Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Ευρετικοί Αλγόριθμοι για τη Βελτιστοποίηση Δρομολόγησης Οχημάτων με Φόρτωση σε Διαμερίσματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΩΝ Π. ΛΑΖΑΡΙΔΗΣ

Επιβλέπων : Δημήτριος Φωτάκης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Ευρετικοί Αλγόριθμοι για τη Βελτιστοποίηση Δρομολόγησης Οχημάτων με Φόρτωση σε Διαμερίσματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΩΝ Π. ΛΑΖΑΡΙΔΗΣ

Επιβλέπων : Δημήτριος Φωτάκης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 8η Νοεμβρίου 2021.

.....	.....	.....
Δημήτριος Φωτάκης Αν. Καθηγητής Ε.Μ.Π.	Ιωάννης Εμίρης Καθηγητής Ε.Κ.Π.Α.	Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2021

Ιάσων Π. Λαζαρίδης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της παρούσας εργασίας είναι η ανάπτυξη ενός ευρετικού αλγορίθμου για το Πρόβλημα Δρομολόγησης Διαμερισματοποιημένων Οχημάτων, όπου κάθε διαμέρισμα μπορεί να κρατήσει μέχρι μία παραγγελία. Αυτό το πρόβλημα συναντάται κυρίως στον διαμοιρασμό προϊόντων πετρελαίου και για αυτό είναι επίσης γνωστό ως Πρόβλημα Ανεφοδιασμού Πρατηρίων Βενζίνης.

Ο αλγόριθμος μας βασίζεται στην τεχνική tabu search και μπορεί επίσης να χρησιμοποιηθεί για την επίλυση του κλασσικού Προβλήματος Δρομολόγησης Οχημάτων, αν και κάποια από τα χαρακτηριστικά του έχουν καλύτερη επίδοση στα διαμερισματοποιημένα οχήματα. Επίσης, ο αλγόριθμος μας μπορεί να διαχειριστεί εξίσου καλά ομοιογενείς και ετερογενείς στόλους φορτηγών.

Η ανυπαρξία δημοσίων συνόλων παραδειγμάτων για το Πρόβλημα Ανεφοδιασμού Πρατηρίων Βενζίνης μας οδήγησε στο να χρησιμοποιήσουμε τρία διαφορετικά σύνολα για να αξιολογήσουμε τον αλγόριθμο μας. Το πρώτο είναι το σύνολο A του Augerat το οποίο χρησιμοποιείται για το κλασσικό Πρόβλημα Δρομολόγησης Οχημάτων με απεριόριστο ομοιογενή στόλο οχημάτων και τα παραδείγματα του περιέχουν από 30 έως 80 πελάτες το καθένα. Ο αλγόριθμος αποδείχθηκε πολύ ανταγωνιστικός σε αυτά τα παραδείγματα, με τις λύσεις που βρίσκει κατά μέσο όρο να είμαι μόλις 2% χειρότερες από τις βέλτιστες.

Επιπλέον, τροποποιήσαμε το σύνολο X του Uchoa, που είναι επίσης ένα σύνολο που χρησιμοποιείται σαν σημείο αναφοράς για την αξιολόγηση στο κλασσικό πρόβλημα και προσθέσαμε διαμερίσματα στα οχήματα. Χρησιμοποιήσαμε 5 διαφορετικές κατηγορίες διαμερισμάτων για να κάνουμε το στόλο ετερογενή όπως και στις πραγματικές εφαρμογές. Στις περιπτώσεις αυτές η απευθείας σύγκριση των λύσεων μας με τις βέλτιστες λύσεις χωρίς διαμερίσματα δεν έχει νόημα, καθώς το πρόβλημα με τα διαμερίσματα είναι πολύ πιο δύσκολο. Ο σκοπός μας είναι να αξιολογήσουμε πόσο καλά φορτώνει τα οχήματα ο αλγόριθμος μας, καθώς και αν καταφέρνει να αποφεύγει τοπικά ελάχιστα. Βρήκαμε ότι επιτυγχάνει σε σημαντικό βαθμό και στα 2 αυτά κριτήρια.

Τέλος, αξιολογήσαμε τον αλγόριθμο μας σε ένα σύνολο από πραγματικά δεδομένα. Σε αυτό το σύνολο, συγκρίναμε τα αποτελέσματα μας με έναν αλγόριθμο που είχε αναπτυχθεί ανεξάρτητα και βασίζεται στους αλγόριθμους δρομολόγησης οχημάτων που περιέχονται στη βιβλιοθήκη OR tools της Google. Η σύγκριση αποβαίνει σαφώς υπερ του αλγορίθμου μας.

Λέξεις κλειδιά

Συνδυαστική βελτιστοποίηση, Επιχειρησιακή έρευνα, Τοπική αναζήτηση, Πρόβλημα Δρομολόγησης Οχημάτων, Πρόβλημα Δρομολόγησης Διαμερισματοποιημένων Οχημάτων Πρόβλημα Ανεφοδιασμού Πρατηρίου Βενζίνης.

Abstract

The purpose of this diploma dissertation is the development of a heuristic algorithm for the Multi-Compartment Vehicle Routing Problem, where each compartment can only hold up to one order. This is a problem commonly found in the distribution of oil products and thus it is also known as the Petrol Station Replenishment Problem.

Our algorithm is based on tabu search and can also be used for solving the classic Vehicle Routing Problem, even though some of its features work better for compartmentalized vehicles. It also supports both homogeneous and heterogeneous fleets.

Because no public instances are available for the Petrol Station Replenishment Problem, we had to get creative in how to evaluate our algorithm by using three different sets of instances. The first one was set A of Augerat, which is a set of instances ranging from 30 to 80 customers targeted to the capacitated VRP with unlimited homogeneous vehicles. Our algorithm proved to be very competitive in these particular instances as the solutions it finds are within 2% of the optimal solutions.

We also modified set X of Uchoa, which also is a set of benchmark instances for the CVRP, by adding compartments to the vehicles. Five different compartment categories were used, so as to make the fleet heterogeneous which is also the case in real applications. There comparing our results to the optimal non-compartment solutions is meaningless since they solve a much easier problem than the variation with the compartments. Our purpose in the evaluation in this instances was to see how well our algorithm corresponds in benchmark instances, regarding the loading of the compartments and the avoidance of local minima. We found out that our algorithm succeeds in both as the loadings we found would be considered excellent in the fuel distribution industry and our algorithm can achieve similar results from a random initialization to solutions that are initialized based on the optimal non-compartment solutions.

Finally, we evaluated our algorithm in a set of real instances. In this set we compared our results with those found by an algorithm developed independently and is based on the routing algorithms of Google's OR-Tools library. The comparison clearly favors our algorithm.

Key words

Combinatorial optimization, Operations Research, Local Search, Tabu Search, Vehicle Routing Problem, Multi-Compartment Vehicle Routing Problem, Petrol Station Replenishment Problem.

Ευχαριστίες

Κατ’ αρχάς θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας κ. Φωτάκη για την εμπιστοσύνη που μου επέδειξε και την καθοδήγηση του. Επίσης, θα ήθελα να ευχαριστήσω τους φίλους μου χάρη στους οποίους τα χρόνια φοίτησης μου στο Εθνικό Μετσόβιο Πολυτεχνείο θα μου μείνουν για πάντα σαν μια υπέροχη εμπειρία. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδελφό μου για τη διαρκή στήριξη τους.

