήματος 1	0->21->2->37->6->35->19->18->38->10->0	0->21->2->37->6->35->19->18->38->10->0
Απόσταση Δρομολογίου 1	1741	1741
Φορτίο Δρομολογίου 1	57	57
Δρομολόγιο Οχήματος 2	0->1->20->31->34->14->7->25->12->16->3->0	0->1->20->31->34->14->7->25->12->16->3->0
Απόσταση Δρομολογίου 2	2584	2584
Φορτίο Δρομολογίου 2	57	57

Δρομολόγιο Οχήματος 3	0->26->9->28->17->24->32->27->33->5->0	0->26->9->28->17->24->32->27->33->5->0
Απόσταση Δρομολογίου 3	2387	2387
Φορτίο Δρομολογίου 3	57	57
Δρομολόγιο Οχήματος 4	0->11->29->36->13->22->23->15->8->30->4->0	0->11->29->36->13->22->23->15->8->30->4->0
Απόσταση Δρομολογίου 4	1781	1781
Φορτίο Δρομολογίου 4	57	57

Πινακας 4.5.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 5



Εικόνα 4.5 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 5

Πρόβλημα 6

- Ανομοιομορφο Μη-Πλεονασματικο. Επιλυση με Tabu Search.

Δεδομενα

Αριθμός Τοποθεσιών	37
Αριθμός Πελατών	36
Αριθμός Οχημάτων	3
Σύνολο Ζητήσεων	155
Σύνολο Χωρητικοτήτων	155

Πινακας 4.6.1 Πίνακας δεδομένων Προβλήματος 6

Όχημα	1	2	3
Χωρητικότητα Οχήματος	91	33	31

Πινακας 4.6.2 Πίνακας χωρητικοτήτων Προβλήματος 6

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	832	645	11	4	61	884	21	9	453	303	31	2	835	73
2	8	78	548	12	3	250	103	22	1	845	110	32	5	698	721
3	1	938	195	13	3	904	213	23	3	317	819	33	8	715	600
4	4	294	56	14	3	978	461	24	3	584	372	34	3	949	799
5	5	144	562	15	8	454	292	25	3	431	913	35	6	485	613
6	7	80	463	16	6	587	263	26	4	466	133	36	10	306	670
7	2	36	185	17	9	505	920	27	3	23	514	37	1	76	522
8	5	408	598	18	1	825	113	28	3	646	83				
9	4	812	445	19	1	352	202	29	2	482	218				
10	10	141	923	20	3	787	77	30	2	397	308				

Πινακας 4.6.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 6

Αποτελεσματα

	Λύση OR	Λυση FTSP
Αντικειμενική τιμή	6231	8753
Δρομολόγιο Οχήματος 1	0->23->20->14->29->18->3->11->6->5->26->36->1->4->10->9->35->7->34-> 0	0->8->13->12->2->21->17->19->15->14->18->3->11->6->5->1->4->10->9->22->16->7->0
Απόσταση Δρομολογίου 1	2931	3893
Φορτίο Δρομολογίου 1	91	91
Δρομολόγιο Οχήματος 2	0->8->15->28->25->27->19->30->17->21->2->12->13-> 0	0->33->24->36->26->29->20->28->25->27->23->0
Απόσταση Δρομολογίου 2	1785	2816
Φορτίο Δρομολογίου 2	33	33
Δρομολόγιο Οχήματος 3	0->32->31->22->24->16->33-> 0	0->32->31->34->35->30->0
Απόσταση Δρομολογίου 3	1515	2044
Φορτίο Δρομολογίου 3	31	31

Πινακας 4.6.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 6

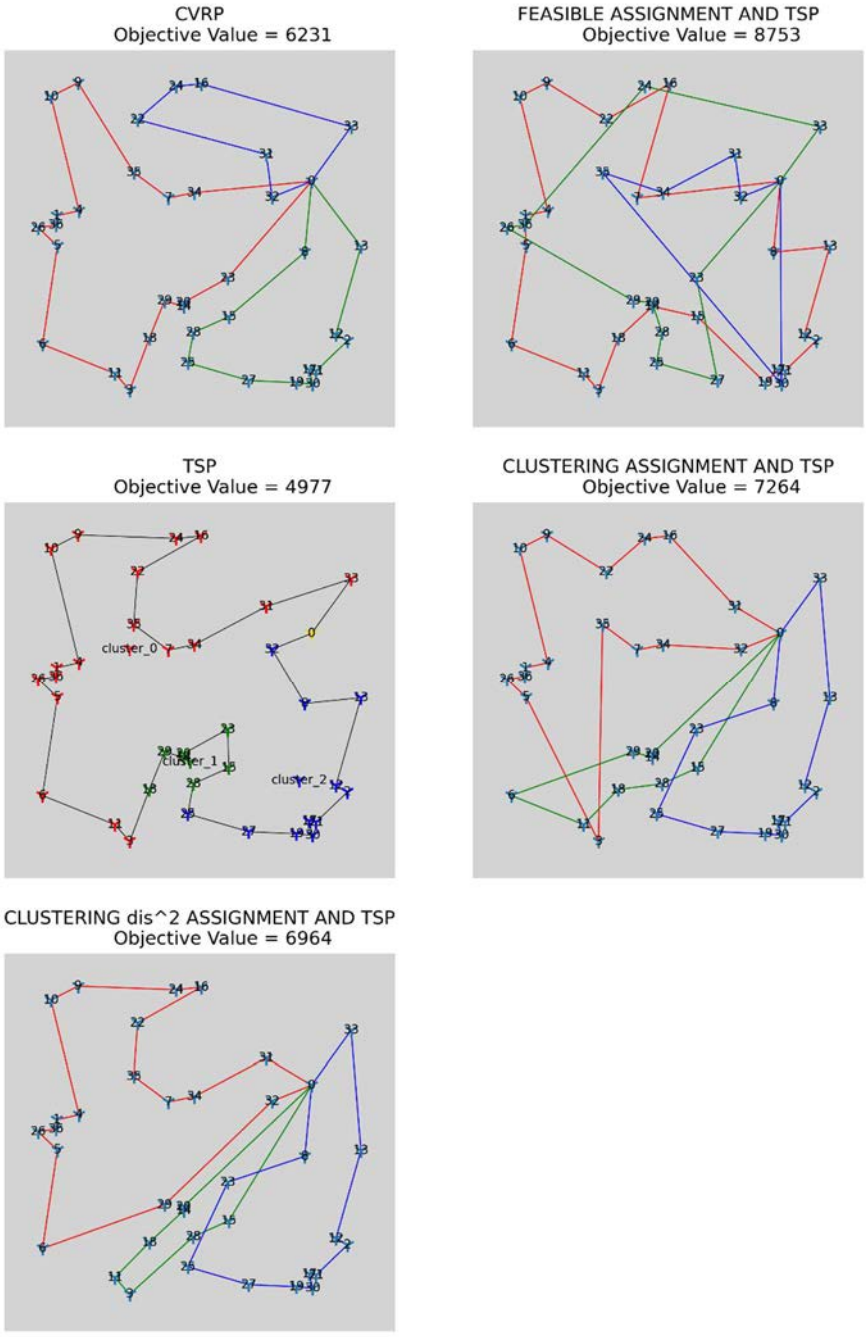
	Λύση TSP
Αντικειμενική τιμή	4977
Δρομολόγιο	0->33->31->34->7->35->22->16->24->9->10->4->1->36->26->5->6->11->3->18->29->14->20->23->15->28->25->27->19->30->17->21->2->12->13->8->32->0

Πινακας 4.6.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 6

	Λύση CL	Λυση CL2
Αντικειμενική τιμή	7264	6964
Δρομολόγιο Οχήματος 1	0->32->34->7->35->3->5->26->36->1->4->10->9->22->24->16->31->0	0->32->29->6->5->26->36->1->4->10->9->24->16->22->35->7->34->31->0
Απόσταση Δρομολογίου 1	3134	3186
Φορτίο Δρομολογίου 1	91	91
Δρομολόγιο Οχήματος 2	0->15->28->18->11->6->29->14->20->0	0->15->28->3->11->18->14->20->0
Απόσταση Δρομολογίου 2	2031	1679
Φορτίο Δρομολογίου 2	33	33
Δρομολόγιο Οχήματος 3	0->33->13->12->2->21->17->30->19->27->25->23->8->0	0->33->13->12->2->21->17->30->19->27->25->23->8->0
Απόσταση Δρομολογίου 3	2099	2099

Φορτίο Δρομολογίου 3	31	31

Πινακας 4.6.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 6



Εικόνα 4.6 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 6

Πρόβλημα 7

- Ανομοιομορφο Στρογγυλοποιημενο. Επιλυση με Guided Local Search

Δεδιομενα

Αριθμός Τοποθεσιών	30
Αριθμός Πελατών	29
Αριθμός Οχημάτων	3
Σύνολο Ζητήσεων	164
Σύνολο Χωρητικοτήτων	180

Πινακας 4.7.1 Πίνακας δεδομένων Προβλήματος 7

Όχημα	1	2	3
Χωρητικότητα Οχήματος	130	30	20

Πινακας 4.7.2 Πίνακας χωρητικοτήτων Προβλήματος 7

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	580	203	11	9	396	327	21	3	210	321
2	5	802	609	12	6	255	502	22	3	816	800
3	5	972	431	13	8	707	548	23	5	306	751
4	4	536	110	14	9	150	25	24	10	399	909
5	6	662	953	15	7	257	417	25	6	194	10
6	9	609	689	16	4	817	138	26	4	200	511
7	7	789	194	17	1	38	145	27	9	139	47
8	5	901	866	18	6	361	963	28	3	645	987
9	5	326	401	19	7	735	960	29	3	945	72
10	10	307	196	20	2	42	47	30	3	559	214

Πινακας 4.7.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 7

Αποτελεσματα

	Λύση OR	Λυση FTSP
Αντικειμενική τιμή	5035	
Δρομολόγιο Οχήματος 1	0->6->15->28->2->1->12->5->21->7->18->4->27->17->23->22->25->11->14->8->10->9->29-> 0	0->6->15->2->1->12->5->21->7->18->4->17->22->25->11->14->8->10->20->16->19->13->9->3->0
Απόσταση Δρομολογίου 1	3497	4057
Φορτίο Δρομολογίου 1	130	130
Δρομολόγιο Οχήματος 2	0->20->16->19->26->13->24-> 0	0->24->26->23->27->0
Απόσταση Δρομολογίου 2	1332	2443
Φορτίο Δρομολογίου 2	30	28
Δρομολόγιο Οχήματος 3	0->3-> 0	0->29->28->0
Απόσταση Δρομολογίου 3	206	823
Φορτίο Δρομολογίου 3	4	6

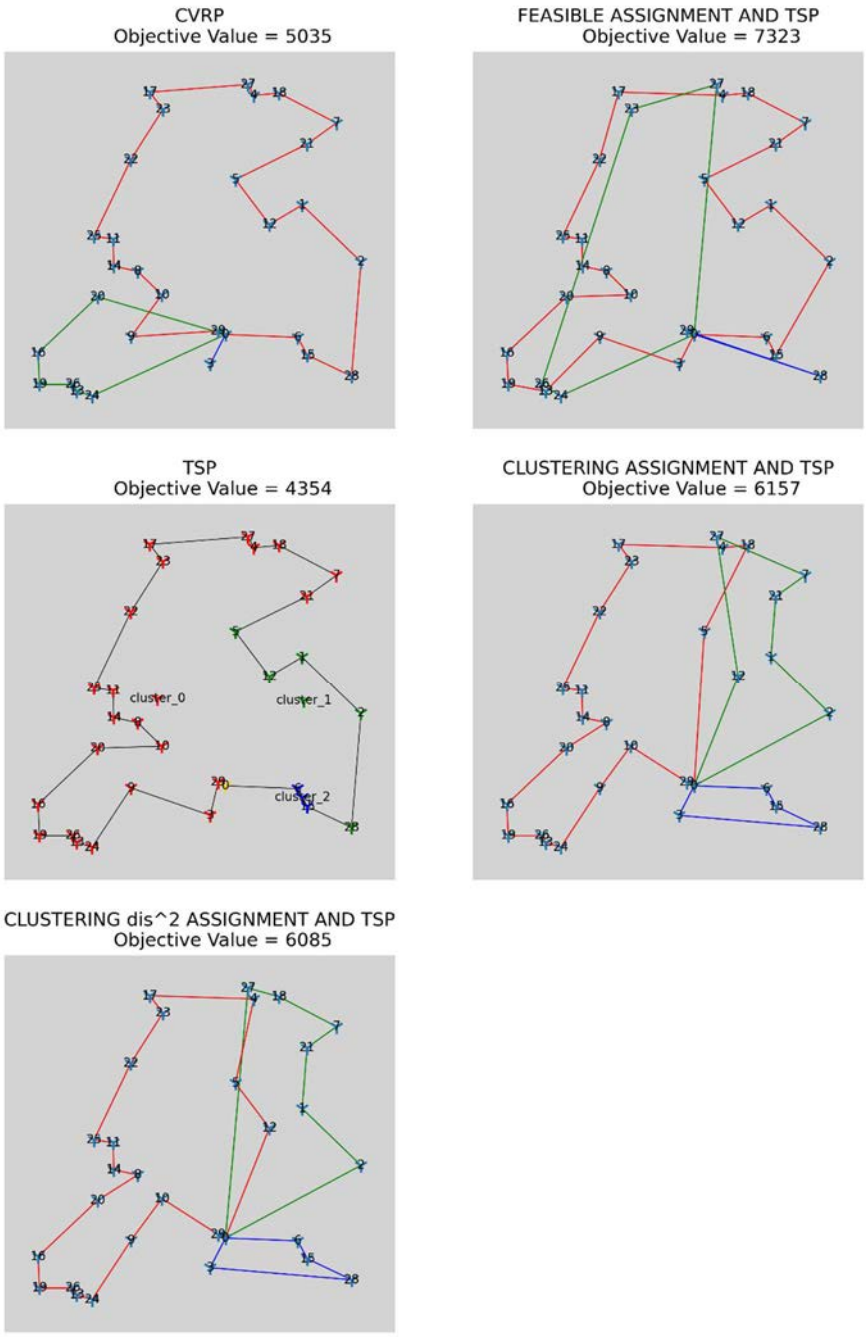
Πινακας 4.7.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 7

	Λύση TSP
Αντικειμενική τιμή	4354
Δρομολόγιο	0->29->3->9->24->13->26->19->16->20->10->8->14->11->25->22->23->17->27->4->18->7->21->5->12->1->2->28->15->6->0

Πινακας 4.7.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 7

	Λύση CL	Λυση CL2
Αντικειμενική τιμή	6157	6085
Δρομολόγιο Οχήματος 1	0->29->10->9->24->13->26->19->16->20->8->14->11->25->22->23->17->4->18->5->0	0->29->10->9->24->13->26->19->16->20->8->14->11->25->22->23->17->4->5->12->0
Απόσταση Δρομολογίου 1	3134	3084
Φορτίο Δρομολογίου 1	117	118
Δρομολόγιο Οχήματος 2	0->12->27->7->21->1->2->0	0->2->1->21->7->18->27->0
Απόσταση Δρομολογίου 2	2093	2071
Φορτίο Δρομολογίου 2	29	28
Δρομολόγιο Οχήματος 3	0->3->28->15->6->0	0->3->28->15->6->0
Απόσταση Δρομολογίου 3	930	930
Φορτίο Δρομολογίου 3	18	18

Πινακας 4.7.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 7



Εικόνα 4.7 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 7

Πρόβλημα 8

- Ανομοιόμορφο Πλεονασματικό. Επίλυση με Guided Local Search

Δεδομένα

Αριθμός Τοποθεσιών	30
Αριθμός Πελατών	29
Αριθμός Οχημάτων	4
Σύνολο Ζητήσεων	145
Σύνολο Χωρητικοτήτων	162

Πίνακας 4.8.1 Πίνακας δεδομένων Προβλήματος 8

Όχημα	1	2	3	4
Χωρητικότητα Οχήματος	89	33	23	17

Πίνακας 4.8.2 Πίνακας χωρητικοτήτων Προβλήματος 8

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	73	378	11	1	767	76	21	9	992	172
2	5	47	451	12	4	740	147	22	5	576	542
3	4	296	214	13	2	66	75	23	1	123	265
4	1	195	521	14	5	187	874	24	3	646	544
5	10	199	197	15	6	727	118	25	2	84	363
6	8	887	454	16	3	210	603	26	4	776	537
7	6	973	294	17	6	340	604	27	8	20	787
8	9	372	413	18	5	193	342	28	7	168	997
9	2	905	415	19	6	856	805	29	6	159	242
10	6	286	446	20	7	211	988	30	4	0	454

Πίνακας 4.8.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 8

Αποτελέσματα

	Λύση OR	Λύση FTSP
Αντικειμενική τιμή	5257	8614
Δρομολόγιο Οχήματος 1	0->1->3->15->16->21->23->18->25->5->8->6->20->10->14->11->7->9->17-> 0	0->28->12->4->2->11->14->10->6->8->5->7->9->3->15->13->27->29->1->0
Απόσταση Δρομολογίου 1	2973	3549
Φορτίο Δρομολογίου 1	89	89
Δρομολόγιο Οχήματος 2	0->24->13->19->27->26->29-> 0	0->17->20->18->19->16->0
Απόσταση Δρομολογίου 2	1396	3014
Φορτίο Δρομολογίου 2	33	33
Δρομολόγιο Οχήματος 3	0->12->4->2->28->22-> 0	0->24->22->25->23->21->26->0
Απόσταση Δρομολογίου 3	888	2051
Φορτίο Δρομολογίου 3	23	23
Δρομολόγιο Οχήματος 4	0-> 0	0->0
Απόσταση Δρομολογίου 4	0	0
Φορτίο Δρομολογίου 4	0	0

Πίνακας 4.8.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 8

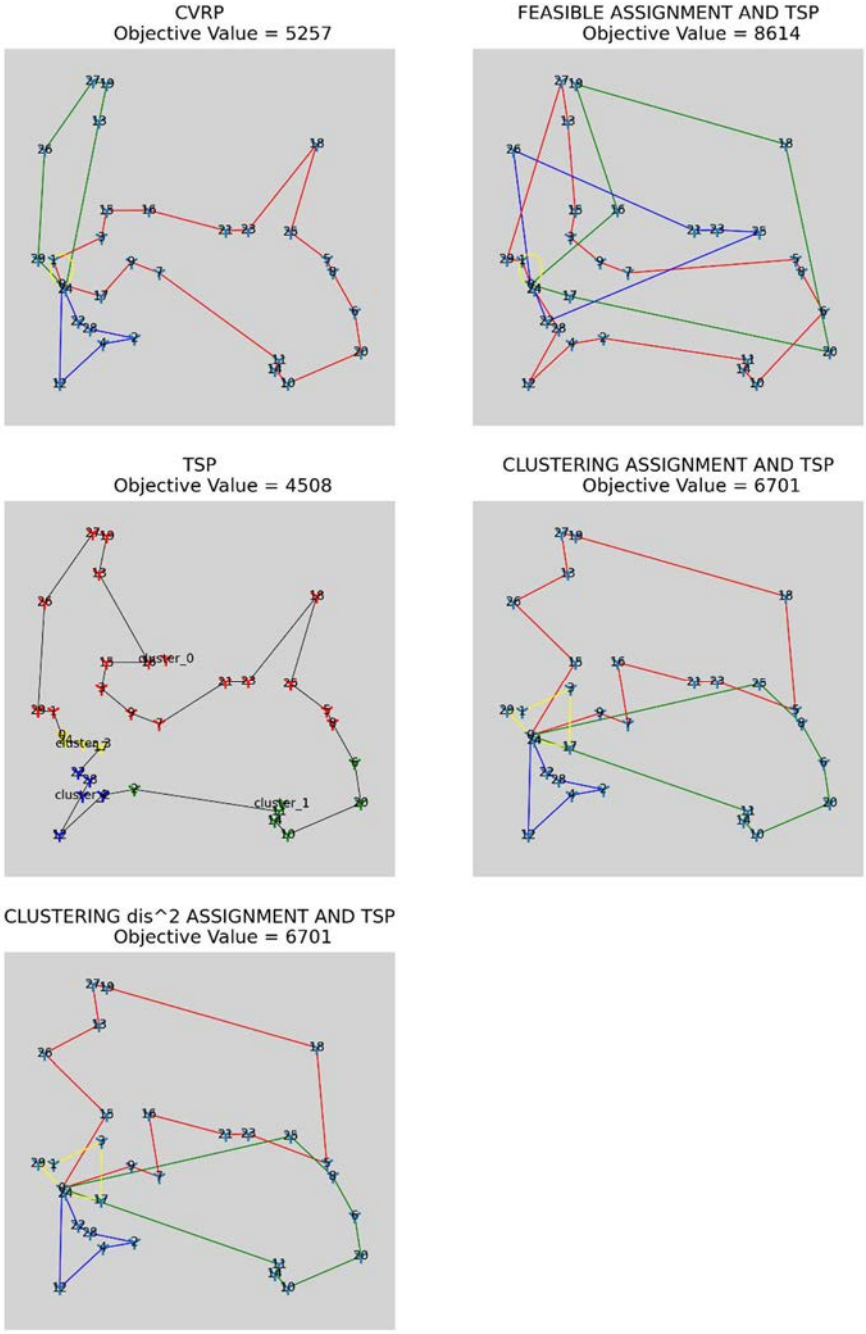
	Λύση TSP
Αντικειμενική τιμή	4508
Δρομολόγιο	0->1->29->26->27->19->13->16->15->3->9->7->21->23->18->25->5->8->6->20->10->14->11->2->4->12->28->22->17->24->0