Ιάσων Π. Λαζαρίδης,

Αθήνα, 8η Νοεμβρίου 2021

Περιεχόμενα

Εκτεταμένη Ελληνική Περίληψη

Εισαγωγή

Σε αυτή τη διπλωματική εργασία εξετάζουμε το Πρόβλημα Δρομολόγησης Οχήματος (Vehicle Routing Problem ή VRP). Το VRP είναι ένα κλασσικό πρόβλημα συνδυαστικής βελτιστοποιήσης το οποίο μας συστήθηκε για πρώτη φορά το 1959 [Dant59] και έχει μελετηθεί εκτενώς από τότε και μετά. Στο Πρόβλημα Δρομολόγησης Οχήματος, μας δίνεται ένα σύνολο από κόμβους, από τους οποίους ένας είναι η αποθήκη εφοδίων και οι υπόλοιποι αντιπροσωπεύουν τους πελάτες. Κάθε πελάτης απαιτεί μια ποσότητα q_i του προϊόντος να παραδοθεί σε αυτόν. Για το σκοπό αυτό, γίνεται χρήση ενός απεριόριστου ομοιογενούς στόλου φορτηγών οχημάτων, καθένα εκ των οποίων έχει χωρητικότητα C. Κάθε παραγγελία πρέπει να παραδοθεί από ένα μόνο όχημα και δεν μπορεί να διασπαστεί, ενώ κάθε όχημα μπορεί να κάνει μόνο ένα ταξίδι. Ο στόχος είναι να βρεθούν οι διαδρομές που ελαχιστοποιούν τη συνολική απόσταση ταξιδιού, σεβόμενοι της χωρητικότητες των οχημάτων.

Οι ποικίλες πρακτικές του VRP το έχουν οδηγήσει στο να είναι ένα από τα πιο μελετημένα προβλήματα στο χώρο της επιχειρησιακής έρευνας. Αυτό είναι αυταπόδεικτο από το μεγάλο αριθμό συγκεντρωτικών ερευνών [Eksi09, Brae16] πάνω στις μεθόδους που έχουν προταθεί για το VRP και τις παραλλαγές του. Μέχρι και βιβλία έχουν εκδοθεί που ασχολούνται αποκλειστικά με το Πρόβλημα Δρομολόγησης Οχήματος [Toth02a, Gold08, Toth14].

Το Πρόβλημα Δρομολόγησης οχήματος ανήκει στην κατηγορία των NP-Hard προβλημάτων μιας και πρόκειται για γενίκευση του κλασσικού προβλήματος του Πλανόδιου Πωλητή. Αυτό έχει σαν αποτέλεσμα την αδυναμία εύρεσης αλγορίθμων που να μπορούν να επιλύσουν περιστατικά με πολλούς πελάτες σε λογικό χρονικό διάστημα. Για να αντιμετωπιστεί αυτό το πρόβλημα οι ερευνητές έχουν επικεντρωθεί σε ευρετικές μεθόδους που λύνουν προσεγγιστικά το πρόβλημα. Σε αυτήν την κατεύθυνση έχουν αναπτυχθεί αλγόριθμοι που χρησιμοποιούν μεθόδους τοπικής αναζήτησης [Perr] ή έναν συνδυασμό τοπικής αναζήτησης και ακέραιου προγραμματισμού [Subr13] με εξαιρετικά αποτελέσματα στο κλασσικό Πρόβλημα Δρομολόγησης Οχημάτων.

Παραλλαγές

Όπως είπαμε και νωρίτερα το Πρόβλημα Δρομολόγησης οχήματος έχει πολλαπλές πραγματικές εφαρμογές. Σε πολλές από αυτές οι περιορισμοί και τα χαρακτηριστικά είναι ελαφρώς διαφορετικά από αυτά που περιγράψαμε νωρίτερα και γι' αυτό έχουν δημιουργηθεί πολλές παραλλαγές του VRP. Οι πιο σημαντικές από αυτές είναι οι ακόλουθες:

HFVRP

Σε αυτήν την παραλλαγή αντί για έναν ομοιογενή στόλο έχουμε έναν ετερογενή όπου κάθε όχημα μπορεί να έχει διαφορετικά χαρακτηριστικά από τα υπόλοιπα. Στην πράξη συνήθως υπάρχουν κάποιες κατηγορίες και κάθε όχημα ανήκει σε μία από αυτές. Ο στόλος επίσης μπορεί να είναι και συγκεκριμένος αντί για απεριόριστος. Το HFVRP περιγράφεται από το παρακάτω ακέραιο πρόγραμμα.

Ας θεωρήσουμε ότι V είναι το σύνολο όλων των κόμβων, με το V' να αντιπροσωπεύει το υποσύνολο του V που περιέχει μόνο τους πελάτες. Θεωρούμε ακόμα ότι M είναι το σύνολο όλων των διαθέσιμων οχημάτων. Έστω επίσης ότι R είναι το σύνολο όλων των εφικτών διαδρομών για όλα τα οχήματα, $\mathcal{R}_i \subseteq R$ το υποσύνολο των διαδρομών που περιέχει τον πελάτη $i \in V'$, και $R_k \subseteq R$ το σύνολο όλων των διαδρομών που σχετίζονται με το όχημα $k \in M$. Ορίζουμε ως y_j την δυαδική μεταβλητή που σχετίζεται με τη διαδρομή $j \in R$ και c_j το συνολικό κόστος της διαδρομής. Ακολουθεί το ακέραιο πρόγραμμα που περιγράφει το HFVRP.

$$\min \quad \sum_{j \in R} c_j y_j \tag{0.1}$$

$$\text{s.t} \quad \sum_{j \in \mathcal{R}_i} y_j = 1 \quad \forall i \in V' \tag{0.2}$$

$$\sum_{j \in R_k} y_j \leq 1 \quad \forall k \in M \tag{0.3}$$

$$y_j \in \{0, 1\}. \tag{0.4}$$

Η αντικειμενική συνάρτηση (0.1) ελαχιστοποιεί το άθροισμα του κόστους επιλέγοντας τους κατάλληλους συνδυασμούς διαδρομών. Οι περιορισμοί (0.2) δηλώνουν ότι μόνο μία διαδρομή πρέπει του υποσυνόλου $\mathcal{R}_i$ πρέπει να επισκεφτεί τον πελάτη $i \in V'$. Οι περιορισμοί (0.3) προσδιορίζουν ότι κάθε όχημα μπορεί να χρησιμοποιηθεί το πολύ μία φορά. Τέλος, οι περιορισμοί (0.4) ορίζουν το εύρος των μεταβλητών.

VRPTW

Στο Πρόβλημα Δρομολόγησης Οχημάτων με Χρονικά Παράθυρα κάθε πελάτης πρέπει να εξυπηρετηθεί μέσα σε ένα χρονικό παράθυρο που έχει ορίσει ο ίδιος. Επιπλέον για κάθε πελάτη ενδεχομένως να υπάρχει κάποιος χρόνος που απαιτείται για την εξυπηρέτηση του. Τα οχήματα μπορεί και αυτά να λειτουργούν μόνο εντός ενός χρονικού παραθύρου.