Πίνακας 4.8.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 8

	Λύση CL	Λύση CL2
Αντικειμενική τιμή	6701	6701
Δρομολόγιο Οχήματος 1	0->9->7->16->21->23->5->18->19->27->13->26->15->0	0->9->7->16->21->23->5->18->19->27->13->26->15->0
Απόσταση Δρομολογίου 1	2986	2986
Φορτίο Δρομολογίου 1	73	73
Δρομολόγιο Οχήματος 2	0->11->14->10->20->6->8->25->0	0->11->14->10->20->6->8->25->0
Απόσταση Δρομολογίου 2	2202	2202
Φορτίο Δρομολογίου 2	32	32

Δρομολόγιο Οχήματος 3	0->22->28->2->4->12->0	0->22->28->2->4->12->0
Απόσταση Δρομολογίου 3	888	888
Φορτίο Δρομολογίου 3	23	23
Δρομολόγιο Οχήματος 4	0->24->17->3->1->29->0	0->24->17->3->1->29->0
Απόσταση Δρομολογίου 4	625	625
Φορτίο Δρομολογίου 4	17	17

Πίνακας 4.8.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 8



Εικόνα 4.8 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 8

Πρόβλημα 9

- Ομοιόμορφο Στρογγυλοποιημένο. Επίλυση με Guided Local Search

Δεδομένα

Αριθμός Τοποθεσιών	32
Αριθμός Πελατών	31
Αριθμός Οχημάτων	3
Σύνολο Ζητήσεων	184
Σύνολο Χωρητικοτήτων	210

Πίνακας 4.9.1 Πίνακας δεδομένων Προβλήματος 9

Όχημα	1	2	3
Χωρητικότητα Οχήματος	70	70	70

Πίνακας 4.9.2 Πίνακας χωρητικοτήτων Προβλήματος 9

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	203	612	11	1	304	112	21	7	312	969	31	4	197	929
2	1	56	866	12	8	141	285	22	1	380	45	32	5	840	618
3	2	43	0	13	10	986	259	23	8	692	510				
4	7	547	153	14	6	739	564	24	3	233	398				
5	10	55	512	15	9	659	691	25	1	390	788				
6	7	242	656	16	7	326	144	26	10	289	778				
7	9	915	261	17	9	84	53	27	2	949	373				
8	4	569	492	18	3	894	970	28	7	958	568				
9	6	526	697	19	5	140	574	29	5	740	870				
10	9	266	919	20	9	286	622	30	9	750	199				

Πίνακας 4.9.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 9

Αποτελέσματα

Λύση OR		
Αντικειμενική τιμή	6040	
Δρομολόγιο Οχήματος 1	0->26->12->6->29->3->21->15->10->2->16->11->23-> 0	0->5->1->9->8->7->13->6->3->10->2->11->4->0
Απόσταση Δρομολογίου 1	2661	3280
Φορτίο Δρομολογίου 1	68	70
Δρομολόγιο Οχήματος 2	0->18->4->1->30->9->20->25-> 0	0->18->16->15->21->12->22->14->17->20->24->19->0
Απόσταση Δρομολογίου 2	1204	3608
Φορτίο Δρομολογίου 2	46	69
Δρομολόγιο Οχήματος 3	0->19->7->22->13->31->27->17->28->14->8->24->5-> 0	0->25->30->28->31->27->26->29->23->0
Απόσταση Δρομολογίου 3	2175	2538
Φορτίο Δρομολογίου 3	70	45

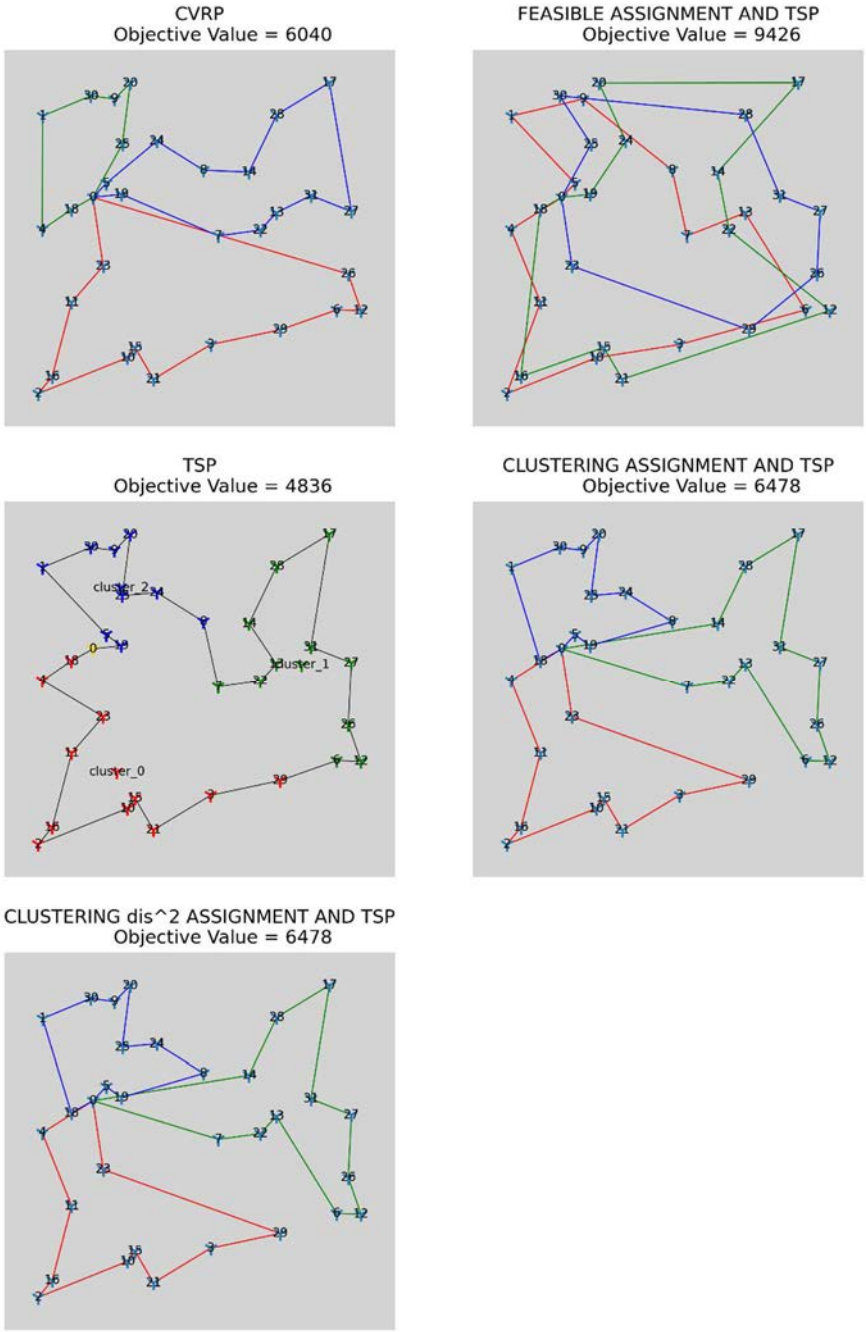
Πίνακας 4.9.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 9

	Λύση TSP
Αντικειμενική τιμή	4836
Δρομολόγιο	0->18->4->23->11->16->2->10->15->21->3->29->6->12->26->27->31->17->28->14->13->22->7->8->24->25->20->9->30->1->5->19->0

Πίνακας 4.9.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 9

	Λύση CL	Λύση CL2
Αντικειμενική τιμή	6478	6478
Δρομολόγιο Οχήματος 1	0->4->11->16->2->10->15->21->3->29->23->0	0->4->11->16->2->10->15->21->3->29->23->0
Απόσταση Δρομολογίου 1	2341	2341
Φορτίο Δρομολογίου 1	57	57
Δρομολόγιο Οχήματος 2	0->7->22->13->6->12->26->27->31->17->28->14->0	0->7->22->13->6->12->26->27->31->17->28->14->0
Απόσταση Δρομολογίου 2	2644	2644
Φορτίο Δρομολογίου 2	68	68
Δρομολόγιο Οχήματος 3	0->5->19->8->24->25->20->9->30->1->18->0	0->5->19->8->24->25->20->9->30->1->18->0
Απόσταση Δρομολογίου 3	1493	1493
Φορτίο Δρομολογίου 3	59	59

Πίνακας 4.9.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 9



Εικόνα 4.9 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 9

Πρόβλημα 10

- Ανομοιόμορφο Πλεονασματικό. Επίλυση με Tabu Search

Δεδομένα

Αριθμός Τοποθεσιών	38
Αριθμός Πελατών	37
Αριθμός Οχημάτων	4
Σύνολο Ζητήσεων	191
Σύνολο Χωρητικοτήτων	236

Πίνακας 4.10.1 Πίνακας δεδομένων Προβλήματος 10

Όχημα	1	2	3	4
Χωρητικότητα Οχήματος	83	60	48	45

Πίνακας 4.10.2 Πίνακας χωρητικοτήτων Προβλήματος 10

Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y	Τοποθεσία	Ζήτηση	X	Y
1	0	188	322	11	9	17	647	21	6	329	962	31	3	591	980
2	4	78	906	12	7	514	698	22	2	596	320	32	3	726	22
3	10	465	880	13	7	41	78	23	5	849	709	33	5	9	327
4	6	191	837	14	2	356	418	24	9	484	704	34	8	442	942
5	4	550	701	15	1	911	965	25	3	65	327	35	5	579	347
6	3	350	697	16	9	315	364	26	8	320	291	36	2	373	811
7	6	633	930	17	1	157	320	27	7	854	516	37	2	272	979
8	8	106	698	18	7	525	259	28	8	832	84	38	9	516	888
9	4	523	877	19	2	978	313	29	3	803	221				
10	1	84	153	20	10	541	798	30	2	753	727				

Πίνακας 4.10.3 Πίνακας τοποθεσιών και ζητήσεων Προβλήματος 10

Αποτελέσματα

	Λύση OR	Λύση FTSP
Αντικειμενική τιμή	6513	10409
Δρομολόγιο Οχήματος 1	0->31->27->28->18->26->22->29->14->6->30->33->20->36->1->3->7->10-> 0	0->16->9->12->10->7->1->3->2->8->6->14->4->11->5->13->15->0
Απόσταση Δρομολογίου 1	3537	3179
Φορτίο Δρομολογίου 1	83	82
Δρομολόγιο Οχήματος 2	0->5->35->2->37->8->19->4->11->23->13-> 0	0->24->20->23->19->22->26->18->21->17->25->0
Απόσταση Δρομολογίου 2	1455	2793
Φορτίο Δρομολογίου 2	60	59
Δρομολόγιο Οχήματος 3	0->16->24->32->12->9->25->17->21->34->15-> 0	0->32->35->36->33->30->29->28->27->31->34->0
Απόσταση Δρομολογίου 3	1521	3129
Φορτίο Δρομολογίου 3	48	41
Δρομολόγιο Οχήματος 4	0-> 0	0->37->0
Απόσταση Δρομολογίου 4	0	1308
Φορτίο Δρομολογίου 4	0	9

Πίνακας 4.10.4 Πίνακας αποτελεσμάτων OR και FTSP Προβλήματος 10

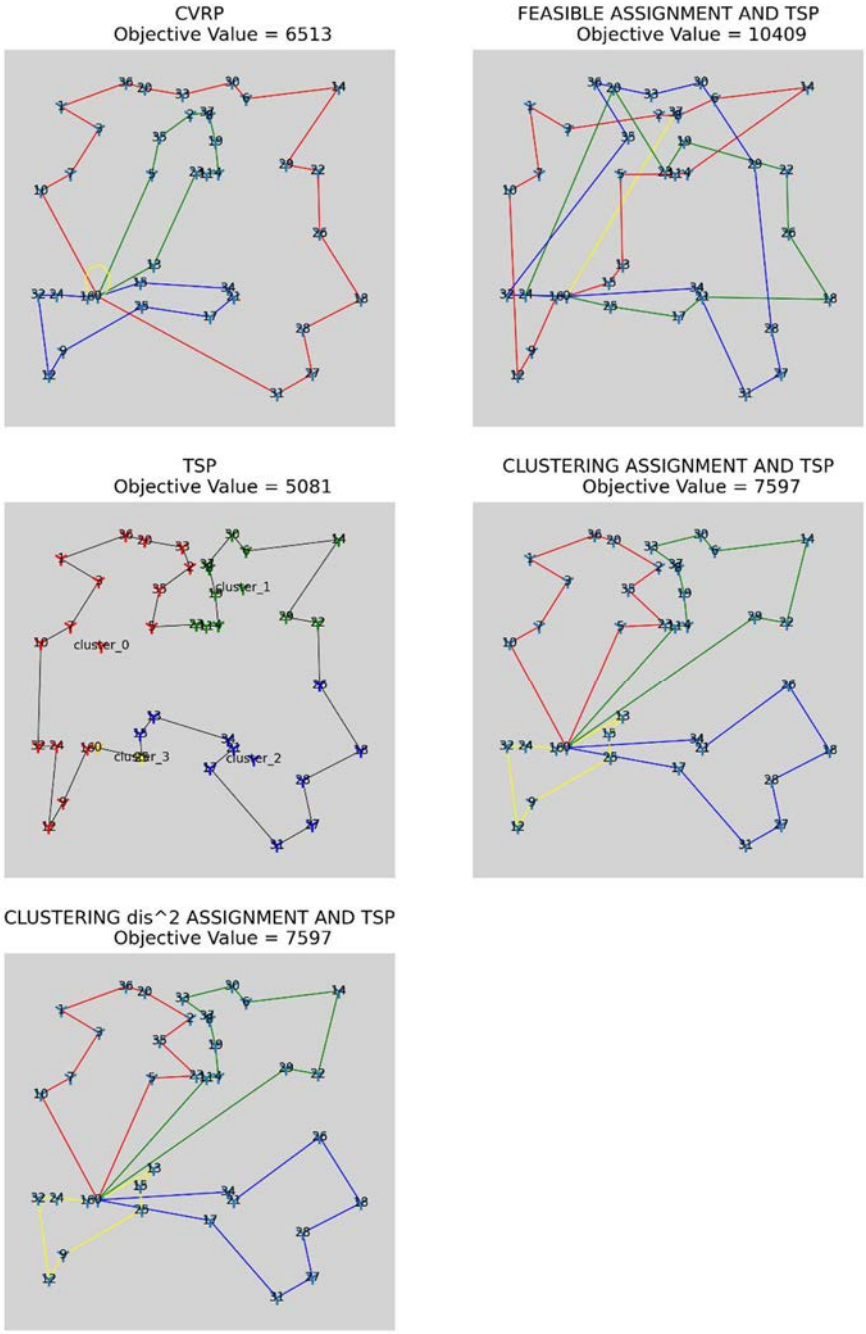
	Λύση TSP
Αντικειμενική τιμή	5081
Δρομολόγιο	0->16->9->12->24->32->10->7->3->1->36->20->33->2->35->5->23->11->4->19->8->37->30->6->14->29->22->26->18->28->27->31->17->21->34->13->15->25->0

Πίνακας 4.10.5 Πίνακας αποτελεσμάτων TSP Προβλήματος 10

	Λύση CL	Λύση CL2
Αντικειμενική τιμή	7597	7597
Δρομολόγιο Οχήματος 1	0->10->7->3->1->36->20->2->35->23->5->0	0->10->7->3->1->36->20->2->35->23->5->0
Απόσταση Δρομολογίου 1	2001	2001
Φορτίο Δρομολογίου 1	59	59
Δρομολόγιο Οχήματος 2	0->11->4->19->8->37->33->30->6->14->22->29->0	0->11->4->19->8->37->33->30->6->14->22->29->0
Απόσταση Δρομολογίου 2	2372	2372

Φορτίο Δρομολογίου 2	59	59
Δρομολόγιο Οχήματος 3	0->17->31->27->28->18->26->21->34->0	0->17->31->27->28->18->26->21->34->0
Απόσταση Δρομολογίου 3	2101	2101
Φορτίο Δρομολογίου 3	37	37
Δρομολόγιο Οχήματος 4	0->16->24->32->12->9->25->15->13->0	0->16->24->32->12->9->25->15->13->0
Απόσταση Δρομολογίου 4	1123	1123
Φορτίο Δρομολογίου 4	36	36

Πίνακας 4.10.6 Πίνακας αποτελεσμάτων CL και CL2 Προβλήματος 10



Εικόνα 4.10 Σχηματική αναπαράσταση αποτελεσμάτων Προβλήματος 10

Συμπεράσματα.

Μελετώντας τα αποτελέσματα, παρατηρείται ότι η μέθοδος OR καταφέρνει να παράγει πάντα το χαμηλότερο, κόστος εκτός από την περίπτωση του προβλήματος 2 οπού εκεί η μέθοδος CL2 εντοπίζει ελαφρά καλύτερη λύση. Αυτό επιβεβαιώνει την αποδοτικότητά της συγκριτικά με τις υπόλοιπες μεθόδους, και ιδίως με την FTSP, η οποία επιστρέφει σταθερά το υψηλότερο κόστος από τις τέσσερις. Η μέση τιμή κόστους της OR είναι σημαντικά χαμηλότερη, γεγονός που υποδηλώνει ότι η μέθοδος είναι σταθερή και αξιόπιστη.

Συγκεκριμένα, οι διαφορές κόστους μεταξύ OR και FTSP κυμαίνονται από 35% έως και 63.86%, ενώ η μέση τιμή είναι 52.8% υψηλότερη, επιβεβαιώνοντας ότι η FTSP είναι γενικά λιγότερο αποδοτική. Αυτό ήταν αναμενομένων δεδομένου ότι εμφανίζει αδυναμία στο σκέλος της ανάθεσης, όπως αναφέρθηκε σε προηγούμενα κεφάλαια. Αντίστοιχα η σύγκριση της μεθόδου FTSP με τις CL και CL2 επιστέφει παρόμοιες διάφορες. Η διαφορά κόστους για τη μέθοδο CL κυμαίνεται από 18.94% έως 56.96% με μέση τιμή 37.63%, ενώ η διαφορά κόστους της μεθόδου CL2 κυμαίνεται από 20.35% έως 56.96% με μέση τιμή 38.64%, γεγονός που φανερώνει την αδυναμία της FTSP να διατηρήσει χαμηλό κόστος σε σταθερή βάση.

Οι μέθοδοι CL και CL2, αντίθετα, παρουσιάζουν πολύ μικρότερες αποκλίσεις σε σχέση με την OR.Η μέση αύξηση κόστους ανέρχεται στο 11.59% για την μέθοδο CL και 10.73% για την CL2, ενώ οι τιμές ξεκινούν από 0.89% έως το 27.47% για τη CΛ και -99.8% έως 27.47% για τη CL2. Αξίζει να σημειωθεί ότι σε μια περίπτωση η CL2 έδωσε χαμηλότερο κόστος από την OR. Αυτές οι τιμές επιβεβαιώνουν ότι τόσο η CL όσο και η CL2 είναι αξιόπιστες εναλλακτικές λύσεις και παρουσιάζουν κόστος που προσεγγίζει την απόδοση της OR, ιδιαίτερα στις περιπτώσεις όπου το κόστος είναι η κύρια προτεραιότητα.

Η σύγκριση μεταξύ των μεθόδων CL και CL2 δείχνει μια σταθερότητα, καθώς οι διαφορές κόστους μεταξύ τους είναι από μικρές έως ανύπαρκτες. Συγκεκριμένα οι μόνες περιπτώσεις που αποδίδουν διαφορετικές λύσεις είναι στα προβλήματα 2,6 και 7 όπου οι CL2 δίνει καλυτέρα αποτελέσματα κατά 2.3%, 4.3% και 1.2% αντίστοιχα.

Η μέση διαφορά μεταξύ CL και CL2 είναι σχεδόν αμελητέα, υποδεικνύοντας ότι οι δύο μέθοδοι είναι σχεδόν ισοδύναμες.

Η σύγκριση των μεθόδων CL και CL2 σε προβλήματα με και χωρίς πλεόνασμα χωρητικότητας δείχνει πώς επηρεάζεται η απόδοσή τους ανάλογα με τη διαθεσιμότητα επιπλέον χωρητικότητας. Συνολικά, φαίνεται ότι οι δύο μέθοδοι παρουσιάζουν καλύτερες επιδόσεις σε μη-πλεονασματικά προβλήματα. Αυτό πιθανός, προδίδει την αδυναμία των μεθόδων, να διαχειριστούν με οικονομικό τρόπο την επιπλέον χωρητικότητα τους, με αποτέλεσμα να εκτελούν περιττά δρομολόγια.