MTVRP

Στο Πρόβλημα Δρομολόγησης Οχήματος με Πολλαπλά Ταξίδια ένα όχημα μπορεί να επιστρέψει στην αποθήκη και να ξαναφύγει. Συνήθως συνδυάζεται με χρονικά παράθυρα.

MCVRP

Στο Πρόβλημα Δρομολόγησης Οχημάτων με Διαμερίσματα, υπάρχουν πολλαπλοί τύποι προϊόντων που ζητάνε οι πελάτες. Κάθε όχημα χωρίζετε σε διαμερίσματα και κάθε διαμέρισμα μπορεί να έχει μόνο προϊόντα του ίδιου τύπου. Οι απαιτήσεις των πελατών έχουν πλέον τη μορφή ενός διανύσματος, ίδιας διάστασης με τον αριθμό των διαφορετικών τύπων προϊόντων. Προφανώς, δεν είναι απαραίτητο όλοι οι πελάτες να έχουν παραγγελία για κάθε τύπο (κάποιες από τις τιμές του διανύσματος μπορεί να είναι μηδενικές).

Μια ειδική περίπτωση αυτού είναι το Πρόβλημα Ανεφοδιασμού Πρατηρίου Βενζίνης (PSRP), με το οποίο ασχολούμαστε σε αυτή τη διπλωματική διατριβή. Στο Πρόβλημα Ανεφοδιασμού Πρατηρίου Βενζίνης διαφορετικές παραγγελίες πρέπει να τοποθετούνται και σε διαφορετικά διαμερίσματα. Έτσι θα μπορούσαμε να θεωρήσουμε ότι κάθε παραγγελία κάποιου πελάτη ορίζει ένα νέο τύπο προϊόντος. Όπως υπονοείται και από το όνομα του προβλήματος, οι περιορισμοί αυτοί συνηθίζονται στη μεταφορά προϊόντων πετρελαίου, αλλά συναντώνται επίσης και στον διαμοιρασμό άλλων προϊόντων.

Στο πρόβλημα που λύνουμε εμείς, ένας ετερογενής στόλος διαμερισματοποιημένων οχημάτων χρησιμοποιείται για την εξυπηρέτηση πελατών. Όλα τα οχήματα είναι αρχικά τοποθετημένα σε μια κεντρική αποθήκη εφοδίων, όπου και θα πρέπει να έχουν επιστρέψει στο τέλος της ημέρας. Κάθε όχημα μπορεί να κάνει μέχρι ένα ταξίδι. Κάθε πελάτης έχει ζητήση την παράδοση ενός αριθμού παραγγελιών διαφορετικού τύπου. Όλες οι παραγγελίες ενός πελάτη πρέπει να παραδοθούν από το ίδιο όχημα. Τα προϊόντα διαφορετικού τύπου δεν πρέπει να αναμειγνύονται μεταξύ τους και άρα τοποθετούνται σε διαφορετικά διαμερίσματα. Τα διαμερίσματα δεν είναι εξοπλισμένα με μετρητές ροής και επομένως πρέπει να αδειάσουν τελείως σε κάθε παράδοση. Γι' αυτόν τον λόγο διαφορετικές παραγγελίες δεν μπορούν να φορτωθούν στο ίδιο διαμέρισμα. Ο στόχος μας είναι να υπολογίσουμε τις διαδρομές που εξυπηρετούν όλους τους πελάτες και ελαχιστοποιούν τη συνολική απόσταση που διανύεται.

Tabu search

Η τοπική αναζήτηση είναι μια ευρετική μέθοδος που χρησιμοποιείται σε πολλά προβλήματα βελτιστοποιήσης. Οι αλγόριθμοι τοπικής αναζήτησης ξεκινούν από μία αρχική λύση και επαναληπτικά προσπαθούν να βελτιώσουν το κόστος της ερευνώντας της γειτονικές λύσης της τρέχουσας. Αυτή η διαδικασία συνεχίζεται μέχρι να ικανοποιηθεί το κριτήριο τερματισμού (π.χ. βρέθηκε κάποιο τοπικό βέλτιστο ή ξεπεράστηκε ένα χρονικό όριο).

Από τον παραπάνω ορισμό είναι ξεκάθαρο, ότι χρείαζεται να έχει οριστεί μια συνάρτηση γειτονίας στον χώρο αναζήτησης. Χρησιμοποιούμε τον ορισμό από [Aart03] για τη συνάρτηση γειτονίας.

Definition 0.0.1. Έστω $(S,\ f)$ ένα περιστατικό ενός προβλήματος συνδυαστικής βελτιστοποιήσης, όπου S είναι το σύνολο των εφικτών λύσεων, και f είναι η συνάρτηση κόστους ($f\ :\ S\ \rightarrow\ \mathbf{R}$). Η συνάρτηση γειτονίας είναι μια αντιστοίχιση $N : S \rightarrow 2^S$, που για κάθε λύση $i \in S$ ορίζει ένα σύνολο $N(i) \subseteq S$ λύσεων, οι οποίες βρίσκονται υπό μία έννοια κοντά στην i. Το σύνολο $N(i)$ είναι η γειτονιά της λύσης i, και κάθε $j \in N(i)$ είναι γειτονική της i.

Η επιλογή μιας καλής συνάρτησης γειτονίας έχει καθοριστική σημασία για την εύρεση μια καλής λύσης και είναι πάντοτε μία από τις προκλήσεις τις τοπικής αναζήτησης μιας και δεν υπάρχει κάποια μοναδική συνάρτηση γειτονιάς που θα μπορούσε να εφαρμοστεί σε όλα τα προβλήματα με καλά αποτελέσματα.

Η απλότερη μορφή της τοπικής αναζήτησης είναι ο αλγόριθμος αναρρίχησης λόφου. Σε αυτόν τον αλγόριθμο ξεκινάμε από μια αρχική υποψήφια λύση και σε κάθε επανάληψη μετακινούμαστε στη γειτονική λύση της υποψήφιας που βελτιώνει περισσότερο την αντικειμενική τιμή. Αν δεν υπάρχει κάποια γειτονική λύση η αναζήτηση σταματάει και επιστρέφεται η τελευταία λύση.

Αυτή η διαδικασία λειτουργεί καλά όταν ψάχνουμε την βέλτιστη λύση σε κυρτούς χώρους, αλλά υποφέρει σε μη κυρτούς λόγω της ύπαρξης τοπικών βέλτιστων. Στην προσπάθεια να αποφευχθούν τα τοπικά ελάχιστα έχουν αναπτυχθεί διάφορες μέθοδοι, μερικές εκ των οποίων βασίζονται στην επανεκκίνηση την αναζήτησης από διαφορετική αρχική λύση. Αν η νέα αρχική λύση, αγνοεί τις λύσης που έχουν ήδη εξερευνηθεί η μέθοδος λέγεται επαναλαμβανόμενη τοπική αναζήτηση (repeated local search). Στην πράξη, όμως είναι συχνό φαινόμενο δύο τοπικά ελάχιστα να μην βρίσκονται πολύ μακριά μεταξύ τους. Γι' αυτό το λόγο είναι συνήθως προτιμότερο η νέα αρχική λύση να προκύπτει μέσω της εφαρμογής ενός ταρακουνήματος στην καλύτερη λύση που έχει βρεθεί μέχρι στιγμής. Αυτή η μέθοδος λέγεται επαναληπτική τοπική αναζήτηση (iterated local search). Μια άλλη τεχνική, που έχει βρει μεγάλη επιτυχία σε πολλά πεδία και ειδικότερα στο VRP, βασίζεται στη χρήση δομών μνήμης και λέγεται tabu search. Αυτή είναι και η μέθοδος που χρησιμοποιήσαμε για τη λύση μας.