- **Προβλήματα χωρίς πλεόνασμα χωρητικότητας (2, 3, 5, 6)**: Σε αυτά τα προβλήματα, όπου οι χωρητικότητες είναι πλήρως κατανεμημένες, οι μέθοδοι CL και CL2 τείνουν να παραμένουν κοντά στην OR, αποφεύγοντας σημαντικές υπερβάσεις κόστους. Για τη μέθοδο CL η μέση τιμή της αύξησης κόστους είναι 5.35% ενώ οι διακυμάνσεις κυμαίνονται από το 0.9% έως το 16.58%. Για τη μέθοδο CL2 η μέση τιμή της αύξησης κόστους είναι 3.57% ενώ οι διακυμάνσεις κυμαίνονται από το -99.8% έως το 11.76%.
- **Προβλήματα με μικρό πλεόνασμα χωρητικότητας (1, 4, 7, 8, 9, 10):** Σε αυτά τα προβλήματα οι μέθοδοι CL και CL2 εμφανίζουν μεγαλύτερη απόκλιση από την OR. . Για τη μεθοδο CL η μέση τιμή της άυξησης κόστους είναι 15.74% ενώ οι αποκλίσεις κυμαίνονται από το 7.25% εως το 27.47%. Για τη μέθοδο CL2 η μέση τιμή της αύξησης κόστους είναι 15.5% ενώ οι διακυμάνσεις κυμαίνονται από το 7.25% έως το 27.47%.

Αξιοσημείωτο είναι ότι, παρόλο που η OR υπερέχει ως προς το κόστος, οι μέθοδοι CL και CL2 εμφανίζουν ελάχιστες διακυμάνσεις και παραμένουν αρκετά κοντά στην OR, γεγονός που τις καθιστά αξιόλογες εναλλακτικές. Αντίθετα, η FTSP αποδεικνύεται πιο ασταθής και κατάλληλη μόνο σε περιπτώσεις όπου το κόστος δεν είναι κρίσιμο.

Συνολικά, η OR αναδεικνύεται η πιο αποδοτική και αξιόπιστη μέθοδος, με τις CL και CL2 να αποτελούν δυνατές εναλλακτικές, ενώ η FTSP παραμένει η λιγότερο αποδοτική μέθοδος, με υψηλότερο κόστος και μεγαλύτερες διακυμάνσεις.

Πρόβλημα\Μέθοδος	OR	FTSP	CL	CL2
1	6376	10046	7114	7114
2	7589	11729	7751	7574
3	6197	9813	6252	6252
4	5976	9325	6528	6528
5	8341	11331	8493	8493
6	6231	8753	7264	6964
7	5035	7323	6157	6085
8	5257	8614	6701	6701
9	6040	9426	6478	6478
10	6513	10409	7597	7597

Πίνακας 4.11.1 Πίνακας αποτελεσμάτων

	FTSP/OR	CL/OR	CL2/OR	FTSP/CL	FTSP/CL2	CL/CL2
Mean	1.528	1.1159	1.1073	1.3763	1.3864	1.0078
Min	1.3585	1.0089	0.998	1.1894	1.2035	1.0
Max	1.6386	1.2747	1.2747	1.5696	1.5696	1.0431

Πίνακας 4.11.2 Πίνακας μέσων τιμών, μέγιστων και ελάχιστων των λόγων μεταξύ μεθόδων

ΑΝΑΦΟΡΕΣ

[1] G. B. Dantzig and J. H. Ramser, “The Truck Dispatching Problem,” *Source: Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[2] “(PDF) Vehicle routing problem: Models and solutions.” Accessed: Oct. 04, 2024. [Online]. Available: https://www.researchgate.net/publication/313005083_Vehicle_routing_problem_Models_and_solutions

[3] F. Arnold and K. Sörensen, “Knowledge-guided local search for the vehicle routing problem,” *Comput Oper Res*, vol. 105, pp. 32–46, May 2019, doi: 10.1016/J.COR.2019.01.002.

[4] C. Voudouris, “Guided Local Search for Combinatorial Optimisation Problems,” 1997.

[5] “Assignment problem - Wikipedia.” Accessed: Sep. 02, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Assignment_problem

[6] “Capacitated Vehicle Routing Problem (CVRP) Optimization using Google-OR Tools and Python | by Nagarjuna Chidarala | Medium.” Accessed: Aug. 14, 2024. [Online]. Available: https://medium.com/@nag96.chidara/capacitated-vehicle-routing-problem-cvrp-optimization-using-google-or-tools-and-python-7848fb5ffd16

[7] A. Van Breedam, “Comparing descent heuristics and metaheuristics for the vehicle routing problem,” *Comput Oper Res*, vol. 28, no. 4, pp. 289–315, Apr. 2001, doi: 10.1016/S0305-0548(99)00101-X.

[8] M. Perjalanan *et al.*, “VEHICLE ROUTING PROBLEM: MODELS AND SOLUTIONS,” *Journal of Quality Measurement and Analysis JQMA*, vol. 4, no. 1, pp. 205–218, 2008.

[9] “Capacitated Vehicle Routing Problem (CVRP) Optimization using Google-OR Tools and Python | by Nagarjuna Chidarala | Medium.” Accessed: Aug. 13, 2024. [Online]. Available: https://medium.com/@nag96.chidara/capacitated-vehicle-routing-problem-cvrp-optimization-using-google-or-tools-and-python-7848fb5ffd16

[10] P. Munari, T. Dollevoet, and R. Spliet, “A generalized formulation for vehicle routing problems,” 2017.

[11] T. Dollevoet, P. Munari, and R. Spliet, “A p-step Formulation for the Capacitated Vehicle Routing Problem”.

[12] Z. Borčinová, “Two models of the capacitated vehicle routing problem,” *Croatian Operational Research Review*, vol. 463, pp. 463–469, 2017, doi: 10.17535/crorr.2017.0029.

[13] “GitHub - d-krupke/cpsat-primer: The CP-SAT Primer: Using and Understanding Google OR-Tools’ CP-SAT Solver.” Accessed: Aug. 08, 2024. [Online]. Available: https://github.com/d-krupke/cpsat-primer

[14] Anthony Wren & Alan Holliday, “Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points Operational Research Quarterly ,” 1977.

[15] “Christofides algorithm - Wikipedia.” Accessed: Aug. 07, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Christofides_algorithm

[16] G. Clarke and ; J W Wright, “Scheduling of Vehicles from a Central Depot to a Number of Delivery Points,” *Oper Res*, vol. 12, no. 4, pp. 568–581.

[17] “Operations Research and Constraint Programming at Google | SpringerLink.” Accessed: Aug. 06, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-23786-7_2

[18] L. Perron, “Operations Research and Constraint Programming at Google,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6876 LNCS, no. 2, pp. 2–2, 2011, doi: 10.1007/978-3-642-23786-7_2.

[19] “Solving an Assignment Problem | OR-Tools | Google for Developers.” Accessed: Aug. 06, 2024. [Online]. Available: https://developers.google.com/optimization/assignment/assignment_example

ΠΑΡΑΡΤΗΜΑ

```
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
from ortools.linear_solver import pywraplp
import pandas as pd
import numpy as np
import random
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib as mpl

def data_generation(loc_lb, loc_ub, nv_lb, nv_ub, dem_lb, dem_ub, capacities_equal,
rounding, zero_slack, slack, output) :
    """Generates problem using random numbers
    inputes : -number of locations including depot lower/upper bound
              -number of vehicles lower/upper bound
              -demand level lower/upper bound
              -slack whether sum of demands== sum of capacities bool
              -amount of slackness"""
    # random problem data.
    data = {}
    # number of locations.
    number_loc = random.randint(loc_lb, loc_ub)
    # number of vehicles.
    data["num_vehicles"] = random.randint(nv_lb, nv_ub)
    # random demands.
    data['demands'] = [0]
    for i in range(number_loc - 1):
        data['demands'].append(random.randint(dem_lb, dem_ub))
    # Check for capacities equal and zero slack feasibility
    if zero_slack == True:
        not_zs = True
        while not_zs == True :
            if sum(data['demands'])%data['num_vehicles'] == 0:
                not_zs = False
            else :
                data['demands'] = [0]
                for i in range(number_loc - 1):
                    data['demands'].append(random.randint(dem_lb, dem_ub))
    # number of locations without start.
    data['num_demands'] = len(data['demands']) - 1
    # random vehicles capacities.
    data["vehicle_capacities"] = []
    if capacities_equal == True :
        for i in range(data["num_vehicles"]):

data["vehicle_capacities"].append(round(sum(data['demands'])/data["num_vehicles"]))
        # Feasibility check.
        while sum(data['vehicle_capacities']) < sum(data['demands']):
            data['vehicle_capacities'] = [x + 1 for x in data['vehicle_capacities']]
    if capacities_equal == False :
        random_floats = np.random.dirichlet(np.ones(data["num_vehicles"]))
        for i in range(data["num_vehicles"]):

data["vehicle_capacities"].append(round(sum(data['demands'])*random_floats[i]))
        data['vehicle_capacities'].sort(reverse = True)
        # Feasibility check.
        while sum(data['vehicle_capacities']) != sum(data['demands']):
```

```
            if sum(data['vehicle_capacities']) > sum(data['demands']):
                data['vehicle_capacities'][0] -= 1
            if sum(data['vehicle_capacities']) < sum(data['demands']):
                data['vehicle_capacities'][0] += 1
    # Overslack
    if slack > 0:
        while  sum(data["vehicle_capacities"][:-slack]) < sum(data['demands']):
            if capacities_equal == True :
                data['vehicle_capacities'] = [x + 1 for x in
data['vehicle_capacities']]
            if capacities_equal == False :
                data['vehicle_capacities'][random.randint(0,data['num_vehicles']-1)]
+= 1
                data['vehicle_capacities'].sort(reverse = True)
    # Round vehicle capaciies.
    if rounding == True :
        for i in range(data['num_vehicles']):
            while data['vehicle_capacities'][i]%10 != 0:
                data['vehicle_capacities'][i] += 1
    # random coordinates
    locs = {}
    for i in range(number_loc):
        locs[f'location_{i}'] = (random.randint(0, 10000), random.randint(0, 10000))
    # location distances
    data["distance_matrix"] = []
    Distance_Matrix = pd.DataFrame(columns = [str(i) for i in range(number_loc)])
    for i in range(number_loc):
        iline = []
        for j in range(number_loc):
            iline.append(round(((locs[f'location_{i}'][0]-locs[f'location_{j}'][0])**2
+
                                (locs[f'location_{i}'][1]-
locs[f'location_{j}'][1])**2)**0.5))
        Distance_Matrix.loc[len(Distance_Matrix)] = iline
        data["distance_matrix"].append(iline)
    data["depot"] = 0
    # Display Data
    if output == True :
        print('---------------PROBLEM DATA---------------')
        print('Number of locations = ',number_loc)
        print('Sum demands = ',sum(data['demands']))
        print('Sum capacities = ', sum(data["vehicle_capacities"]))
        print('Number of vehicles = ', data['num_vehicles'])
        print('Vehicles capacities = ', data['vehicle_capacities'])
        print('Number of demands = ', data['num_demands'])
        print('Demand levels = ', data['demands'])
        print('Location Coordinates :')
        print(locs)
        #print(Distance_Matrix)
    return data, locs, number_loc, Distance_Matrix

# Solve VRP
def main(manual, loc_lb, loc_ub, nv_lb, nv_ub, dem_lb, dem_ub, capacities_equal,
rounding, zero_slack, slack, time_limit, output):
    """Solves VRP """
    if manual == False :
            data, locs, number_loc, dm = data_generation(loc_lb, loc_ub, nv_lb, nv_ub,
dem_lb, dem_ub, capacities_equal, rounding, zero_slack, slack, output)
    elif manual == True : data, locs, number_loc = data_local, locs_local,
number_loc_local
```

```
    #                                   CVRP
    # routes data
    def get_routes_vrp(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))
                route.append(manager.IndexToNode(index))
            routes.append(route)
        return routes
    # solution formate function
    def print_solution_vrp(data, manager, routing, solution):
        """Prints solution on console."""
        print(f"Objective: {solution.ObjectiveValue()}")
        total_distance = 0
        total_load = 0
        for vehicle_id in range(data["num_vehicles"]):
            index = routing.Start(vehicle_id)
            plan_output = f"Route for vehicle {vehicle_id}:\n"
            route_distance = 0
            route_load = 0
            while not routing.IsEnd(index):
                node_index = manager.IndexToNode(index)
                route_load += data["demands"][node_index]
                plan_output += f" {node_index} Load({route_load}) -> "
                previous_index = index
                index = solution.Value(routing.NextVar(index))
                route_distance += routing.GetArcCostForVehicle(previous_index, index,
vehicle_id)
            plan_output += f" {manager.IndexToNode(index)} Load({route_load})\n"
            plan_output += f"Distance of the route: {route_distance}m\n"
            plan_output += f"Load of the route: {route_load}\n"
            print(plan_output)
            total_distance += route_distance
            total_load += route_load
        print(f"Total distance of all routes: {total_distance}m")
        print(f"Total load of all routes: {total_load}")

    # Instantiate the data problem.
    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(len(data["distance_matrix"]),
data["num_vehicles"], data["depot"])
    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)

    # Create and register a transit callback.
    def distance_callback_vrp(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data["distance_matrix"][from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback_vrp)

    # Define cost of each arc.
```

```
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Add Capacity constraint.
    def demand_callback_vrp(from_index):
        """Returns the demand of the node."""
        # Convert from routing variable Index to demands NodeIndex.
        from_node = manager.IndexToNode(from_index)
        return data["demands"][from_node]

    demand_callback_index = routing.RegisterUnaryTransitCallback(demand_callback_vrp)
    routing.AddDimensionWithVehicleCapacity(
        demand_callback_index,
        0,  # null capacity slack
        data["vehicle_capacities"],
        # vehicle maximum capacities
        True,  # start cumul to zero
        "Capacity",
    )

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.FromSeconds(time_limit)

    # Solve the problem.
    solution = routing.SolveWithParameters(search_parameters)

    # Print and Save solution on console.
    if solution:
        routes_vrp = get_routes_vrp(solution, routing, manager)
        obj_vrp = solution.ObjectiveValue()
        if output == True :
            print('---------------SOLUTION CVRP---------------')
            print_solution_vrp(data, manager, routing, solution)
            print('Routes :')
            print(routes_vrp)
            print("Solver status: ", routing.status())
            solver_status = routing.status()
    else :
        return False

    #                     FEASIBLE ASSIGNMENT AND TSP
    # Assignment
    # Solver
    # Create the mip solver with the SCIP backend.
    solver = pywraplp.Solver.CreateSolver("SCIP")
    if not solver:
        print("not solver")

    # Variables
    # x[i, j] is an array of 0-1 variables, which will be 1
    # if worker i is assigned to task j.
    x = {}
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            x[i, j] = solver.IntVar(0, 1, "")

    # Constraints
```

```
    # Each vehicle does not exceed its capacities.
    for i in range(data["num_vehicles"]):
        solver.Add(solver.Sum([x[i, j]*data["demands"][j+1] for j in
range(data["num_demands"])]) <=

data["vehicle_capacities"][i])
    # Each demand is assigned to exactly one vehicle.
    for j in range(data["num_demands"]):
        solver.Add(solver.Sum([x[i, j] for i in range(data["num_vehicles"])]) == 1)

    # Objective Σ x(i,j)
    objective_terms = []
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            objective_terms.append(x[i, j])
    solver.Minimize(solver.Sum(objective_terms))

    # Solve
    print(f"Solving with {solver.SolverVersion()}")
    status = solver.Solve()

    # Print assignment solution.
    if output == True :
        print('---------------SOLUTION FEASIBLE ASSIGNMENT AND TSP---------------')
        # Print assignment solution.
        if status == pywraplp.Solver.OPTIMAL or status == pywraplp.Solver.FEASIBLE:
            print(f"Total cost = {solver.Objective().Value()}\n")
            for i in range(data["num_vehicles"]):
                for j in range(data["num_demands"]):
                    # Test if x[i,j] is 1 (with tolerance for floating point
arithmetic).
                    if x[i, j].solution_value() > 0.5:
                        print(f"Vehicle {i} assigned to location {j+1}.")
        else:
            print("No solution found.")

    # Create the data from assignment to use for TSP
    # Separate locations as assigned to vehicles
    routes_tsp_dict = {}
    for i in range(data["num_vehicles"]):
        routes_tsp_dict[f'vehicle{i}'] = [0]
        for j in range(data["num_demands"]):
            # Test if x[i,j] is 1 (with tolerance for floating point arithmetic).
            if x[i, j].solution_value() > 0.5:
                routes_tsp_dict[f"vehicle{i}"].append(j+1)
        routes_tsp_dict[f'vehicle{i}'].append(0)
    # Data
    data_tsp = {}
    for i in range(data["num_vehicles"]):
        data_tsp[f"distance_matrix{i}"] = []
        data_tsp[f"demands{i}"] = []
        data_tsp[f"vehicle{i}_cap"] = data["vehicle_capacities"][i]
        for l in range(len(routes_tsp_dict[f"vehicle{i}"][:-1])):
            data_tsp[f"distance_matrix{i}"].append([])
            m = 0
        for j in routes_tsp_dict[f"vehicle{i}"][:-1]:
            data_tsp[f"demands{i}"].append(data["demands"][j])
            for k in routes_tsp_dict[f"vehicle{i}"][:-1]:

data_tsp[f"distance_matrix{i}"][m].append(data["distance_matrix"][j][k])
            m += 1
```

```
    data_tsp["num_vehicles"] = 1
    data_tsp["depot"] = 0

    # TSP
    # Print solution for each tsp problem
    def print_solution(manager, routing, solution):
        """Prints solution on console."""
        print(f"TSP_{i} Objecive value (distance) : {solution.ObjectiveValue()} ")
        index = routing.Start(0)
        plan_output = f"Route for vehicle {i}:\n"
        route_distance = 0
        route_load = 0
        while not routing.IsEnd(index):
            route_load += data_tsp[f'demands{i}'][manager.IndexToNode(index)]
            plan_output += f"
{routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]} Load({route_load}) ->"
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
        plan_output += f" {routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]}
Load({route_load}) \n"
        plan_output += f"Distance of the route: {route_distance}\n"
        plan_output += f"Load of the route: {route_load}\n"
        print(plan_output)