Στο tabu search [Glov98], αντίθετα με τις μεθόδους που περιγράψαμε νωρίτερα, λύσεις χειρότερες από την τρέχουσα μπορεί να γίνουν αποδεκτές προκειμένου να ξεφύγει ο αλγόριθμος από τοπικά βέλτιστα. Στο tabu search γίνεται χρήση μιας βραχυπρόθεσμης μνήμης, η οποία αποκαλείται tabu λίστα, και όπου αποθηκεύονται οι λύσεις που έχουν εξερευνηθεί πρόσφατα. Αυτές οι λύσεις απαγορεύεται να επιλεχθούν για μια περίοδο που λέγεται tabu tenure. Ο λόγος που κρατάμε αυτή τη λύστη είναι η αποφυγή κύκλων στις λύσεις που επισκεπτόμαστε και, κυρίως, η εξερεύνηση ανεξερεύνητων περιοχών του χώρου λύσεων.

Η αποθήκευση όλων των πρόσφατων λύσεων και η αναζήτηση σε αυτή τη μνήμη, μπορεί να αποτελέσει μια δύσκολη διαδικασία για πολλά προβλήματα. Για αυτόν τον λόγο, πολλές φορές κρατιούνται κάποια από τα χαρακτηριστικά των κινήσεων. Εκτός από την βραχυπρόθεσμη μνήμη που είναι η tabu λίστα, πολλές φορές χρησιμοποιούνται στρατηγικές εντατικοποίησης και διαφοροποιήσης, οι οποίες μπορεί να αναφερθούν και ως μεσοπρόθεσμη ή μακρυπρόθεσμη μνήμη αντίστοιχα. Οι κανόνες εντατικοποιήσης κατευθύνουν την αναζήτηση σε πιο υποσχόμενες περιοχές. Αυτό μπορεί να επιτευχθεί με διάφορους τρόπους όπως η απαγόρευση λύσεων με ορισμένα χαρακτηριστικά ή η προσαρμογή του μήκους της tabu λίστας. Οι κανόνες διαφοροποιήσης εφαρμόζονται συνήθως όταν η αναζήτηση έχει κολλήσει. Ένας τρόπος να πετύχουμε τη διαφοροποιήση είναι μέσω της επανεκκίνησης από κάποια τυχαία λύση.

Ο αλγόριθμος μας

Ο αλγόριθμος που αναπτύξαμε έχει την κλασσική δομή των tabu search αλγορίθμων όπως την αναλύσαμε παραπάνω. Στις επόμενες ενότητες παρουσιάζουμε πως υλοποιήσαμε τα εσωτερικά χαρακτηριστικά του αλγορίθμου.

Κατασκευή αρχικής λύσης

Για να πάρουμε την αρχική λύση χρησιμοποιούμε έναν απλό άπληστο αλγόριθμο που στοχεύει να τοποθετήσει τους κοντινούς κόμβους στην ίδια διαδρομή. Για το σκοπό αυτό, χρησιμοποιούμε ένα σύνολο αποτελούμενο από τους πελάτες που δεν έχουν δρομολογηθεί και τους αφαιρούμε έναν-έναν καθώς τους αναθέτουμε σε οχήματα. Όπως και οι περισσότεροι tabu search αλγόριθμοι, στηριζόμαστε στην τοπική αναζήτηση να βελτιώσει σημαντικά την αρχική λύση και δεν περιμένουμε αυτή να είναι κοντά στη βέλτιστη. Ο λόγος που θέλουμε καλές αρχικοποιήσεις είναι να μειώσουμε το χρόνο που χρειάζεται για να φτάσουμε σε καλές περιοχές του χώρου λύσεων.

Αρχικά κανένας πελάτης δεν έχει δρομολογηθεί, επομένως το σύνολο περιλαμβάνει όλους ρους πελάτες. Ξεκινάμε από το πρώτο όχημα. Προσπαθούμε επαναληπτικά να τοποθετήσουμε στο τέλος της διαδρομής του τρέχοντος οχήματος τον πελάτη που είναι πιο κοντά στον τελευταίο κόμβο της διαδρομής και οι απαιτήσεις του είναι ικανοποιήσιμες στο όχημα. Αν δεν υπάρχει κάποιος τέτοιος πελάτης, προχωράμε στο επόμενο όχημα. Εάν ξεμείνουμε από οχήματα, σημαίνει ότι ο αλγόριθμος απέτυχε να βρει μια λύση. Τελικά, σε περίπτωση που βρέθηκε μια λύση, όλες οι διαδρομές που δημιούργησε ο αλγόριθμος βελτιστοποιούνται κάνοντας χρήση μιας τεχνικής βελτιστοποιήσης που θα περιγράψουμε σε επόμενη ενότητα.

Εφικτότητα

Για να διαπιστώσουμε εάν μια διαδρομή είναι εφικτή χρησιμοποιούμε μια ευρετική μέθοδο και αν αυτή αποτύχει να μας δώσει μια σίγουρη απάντηση χρησιμοποιούμε έναν επιλυτή μεικτού ακέραιου προγραμματισμού. Η ευρετική μέθοδος είναι η ακόλουθη: Αρχικά αποθηκεύουμε σε έναν πίνακα τις παραγγελίες των πελατών, ενώ κρατάμε επίσης ένα πολυσύνολο με τα μεγέθη των διαμερισμάτων. Εάν το πλήθος των παραγγελιών είναι μεγαλύτερο από το πλήθος των διαμερισμάτος, ή αν το

άθροισμα των απαιτήσεων είνι μεγαλύτερο από το άθροισμα των χωρητικοτήτων των διαμερισμάτων αποκρινόμαστε με σιγουριά ότι η διαδρομή δεν είναι εφικτή. Σε διαφορετική περίπτωση ακολουθούμε την επόμενη διαδικασία για κάθε παραγγελία. Βρίσκουμε το μικρότερο διαθέσιμο διαμέρισμα που να είναι αρκετά μεγάλο ώστε να χωράει την παραγγελία και την τοποθετούμε σε αυτό. Εάν δεν υπάρχει τέτοιο διαμέρισμα, τοποθετούμε στο μεγαλύτερο διαμέρισμα ένα τμήμα της παραγγελίας που είναι ίσο με το μέγεθος του διαμερίσματος και επαναλαμβάνουμε τη διαδικασία για την επόμενη παραγγελία. Αν έχουμε τοποθετήσει επιτυχώς όλες τις παραγγελίες σε διαμερίσματα, αποκρινόμαστε ότι η διαδρομή είναι εφικτή. Αν αντιθέτως, ξεμείνουμε από διαμερίσματα, χρησιμοποιούμε τον επιλυτή μεικτού ακέραιου προγραμματισμού για να απαντήσουμε με σιγουριά.