    # Routes for tsp problems
    def get_routes(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes_tsp = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))

route.append(routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)])
            routes_tsp.append(route)
        return routes_tsp

    # Run TSP for all vehicles
    obj_tsp = 0
    routes_tsp = []
    for i in range(data["num_vehicles"]):

        # Create the routing index manager.
        manager = pywrapcp.RoutingIndexManager(
            len(data_tsp[f"distance_matrix{i}"]), data_tsp["num_vehicles"],
data_tsp["depot"])

        # Create Routing Model.
        routing = pywrapcp.RoutingModel(manager)

        def distance_callback(from_index, to_index):
            """Returns the distance between the two nodes."""
            # Convert from routing variable Index to distance matrix NodeIndex.
            from_node = manager.IndexToNode(from_index)
            to_node = manager.IndexToNode(to_index)
            return data_tsp[f"distance_matrix{i}"][from_node][to_node]
```

```
        transit_callback_index = routing.RegisterTransitCallback(distance_callback)

        # Define cost of each arc.
        routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

        # Setting first solution heuristic.
        search_parameters = pywrapcp.DefaultRoutingSearchParameters()
        search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
        search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
        search_parameters.time_limit.FromSeconds(time_limit)

        # Solve the problem.
        solution = routing.SolveWithParameters(search_parameters)

        # Print and Save solution on console.
        if solution:
            if output == True :
                print_solution(manager, routing, solution)
            obj_tsp += solution.ObjectiveValue()
            routes_tsp.append(get_routes(solution, routing, manager)[0])
    if output == True :
        print("Objective value =",obj_tsp)
        print("All routes :\n", routes_tsp)


    #                           CLUSTER CREATION
    # tsp for one vehicle with unlimited capacity
    # presolution for assignment problem

    def print_solution_TSP(manager, routing, solution):
        """Prints solution on console."""
        print(f"Objective TSP SINGLE VEHICLE: {solution.ObjectiveValue()}")
        index = routing.Start(0)
        plan_output = "Route for vehicle :\n"
        route_distance = 0
        while not routing.IsEnd(index):
            plan_output += f" {manager.IndexToNode(index)} ->"
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
        plan_output += f" {manager.IndexToNode(index)}\n"
        print(plan_output)

    def get_routes_TSP(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))
                route.append(manager.IndexToNode(index))
            routes.append(route)
        return routes

    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(
```

```
            len(data["distance_matrix"]), 1, data["depot"])
    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)


    def distance_callback_TSP(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data["distance_matrix"][from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback_TSP)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.FromSeconds(time_limit)

    # Solve the problem.
    solution = routing.SolveWithParameters(search_parameters)

    # Print on console and Save solution.
    if solution:
        if output == True :
            print('---------------SOLUTION TSP SINGLE VEHICLE---------------')
            print_solution_TSP(manager, routing, solution)
        routes_TSP = get_routes_TSP(solution, routing, manager)
        obj_TSP = solution.ObjectiveValue()

    # Create clusters
    clusters = {}
    clusters["cluster_0"] = []
    load = 0
    vehicle_index = 0
    for i in routes_TSP[0][1:-1]:
        # Add locations to clusters
        if data['vehicle_capacities'][vehicle_index] >= load + data["demands"][i]: #
if next load fits in current vehicle
            load += data["demands"][i]
            clusters[f"cluster_{vehicle_index}"].append(i)
        # if current vehicle is last add remaining locations to last cluster
        elif data['vehicle_capacities'][vehicle_index] <= load + data["demands"][i]
and vehicle_index == data['num_vehicles'] - 1 :
            clusters[f"cluster_{vehicle_index}"].append(i)
        # if next load doesn't fit in current vehicle
        else:
            vehicle_index += 1
            clusters[f"cluster_{vehicle_index}"] = []
            clusters[f"cluster_{vehicle_index}"].append(i)
            load = 0
            load += data["demands"][i]

    # Cluster center coordinates
    cluster_center_coords = {}
```

```
    for i in range(len(clusters)):
        cluster_center_coords[f'center_{i}_coords'] = [0,0]
        div = len(clusters[f'cluster_{i}'])
        for j in clusters[f'cluster_{i}']:
            cluster_center_coords[f'center_{i}_coords'][0] +=
locs[f'location_{j}'][0]/div
            cluster_center_coords[f'center_{i}_coords'][1] +=
locs[f'location_{j}'][1]/div

    # Distance matrix of nodes to clusters
    data_cl = {}
    data_cl['cluster_distance_martix'] = []
    for i in range(len(cluster_center_coords)):
        iline = []
        for j in range(1,number_loc):
            iline.append(round(((cluster_center_coords[f'center_{i}_coords'][0]-
locs[f'location_{j}'][0])**2 +
                               (cluster_center_coords[f'center_{i}_coords'][1]-
locs[f'location_{j}'][1])**2)**0.5))
        data_cl['cluster_distance_martix'].append(iline)
    if len(clusters) < data['num_vehicles']:
        while len(data_cl['cluster_distance_martix']) != data['num_vehicles']:
            data_cl['cluster_distance_martix'].append([10000000 for i in
range(data['num_demands'])])

    #                                   CLUSTERING
    # Assignment
    # Solver
    # Create the mip solver with the SCIP backend.
    solver = pywraplp.Solver.CreateSolver("SCIP")
    if not solver:
        print("not solver")

    # Variables
    # x[i, j] is an array of 0-1 variables, which will be 1
    # if worker i is assigned to task j.
    x = {}
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            x[i, j] = solver.IntVar(0, 1, "")

    # Constraints
    # Each worker is assigned to at most 1 task.
    for i in range(data["num_vehicles"]):
        solver.Add(solver.Sum([x[i, j]*data["demands"][j+1] for j in
range(data["num_demands"])]) <=

data["vehicle_capacities"][i])
    # Each task is assigned to exactly one worker.
    for j in range(data["num_demands"]):
        solver.Add(solver.Sum([x[i, j] for i in range(data["num_vehicles"])]) == 1)

    # Objective Σ x(i,j)*distance(i_cluster to j_demand)
    objective_terms = []
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            objective_terms.append(x[i, j]*data_cl['cluster_distance_martix'][i][j])
    solver.Minimize(solver.Sum(objective_terms))

    # Solve
    print(f"Solving with {solver.SolverVersion()}")
```

```
    status = solver.Solve()

    # Print assignment solution.
    if output == True :
        print('---------------SOLUTION CLUSTERING ASSIGNMENT AND TSP---------------')
        if status == pywraplp.Solver.OPTIMAL or status == pywraplp.Solver.FEASIBLE:
            print(f"Total cost = {solver.Objective().Value()}\n")
            for i in range(data["num_vehicles"]):
                for j in range(data["num_demands"]):
                    # Test if x[i,j] is 1 (with tolerance for floating point
arithmetic).
                    if x[i, j].solution_value() > 0.5:
                        print(f"Vehicle {i} assigned to location {j+1}.")
        else:
            print("No solution found.")

    # Create the data from assignment to use for TSP
    # Separate locations as assigned to vehicles
    routes_cl_dict = {}
    for i in range(data["num_vehicles"]):
        routes_tsp_dict[f'vehicle{i}'] = [0]
        for j in range(data["num_demands"]):
            # Test if x[i,j] is 1 (with tolerance for floating point arithmetic).
            if x[i, j].solution_value() > 0.5:
                routes_tsp_dict[f"vehicle{i}"].append(j+1)
        routes_tsp_dict[f'vehicle{i}'].append(0)

    # Data.
    data_tsp = {}
    for i in range(data["num_vehicles"]):
        data_tsp[f"distance_matrix{i}"] = []
        data_tsp[f"demands{i}"] = []
        data_tsp[f"vehicle{i}_cap"] = data["vehicle_capacities"][i]
        for l in range(len(routes_tsp_dict[f"vehicle{i}"][:-1])):
            data_tsp[f"distance_matrix{i}"].append([])
            m = 0
        for j in routes_tsp_dict[f"vehicle{i}"][:-1]:
            data_tsp[f"demands{i}"].append(data["demands"][j])
            for k in routes_tsp_dict[f"vehicle{i}"][:-1]:

data_tsp[f"distance_matrix{i}"][m].append(data["distance_matrix"][j][k])
            m += 1
    data_tsp["num_vehicles"] = 1
    data_tsp["depot"] = 0

    # TSP
    # Print solution for each tsp problem
    def print_solution_cl(manager, routing, solution):
        """Prints solution on console."""
        print(f"TSP_{i} Objecive value (distance) : {solution.ObjectiveValue()} ")
        index = routing.Start(0)
        plan_output = f"Route for vehicle {i}:\n"
        route_distance = 0
        route_load = 0
        while not routing.IsEnd(index):
            route_load += data_tsp[f'demands{i}'][manager.IndexToNode(index)]
            plan_output += f"
{routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]} Load({route_load}) ->"
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
```

```
        plan_output += f" {routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]}
Load({route_load}) \n"
        plan_output += f"Distance of the route: {route_distance}\n"
        plan_output += f"Load of the route: {route_load}\n"
        print(plan_output)

    # Routes for tsp problems.
    def get_routes_cl(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes_tsp = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))

route.append(routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)])
            routes_tsp.append(route)
        return routes_tsp

    #run tsp for all vehicles
    obj_cl = 0
    routes_cl = []
    for i in range(data["num_vehicles"]):

        # Create the routing index manager.
        manager = pywrapcp.RoutingIndexManager(
            len(data_tsp[f"distance_matrix{i}"]), data_tsp["num_vehicles"],
data_tsp["depot"])

        # Create Routing Model.
        routing = pywrapcp.RoutingModel(manager)


        def distance_callback(from_index, to_index):
            """Returns the distance between the two nodes."""
            # Convert from routing variable Index to distance matrix NodeIndex.
            from_node = manager.IndexToNode(from_index)
            to_node = manager.IndexToNode(to_index)
            return data_tsp[f"distance_matrix{i}"][from_node][to_node]

        transit_callback_index = routing.RegisterTransitCallback(distance_callback)

        # Define cost of each arc.
        routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

        # Setting first solution heuristic.
        search_parameters = pywrapcp.DefaultRoutingSearchParameters()
        search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
        search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
        search_parameters.time_limit.FromSeconds(time_limit)

        # Solve the problem.
        solution = routing.SolveWithParameters(search_parameters)

        # Print solution on console.
        if solution:
```

```
            if output == True :
                print_solution_cl(manager, routing, solution)
            obj_cl += solution.ObjectiveValue()
            routes_cl.append(get_routes_cl(solution, routing, manager)[0])
    if output == True :
        print("Objective value =",obj_cl)
        print("All routes :\n", routes_cl)


    #                               CLUSTERING 2
    # assignment
    # Solver
    # Create the mip solver with the SCIP backend.
    solver = pywraplp.Solver.CreateSolver("SCIP")
    if not solver:
        print("not solver")

    # Variables
    # x[i, j] is an array of 0-1 variables, which will be 1
    # if worker i is assigned to task j.
    x = {}
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            x[i, j] = solver.IntVar(0, 1, "")

    # Constraints