Το ακέραιο πρόγραμμα που λύνουμε εάν η ευρετική μέθοδος αποτύχει να δώσει ξεκάθαρη απάντηση βρίσκεται ακολούθως. Οι μεταβλητές y_{ij} αποφασίζουν εάν η παραγγελία i πρέπει να τοποθετηθεί στο διαμέρισμα j. Έστω επίσης ότι είναι το σύνολο των παραγγελιών και C το σύνολο των διαμερισμάτων, με q_i την ποσότητα της παραγγελίας i και d_j την χωρητικότητα του διαμερίσματος j.

$$\min \quad \sum_{i \in O} \sum_{j \in C} y_{ij} \tag{0.5}$$

$$\text{s.t} \quad q_i \leq \sum_{j \in C} y_{ij} \times d_j \quad \forall i \in O \tag{0.6}$$

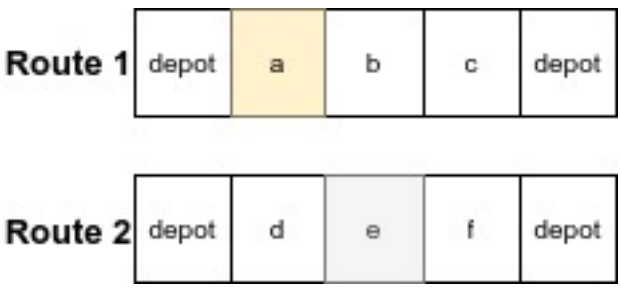
$$\sum_{i \in O} y_i j \leq 1 \quad \forall j \in C \tag{0.7}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i, j) \tag{0.8}$$

όπου η αντικειμενική συνάρτηση (0.5) εκφράζει την ελαχιστοποιήση των χρησιμοποιούμενων διαμερισμάτων. Οι περιορισμοί (0.6) εξασφαλίζουν ότι τα διαμερίσματα που έχουν ανατεθεί για κάθε παραγγελία έχουν επαρκή χωρητικότητα. Οι περιορισμοί (0.7) σιγουρεύουν ότι κάθε διαμέρισμα χρησιμοποιείται το πολύ μία φορά. Τέλος οι περιορισμοί (0.8) ορίζουν ότι οι μεταβλητές y_{ij} είναι δυαδικές.

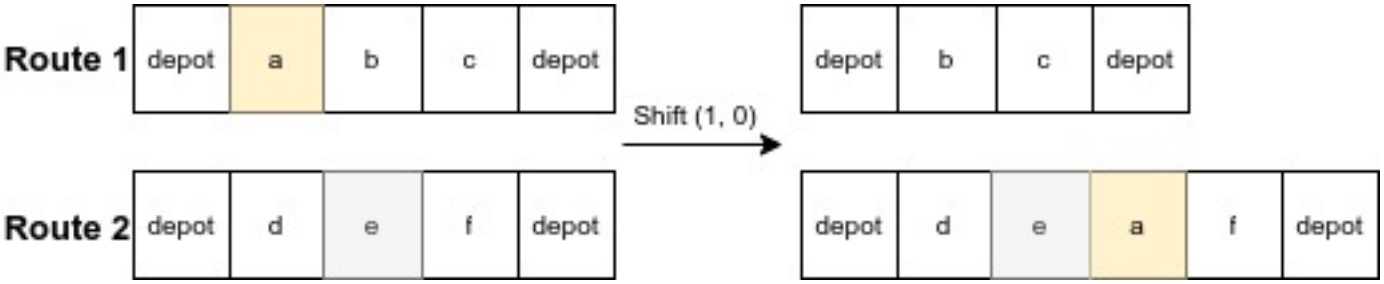
Δομή γειτονιάς

Η πλήρης λίστα των γειτονικών κινήσεων που χρησιμοποιούμε βρίσκεται παρακάτω. Συμπεριλαμβάνουμε επίσης το αποτέλεσμα κάθε κίνησης εάν αυτή εφαρμοστεί με ορίσματα τους χρωματισμένους πελάτες των διαδρομών του σχήματος 0.1.



Σχήμα 0.1: Αρχικές διαδρομές και επιλεγμένοι πελάτες

Shift (1, 0)



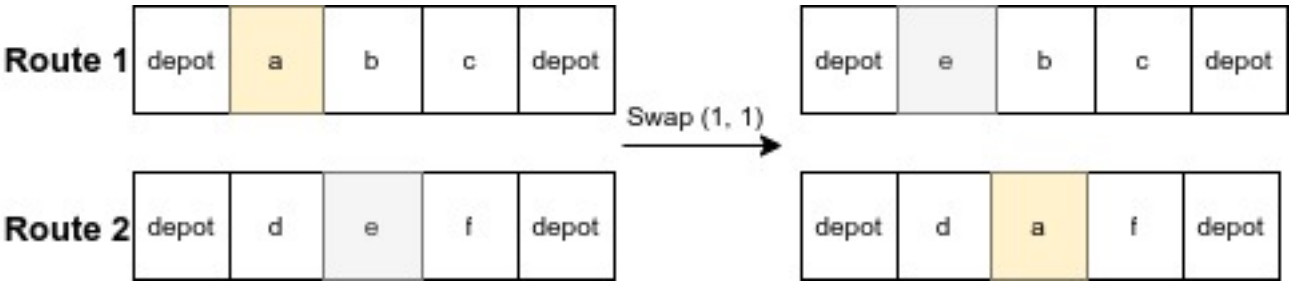
Σχήμα 0.2: Shift (1, 0)

Με την εφαρμογή της shift (1, 0) ο κόμβος που βρίσκεται στη θέση i_1 της διαδρομής r_1 μετακινείται στη θέση $i_2 + 1$ της διαδρομής r_2, όπως φαίνεται και στο σχήμα 0.2. Δεν επιτρέπουμε σε έναν κόμβο που είναι μόνος του στη διαδρομή να μετακινηθεί σε μία κενή διαδρομή.

Στο παραπάνω παράδειγμα, το κόστος c' της δεξιάς λύσης υπολογίζεται ως εξής (υποθέτοντας ότι το κόστος της αριστερής λύσης είναι c):

$$c' = c - distance(depot, a) - distance(a, b) + distance(e, a) + distance(a, f)$$

Swap (1, 1)



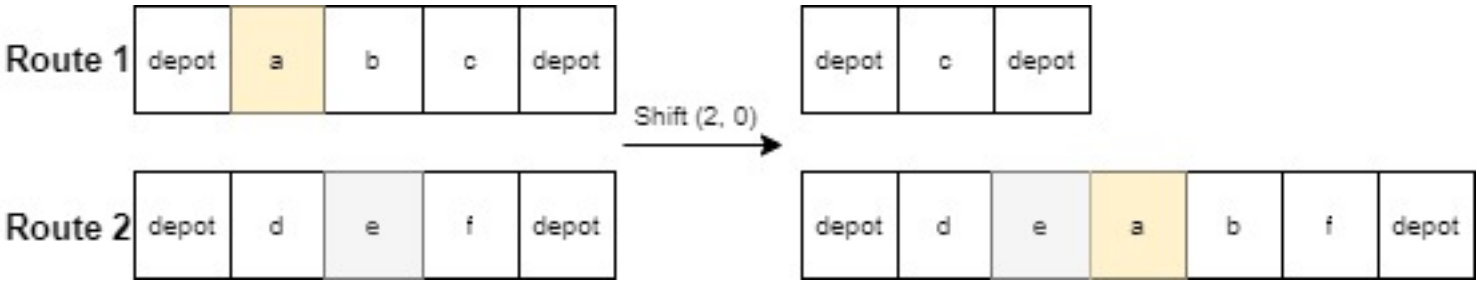
Σχήμα 0.3: Swap (1, 1)

Η κίνηση swap (1, 1) ανταλλάσει τη θέση i_1-οστού κόμβου της διαδρομής r_1 με τον i_2-οστό κόμβο της διαδρομής r_2, όπως φαίνεται στο σχήμα 0.3. Δεν επιτρέπουμε την εφαρμογή αυτής της κίνησης όταν οι δύο κόμβοι είναι οι μόνοι πελάτες στη διαδρομή τους.

Στο παραπάνω παράδειγμα, το κόστος c' της δεξιάς λύσης υπολογίζεται ως εξής (υποθέτοντας ότι το κόστος της αριστερής λύσης είναι c):

$$\begin{aligned} c' = c &- distance(depot, a) - distance(a, b) + distance(d, a) \\ &+ distance(a, f) - distance(d, e) - distance(e, f) \\ &+ distance(depot, e) + distance(e, b) \end{aligned}$$

Shift (2, 0)



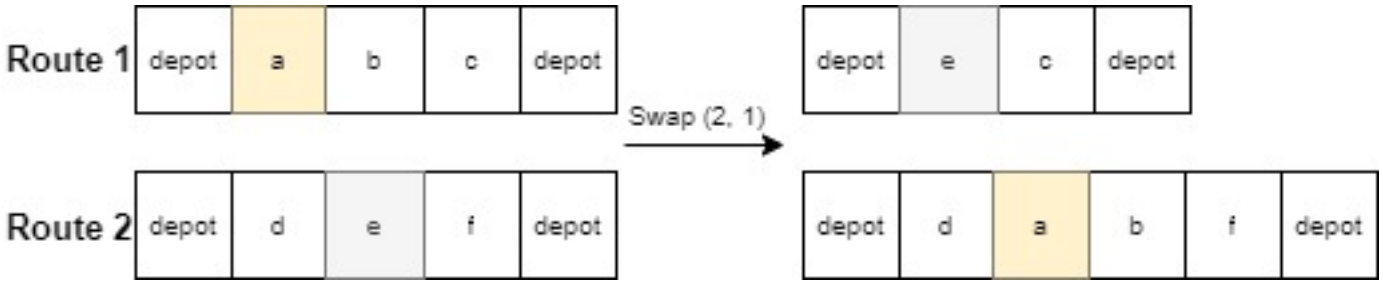
Σχήμα 0.4: Shift (2, 0)

Η κίνηση shift (2, 0) μετακινεί τον κόμβο i_1 της διαδρομής r_1 μαζί με τον επόμενο κόμβο στη θέση μετά τον i_2-οστό κόμβο της διαδρομής r_2, όπως φαίνεται και στο σχήμα 0.4. Δεν επιτρέπουμε 2 πελάτες που είναι η μοναδική μιας διαδρομής να μετακινηθούν σε μια κενή διαδρομή.

Στο παραπάνω παράδειγμα, το κόστος c' της δεξιάς λύσης υπολογίζεται ως εξής (υποθέτοντας ότι το κόστος της αριστερής λύσης είναι c):

$$c' = c - distance(depot, a) - distance(b, c) + distance(e, a) + distance(b, f)$$

Swap (2, 1)



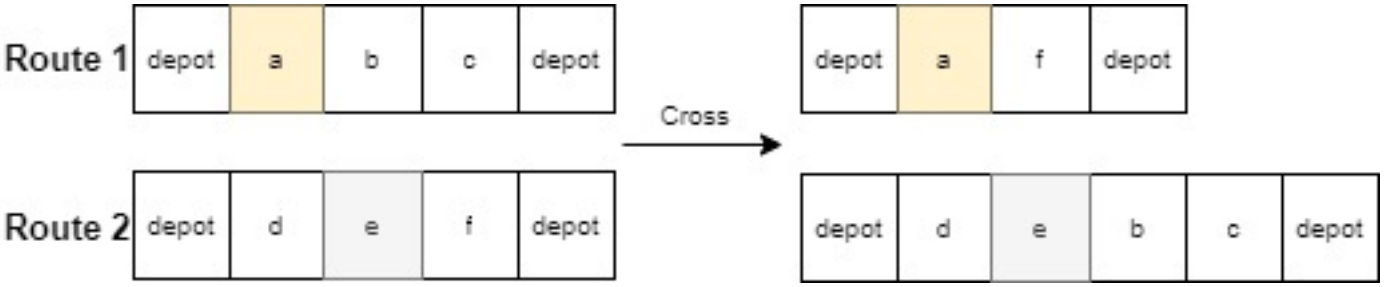
Σχήμα 0.5: Swap (2, 1)

Η κίνηση swap (1, 1) ανταλλάσει τον i_1-οστό κόμβο της διαδρομής r_1 με τον i_2-οστό κόμβο της διαδρομής r_2. Ο κόμβος στη θέση $(i_1 + 1)$ της διαδρομής r_1 μετακινείται μαζί με τον i_1th. Η κίνηση φαίνεται και στο σχήμα 0.5. Δεν επιτρέπουμε σε 2 πελάτες που είναι οι μοναδικοί της διαδρομής τους να ανταλλαχθούν με έναν που είναι μοναδικός στη δική του.

Στο παραπάνω παράδειγμα, το κόστος c' της δεξιάς λύσης υπολογίζεται ως εξής (υποθέτοντας ότι το κόστος της αριστερής λύσης είναι c):

$$\begin{aligned} c' = c &- distance(depot, a) - distance(b, c) + distance(d, a) \\ &+ distance(b, f) - distance(d, e) - distance(e, f) \\ &+ distance(depot, e) + distance(e, c) \end{aligned}$$

Cross



Σχήμα 0.6: Cross

Οι διαδρομές r_1 και r_2, παραμένουν ίδιες μέχρι τον i_1-οστό και i_2-οστό πελάτη, αντίστοιχα, και στη συνέχεια διασταυρώνονται. Αυτό απεικονίζεται και στο σχήμα 0.6. Δεν επιτρέπουμε οι θέσεις i_1 και i_2 να είναι ταυτόχρονα προτελευταίες στις διαδρομές τους αφού τότε μόνο η αποθήκες θα διασταυρώνονταν.

Στο παραπάνω παράδειγμα, το κόστος c' της δεξιάς λύσης υπολογίζεται ως εξής (υποθέτοντας ότι το κόστος της αριστερής λύσης είναι c):

$$c' = c - distance(a, b) - distance(e, f) + distance(a, f) + distance(e, b)$$

Tabu λίστα

Υλοποιούμε την tabu λίστα σαν έναν τετραγωνικό πίνακα διαστάσεων $n \times n$, όπου n είναι είναι το πλήθος των κόμβων, συμπεριλαμβανομένης της αποθήκης. Μια θετική τιμή στο κελί (i, j) υποδεικνύει ότι κάθε κίνηση που συνδέει τους κόμβους i και j θεωρείται tabu και επομένως απαγορεύεται. Αν αυτή η τιμή είναι ίση με t τότε ο περιορισμός αυτός ισχύει για τις επόμενες t επαναλήψεις.