    # Each worker is assigned to at most 1 task.
    for i in range(data["num_vehicles"]):
        solver.Add(solver.Sum([x[i, j]*data["demands"][j+1] for j in
range(data["num_demands"])]) <=

data["vehicle_capacities"][i])

    # Each task is assigned to exactly one worker.
    for j in range(data["num_demands"]):
        solver.Add(solver.Sum([x[i, j] for i in range(data["num_vehicles"])]) == 1)

    # Objective Σ x(i,j)*distance(i_cluster to j_demand)^2
    objective_terms = []
    for i in range(data["num_vehicles"]):
        for j in range(data["num_demands"]):
            objective_terms.append(x[i,
j]*(data_cl['cluster_distance_martix'][i][j]**2))
    solver.Minimize(solver.Sum(objective_terms))

    # Solve
    print(f"Solving with {solver.SolverVersion()}")
    status = solver.Solve()

    # Print assignment solution.
    if output == True :
        print('---------------SOLUTION CLUSTERING dis^2 ASSIGNMENT AND TSP------------
---')
        if status == pywraplp.Solver.OPTIMAL or status == pywraplp.Solver.FEASIBLE:
            print(f"Total cost = {solver.Objective().Value()}\n")
            for i in range(data["num_vehicles"]):
                for j in range(data["num_demands"]):
                    # Test if x[i,j] is 1 (with tolerance for floating point
arithmetic).
                    if x[i, j].solution_value() > 0.5:
                        print(f"Vehicle {i} assigned to location {j+1}.")
```

```
        else:
            print("No solution found.")

    # Create the data from assignment to use for TSP
    # Separate locations as assigned to vehicles
    for i in range(data["num_vehicles"]):
        routes_tsp_dict[f'vehicle{i}'] = [0]
        for j in range(data["num_demands"]):
            # Test if x[i,j] is 1 (with tolerance for floating point arithmetic).
            if x[i, j].solution_value() > 0.5:
                routes_tsp_dict[f"vehicle{i}"].append(j+1)
        routes_tsp_dict[f'vehicle{i}'].append(0)

    # Data.
    data_tsp = {}
    for i in range(data["num_vehicles"]):
        data_tsp[f"distance_matrix{i}"] = []
        data_tsp[f"demands{i}"] = []
        data_tsp[f"vehicle{i}_cap"] = data["vehicle_capacities"][i]
        for l in range(len(routes_tsp_dict[f"vehicle{i}"][:-1])):
            data_tsp[f"distance_matrix{i}"].append([])
            m = 0
        for j in routes_tsp_dict[f"vehicle{i}"][:-1]:
            data_tsp[f"demands{i}"].append(data["demands"][j])
            for k in routes_tsp_dict[f"vehicle{i}"][:-1]:

data_tsp[f"distance_matrix{i}"][m].append(data["distance_matrix"][j][k])
            m += 1
    data_tsp["num_vehicles"] = 1
    data_tsp["depot"] = 0

    # TSP
    # Print solution for each tsp problem
    def print_solution_cl2(manager, routing, solution):
        """Prints solution on console."""
        print(f"TSP_{i} Objecive value (distance) : {solution.ObjectiveValue()} ")
        index = routing.Start(0)
        plan_output = f"Route for vehicle {i}:\n"
        route_distance = 0
        route_load = 0
        while not routing.IsEnd(index):
            route_load += data_tsp[f'demands{i}'][manager.IndexToNode(index)]
            plan_output += f"
{routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]} Load({route_load}) ->"
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
        plan_output += f" {routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)]}
Load({route_load}) \n"
        plan_output += f"Distance of the route: {route_distance}\n"
        plan_output += f"Load of the route: {route_load}\n"
        print(plan_output)

    # Routes for tsp problems
    def get_routes_cl2(solution, routing, manager):
        """Get vehicle routes from a solution and store them in an array."""
        # Get vehicle routes and store them in a two-dimensional array whose
        # i,j entry is the jth location visited by vehicle i along its route.
        routes_tsp = []
        for route_nbr in range(routing.vehicles()):
            index = routing.Start(route_nbr)
```

```
            route = [manager.IndexToNode(index)]
            while not routing.IsEnd(index):
                index = solution.Value(routing.NextVar(index))

route.append(routes_tsp_dict[f'vehicle{i}'][manager.IndexToNode(index)])
            routes_tsp.append(route)
        return routes_tsp

    # Run tsp for all vehicles
    obj_cl2 = 0
    routes_cl2 = []
    for i in range(data["num_vehicles"]):

        # Create the routing index manager.
        manager = pywrapcp.RoutingIndexManager(
            len(data_tsp[f"distance_matrix{i}"]), data_tsp["num_vehicles"],
data_tsp["depot"]
        )

        # Create Routing Model.
        routing = pywrapcp.RoutingModel(manager)


        def distance_callback(from_index, to_index):
            """Returns the distance between the two nodes."""
            # Convert from routing variable Index to distance matrix NodeIndex.
            from_node = manager.IndexToNode(from_index)
            to_node = manager.IndexToNode(to_index)
            return data_tsp[f"distance_matrix{i}"][from_node][to_node]

        transit_callback_index = routing.RegisterTransitCallback(distance_callback)

        # Define cost of each arc.
        routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

        # Setting first solution heuristic.
        search_parameters = pywrapcp.DefaultRoutingSearchParameters()
        search_parameters.first_solution_strategy =
(routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
        search_parameters.local_search_metaheuristic =
(routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
        search_parameters.time_limit.FromSeconds(time_limit)

        # Solve the problem.
        solution = routing.SolveWithParameters(search_parameters)

        # Print solution on console.
        if solution:
            if output == True :
                print_solution_cl(manager, routing, solution)
            obj_cl2 += solution.ObjectiveValue()
            routes_cl2.append(get_routes_cl(solution, routing, manager)[0])
    if output == True :
        print("Objective value =",obj_cl2)
        print("All routes :\n", routes_cl2)

    #                           GRAPHIC
    if output == True:
        # graph presentation
        fig, ax = plt.subplots(3, 2, figsize=(10,15))
        fig.delaxes(ax[2][1])
```

```
G_vrp = nx.Graph()
G_tsp = nx.Graph()
G_TSP = nx.Graph()
G_cl = nx.Graph()
G_cl2 = nx.Graph()
#  nodes
for node in range(len(data['demands'])):
    G_vrp.add_node(node)
    G_tsp.add_node(node)
    G_TSP.add_node(node)
    G_cl.add_node(node)
    G_cl2.add_node(node)
for node in range(len(clusters)):
    G_TSP.add_node(f"cluster_{node}")
# position dictionary
coords = []
stations = []
coords_cl = []
stations_cl = []
for loc in locs:
    coords.append(locs[loc])
for i in range(len(data['demands'])):
    stations.append(i)
pos = dict(zip(stations, coords))
for loc in cluster_center_coords:
    coords_cl.append(cluster_center_coords[loc])
for i in range(len(clusters)):
    stations_cl.append(f"cluster_{i}")
pos.update(dict(zip(stations_cl, coords_cl)))
# edges
edges_path=[]
for i in range(len(routes_vrp)):
    edges_path.append(list(zip(routes_vrp[i],routes_vrp[i][1:])))
for i in range(len(edges_path)):
    for j in range(len(edges_path[i])):
        G_vrp.add_edge(edges_path[i][j][0],edges_path[i][j][1])
edges_path_tsp=[]
for i in range(len(routes_tsp)):
    edges_path_tsp.append(list(zip(routes_tsp[i],routes_tsp[i][1:])))
for i in range(len(edges_path_tsp)):
    for j in range(len(edges_path_tsp[i])):
        G_tsp.add_edge(edges_path_tsp[i][j][0],edges_path_tsp[i][j][1])
edges_path_TSP=[]
for i in range(len(routes_TSP)):
    edges_path_TSP.append(list(zip(routes_TSP[i],routes_TSP[i][1:])))
for i in range(len(edges_path_TSP)):
    for j in range(len(edges_path_TSP[i])):
        G_TSP.add_edge(edges_path_TSP[i][j][0],edges_path_TSP[i][j][1])
edges_path_cl=[]
for i in range(len(routes_cl)):
    edges_path_cl.append(list(zip(routes_cl[i],routes_cl[i][1:])))
for i in range(len(edges_path_cl)):
    for j in range(len(edges_path_cl[i])):
        G_cl.add_edge(edges_path_cl[i][j][0],edges_path_cl[i][j][1])
edges_path_cl2=[]
for i in range(len(routes_cl2)):
    edges_path_cl2.append(list(zip(routes_cl2[i],routes_cl2[i][1:])))
for i in range(len(edges_path_cl2)):
    for j in range(len(edges_path_cl2[i])):
        G_cl2.add_edge(edges_path_cl2[i][j][0],edges_path_cl2[i][j][1])
# color for each route
```

```
        colors =
['red','green','blue','yellow','cyan','magenta','orange','purple','brown',

'pink','teal','lavender','maroon','navy','olive','coral','indigo','salmon','turquoise'
,'gold']
        # macth edges to routes
        edges_path_d_cvrp = []
        for i in range(len(edges_path)):
            edges_path_r = [(y,x) for (x,y) in edges_path[i]]
            edges_path_r = edges_path_r + edges_path[i]
            edges_path_d_cvrp.append(list(set(edges_path_r)))
        edges_path_d_tsp = []
        for i in range(len(edges_path_tsp)):
            edges_path_r = [(y,x) for (x,y) in edges_path_tsp[i]]
            edges_path_r = edges_path_r + edges_path_tsp[i]
            edges_path_d_tsp.append(list(set(edges_path_r)))
        edges_path_d_cl = []
        for i in range(len(edges_path_cl)):
            edges_path_r = [(y,x) for (x,y) in edges_path_cl[i]]
            edges_path_r = edges_path_r + edges_path_cl[i]
            edges_path_d_cl.append(list(set(edges_path_r)))
        edges_path_d_cl2 = []
        for i in range(len(edges_path_cl2)):
            edges_path_r = [(y,x) for (x,y) in edges_path_cl2[i]]
            edges_path_r = edges_path_r + edges_path_cl2[i]
            edges_path_d_cl2.append(list(set(edges_path_r)))
        # assigne color to edges
        edge_colors_cvrp=[]
        order_edges_cvrp = list(G_vrp.edges())
        for k in range(len(order_edges_cvrp)):
            for i in range(len(edges_path_d_cvrp)):
                for j in range(len(edges_path_d_cvrp[i])):
                    if edges_path_d_cvrp[i][j] == order_edges_cvrp[k]:
                        edge_colors_cvrp.append(colors[i])
        edge_colors_tsp=[]
        order_edges_tsp = list(G_tsp.edges())
        for k in range(len(order_edges_tsp)):
            for i in range(len(edges_path_d_tsp)):
                for j in range(len(edges_path_d_tsp[i])):
                    if edges_path_d_tsp[i][j] == order_edges_tsp[k]:
                        edge_colors_tsp.append(colors[i])
        edge_colors_cl=[]
        order_edges_cl = list(G_cl.edges())
        for k in range(len(order_edges_cl)):
            for i in range(len(edges_path_d_cl)):
                for j in range(len(edges_path_d_cl[i])):
                    if edges_path_d_cl[i][j] == order_edges_cl[k]:
                        edge_colors_cl.append(colors[i])
        edge_colors_cl2=[]
        order_edges_cl2 = list(G_cl2.edges())
        for k in range(len(order_edges_cl2)):
            for i in range(len(edges_path_d_cl2)):
                for j in range(len(edges_path_d_cl2[i])):
                    if edges_path_d_cl2[i][j] == order_edges_cl2[k]:
                        edge_colors_cl2.append(colors[i])
        # node colors
        node_colors = ['gold']
        for node in list(G_TSP.nodes()):
            for i in range(len(clusters)):
                if node in clusters[f'cluster_{i}']:
                    node_colors.append(colors[i])
```

```
        for node in list(G_TSP.nodes()):
            for i in range(len(clusters)):
                if node == f'cluster_{i}':
                    node_colors.append(colors[i])
        # display diagram
        nx.draw(G_vrp, pos, with_labels = True, edge_color= edge_colors_cvrp,
ax=ax[0][0])
        ax[0][0].set_title('Graph CVRP')
        nx.draw(G_tsp, pos, with_labels = True, edge_color= edge_colors_tsp,
ax=ax[0][1])
        ax[0][1].set_title('Graph FEASIBLE ASSIGNMENT AND TSP')
        nx.draw(G_TSP, pos, with_labels = True, node_color=node_colors, ax=ax[1][0])
        ax[1][0].set_title('Graph TSP')
        nx.draw(G_cl, pos, with_labels = True, edge_color= edge_colors_cl,
ax=ax[1][1])
        ax[1][1].set_title('Graph CLUSTERING ASSIGNMENT AND TSP')
        nx.draw(G_cl2, pos, with_labels = True, edge_color= edge_colors_cl2,
ax=ax[2][0])
        ax[2][0].set_title('Graph CLUSTERING dis^2 ASSIGNMENT AND TSP')
        plt.show()

    # return solutions
    return [obj_vrp, solver_status, obj_tsp, obj_TSP, obj_cl, obj_cl2]
```