Στην tabu λίστα τοποθετούμε τις ακμές που αφαιρούνται σε κάθε κίνηση. Για παράδειγμα, στην κίνηση του σχήματος 4.5 τα κελιά που θα αποκτήσουν θετική τιμή είναι τα $(depot, a)$, (b, c), (d, e) and (e, f). Εάν κάποιο από τα κελιά $(depot, e)$, (e, c), (d, a) και/ή (b, f) είχαν θετική τιμή αυτή η κίνηση θα απαγορευόταν, καθώς θα ξαναένωνε κόμβους που είχαν διασπαστεί πρόσφατα.

Η ενημέρωσης της tabu λίστας αποτελείται από δύο φάσεις. Πρώτον, όλες οι θετικές τιμές μειώνονται κατά 1, αφού έχει περάσει μία επανάληψη. Έπειτα, όλες οι ακμές που αφαιρέθηκαν λόγω της επιλεχθείσας κίνησης προστίθενται στην λίστα με τιμή ίση με μία περίοδο. Η περίοδος υπολογίζεται ξεχωριστά για κάθε ακμή και είναι το άθροισμα δύο όρων: ο πρώτος όρος είναι μια σταθερά που χρησιμοποιείται σαν κάτω όριο και ο δεύτερος είναι ένας τυχαίος αριθμός στο διάστημα $(0, UB)$.

Εσωτερικές βελτιστοποιήσεις

Μιας και το πλήθος των πελατών φράσσεται από το πλήθος των διαμερισμάτων, που συνήθως δεν ξεπερνούν τα 10, αποφασίσαμε να βελτιστοποιήσουμε κάθε διαδρομή που αλλάζει, λύνοντας το Πρόβλημα του Πλανόδιου Πωλητή που ορίζεται από τους κόμβους της. Για αυτόν τον σκοπό χρησιμοποιήσαμε τον αλγόριθμο δυναμικού προγραμματισμού Held-Karp [Bell62, Held62] που έχει χρονική πολυπλοκότητα $O(n^2 2^n)$. Αυτή η πολυπλοκότητα μπορεί να μοιάζει απαγορευτική, αφού είναι εκθετική, αλλά δεν αποτελεί πρόβλημα για περιπτώσεις με έως 10 κόμβους. Αν κάποια διαδρομή έχει περισσότερους από 12 κόμβους, δεν κάνουμε κάποια βελτιστοποίηση.

Η αναδρομική σχέση που χρησιμοποιούμε για την επίλυση του προβλήματος βρίσκεται στην εξίσωση 0.9. Χρησιμοποιούμε τον πίνακα DP για την απομνημόνευση των κοστών των μερικών λύσεων. Το κελί $DP[S, k]$ εκπροσωπεί το βέλτιστο κόστος του μονοπατιού Hamilton που περιλαβάνει τους κόμβους $S \cup 1$ αν ο κόμβος k είναι τελευταίος και ο 1 είναι πρώτος. Το συνολικό πλήθος αυτών των συνόλων είναι 2^{n-1} κι επομένως $O(2^{n-1}n) = O(n2^n)$ χώρος απαιτείται.

$$DP[S,k] = \begin{cases} distance(1,k), & S = \{k\} \\ \min_j DP[S \setminus k, j] + distance(j,k), & otherwise \end{cases} \tag{0.9}$$

Το κόστος της ελάχιστης λύσης υπολογίζεται έπειτα ως εξής:

$$minCost = \min_j DP[\{2,\ldots,n\}, j] + distance(j,1)$$

Εντατικοποιήση και διαφοροποιήση

Η πρώτη τεχνική διαφοροποιήσης που χρησιμοποιούμε είναι η τυχαίοτητα στην διάρκεια που τοποθετείται κάθε στοιχείο στην tabu λίστα. Αυτό μας οδηγεί στην εξερεύνηση του χώρου αναζήτησης με έναν μη γραμμικό τρόπο και συνεπώς στην εξερεύνηση περιοχών που δε θα επισκεπτόμασταν υπό κανονικές συνθήκες.

Μια άλλη τεχνική που χρησιμοποιούμε είναι ότι όταν βρίσκουμε μια κίνηση που βελτιώνει το κόστος αλλά είναι ανέφικτη, εξετάζουμε εάν τα ελεύθερα οχήματα μπορούν να ικανοποιήσουν τις ανέφικτες διαδρομές. Με αυτόν τον τρόπο εντατικοποιούμε την αναζήτηση σε καλές περιοχές.

Η τρίτη τεχνική εντατικοποίησης και διαφοροποιήσης που χρησιμοποιούμε είναι η επανεκκίνηση από μια τυχαία λύση στην περιοχής της καλύτερης που έχουμε βρει μέχρι στιγμής, όταν η τοπική αναζήτηση έχει κολλήσει. Αυτό το επιτυγχάνουμε μέσω της επαναφοράς της καλύτερης λύσης και την επιλογή μιας τυχαίας κίνησης από αυτές που περιγράφηκαν προηγουμένως. Φυσικά, υπάρχει μεγάλη πιθανότητα η κίνηση που επιλέχθηκε να είναι ανέφικτη. Τότε επαναλμβάνουμε μέχρι να βρεθεί μια εφικτή κίνηση. Αυτό μας βοηθάει τόσο να εντατικοποιήσουμε την αναζήτηση γύρω στην καλύτερη λύση και να την διαφοροποιήσουμε επιλέγοντας μια κίνηση με μη ντετερμινιστικό τρόπο.

Πειραματικά αποτελέσματα

Για την αξιολόγηση της δουλειάς μας χρησιμοποιήσαμε τρία σύνολα παραδειγμάτων: 1) το σύνολο A του Augerat [Auge95], 2) το σύνολο X του Uchoa [Ucho17] και 3) πραγματικά δεδομένα από έναν διακινητή πετρελαίου. Ο κώδικας του αλγόριθμου μας γράφτηκες σε Java. Τα πειράματα έτρεξαν σε έναν επεξεργαστή Intel Core i5-8250U με συχνότητα 1.6 GHz και 8 GB RAM. Το λειτουργικό σύστημα ήταν Ubunut 20.04. Σαν επιλυτή μεικτού ακέραιου προγραμματισμού χρησιμοποιήσαμε το CPLEX 20.1. Ο κώδικα που χρησιμοποιήσαμε βρίσκεται στο `https://github.com/jasonlazar/MCVRP`.

Αξιολόγηση στο σύνολο A

Το σύνολο A του Augerat, περιέχει παραδείγματα με 32 εώς 81 κόμβους. Ο στόλος θεωρείται ομοιογενής και απεριόριστος, με κάθε όχημα να έχει χωρητικότητα 100. Ο αλγόριθμος μας χειρίζεται καθορισμένους στόλους, αλλά μπορούμε να προσομοιώσουμε έναν άπειρο στόλο χρησιμοποιώντας όσα οχήματα όσους και πελάτες. Στην πράξη αποκλείεται ποτέ να χρειαστούν τόσα οχήματα, οπότε χρησιμοποιήσαμε $n/3$ οχήματα. Στο σύνολο αυτό η βέλτιστη λύση ποτέ δεν χρησιμοποιεί πάνω από $n/6$ οχήματα. Το σύνολο αυτό δεν περιέχει διαμερίσματα, οπότε εξετάσαμε πως συμπεριφέρεται ο

Παράδειγμα	Βέλτιστο	Δικό μας	Λόγος Προσέγγισης
A-n32-k5	**784**	**784**	1.00
A-n33-k5	**661**	**661**	1.00
A-n33-k6	**742**	743	1.00
A-n34-k5	**778**	787	1.01
A-n36-k5	**799**	**799**	1.00
A-n37-k5	**669**	678	1.01
A-n37-k6	**949**	960	1.01
A-n38-k5	**730**	743	1.01
A-n39-k5	**822**	825	1.00
A-n39-k6	**831**	833	1.00
A-n44-k7	**937**	976	1.04
A-n45-k6	**944**	955	1.01
A-n45-k7	**1146**	1179	1.03
A-n46-k7	**914**	927	1.01
A-n48-k7	**1073**	1116	1.04
A-n53-k7	**1010**	1064	1.05
A-n54-k7	**1167**	1168	1.00
A-n55-k9	**1073**	1112	1.04
A-n60-k9	**1354**	1368	1.01
A-n61-k9	**1034**	1076	1.04
A-n62-k8	**1288**	1328	1.03
A-n63-k9	**1616**	1632	1.01
A-n63-k10	**1314**	1357	1.03
A-n64-k9	**1401**	1425	1.02
A-n65-k9	**1174**	1187	1.01
A-n69-k9	**1159**	1170	1.01
A-n80-k10	**1763**	1848	1.05

Πίνακας 0.1: Αξιολόγηση στο σύνολο A

αλγόριθμος μας στο κλασσικό VRP.

Για τα πειράματα μας στρογγυλοποιήσαμε τις αποστάσεις μεταξύ των κόμβων στον κοντινότερο ακέραιο. Αυτό συμβαίνει και στις βέλτιστες λύσεις, οπότε μας βοηθάει να συγκρίνουμε πιο εύκολα τις δύο λύσεις. Όσον αφορά τις παραμέτρους, σταματάμε την τοπική αναζήτηση μετά από 10 επαναλήψεις χωρίς βελτίωση του κόστους και κάνουμε 5 επανεκκινήσεις. Η διάρκεια παραμονής στην tabu λίστα έχει σαν σταθερό όρο το 7 στα παραδείγματα που έχουν μέχρι 45 κόμβους και 10 στα υπόλοιπα. Ο τυχαίος όρος είναι μέχρι 5 σε όλα τα παραδείγματα.

Όπως είναι προφανές από τον πίνακα 0.1 οι λύσεις μας είναι αρκετά κοντά στις βέλτιστες. Ο γεωμετρικός όρος του λόγου προσέγγισης είναι κάτω από 2%, το οποίο σημαίνει ότι στην τυπική περίπτωση η λύση μας είναι λιγότερο από 2% χειρότερη απο τη βέλτιστη. Επίσης, σε 3 παραδείγματα καταφέραμε να βρούμε τη βέλτιστη λύση.

Αξιολόγηση στο σύνολο X

Σαν ένα δεύτερο τρόπο αξιολόγησης τροποποίησαμε το σύνολο X του Uchoa, το οποίο είναι επίσης ένα σύνολο παραδειγμάτων που χρησιμοποιούνται ως σημείο αναφοράς για το κλασσικό VRP, προσθέτοντας διαμερίσματα στα οχήματα. Χρησιμοποιήσαμε πέντε διαφορετικές κατηγορίες διαμερισμάτων, προκειμένου να κάνουμε το στόλο ετερογενή όπως συμβαίνει και στις πραγματικές εφαρ-

μογές. Σε αυτές τις περιπτώσεις, η απευθείας σύγκριση των λύσεων μας με τις βέλτιστες χωρίς διαμερίσματα δεν θα είχε νόημα, μιας και εμείς λύνουμε ένα πολύ πιο δύσκολο πρόβλημα. Ο σκοπός μας σε αυτήν την αξιολόγηση είναι να δούμε πόσο καλά ο αλγόριθμος μας φορτώνει τα οχήματα.

Στη βιομηχανία οι φορτώσεις που θεωρούνται αποδεκτές είναι αυτές που γεμίζουν πάνω από 60% του οχήματος. Δεν θα παρουσιάσουμε πως φορτώνεται κάθε όχημα, καθώς έχουμε πολλά παραδείγματα καθένα εκ των οποίων χρησιμοποιεί πολλά οχήματα. Αντί αυτού παρουσιάζουμε τον επόμενο πίνακα όπου για κάθε παράδειγμα παρουσιάζει τη μέση φόρτωση κάθε οχήματος.

Παράδειγμα	Μέση φόρτωση %
X-n101-k25	81
X-n106-k14	82
X-n110-k13	76
X-n115-k10	50
X-n125-k30	82
X-n129-k18	86
X-n134-k13	66
X-n139-k10	64
X-n148-k46	92
X-n153-k22	67
X-n157-k13	88
X-n162-k11	55
X-n172-k51	86
X-n176-k26	75
X-n181-k23	97
X-n186-k15	57
X-n195-k51	88
X-n200-k36	82
X-n204-k19	69
X-n209-k16	64
X-n219-k73	99
X-n223-k34	89
X-n228-k23	66
X-n233-k16	57
X-n242-k48	94
X-n247-k50	77
X-n251-k28	81
X-n266-k58	93
X-n270-k35	78
X-n275-k28	91
X-n289-k60	85
X-n294-k50	85
X-n298-k31	81

Πίνακας 0.2: Μέση φόρτωση οχήματος ανά παράδειγμα

Τα αποτελέσματα του πίνακα 0.2 αποδεικνύουν ότι ο αλγόριθμος μας παράγει πολύ καλές φορτώσεις για τα οχήματα. Όπως είπαμε και πριν ένα όχημα που γεμίζει πάνω από 60% θεωρείται αποδεκτό. Ο αλγόριθμος μας βρίσκει εξαιρετικές φορτώσεις (πάνω από 80%) για το μέσο όχημα σε πάνω από το 55% των περιπτώσεων. Ο αριθμός αυτός ανεβαίνει στο 67% αν συμπεριλάβουμε και τις φορτώσεις από 75% και πάνω. Παρατηρούμε ότι όλα τα υπόλοιπα, εκτός από 2, έχουν λόγο n/k πάνω από 10 και

θα ήταν αδύνατο να φορτωθούν τόσο καλά από τη στιγμή που χρησιμοποιούμε έως 10 διαμερίσματα. Επομένως σε 22 από τα 24 παραδείγματα (92%) με λόγο n/k μικρότερο από 10 επιτυγχάνουμε μέση φόρτωση πάνω από 75%.

Αξιολόγηση σε πραγματικά δεδομένα

Σε αυτή την υποενότητα εξετάζουμε τον αλγόριθμο μας σε ένα σύνολο από πραγματικά παραδείγματα που προμηθευτήκαμε από έναν διακινητή πετρελαίου που ενεργεί στην Ελλάδα. Στην πραγματικότητα, αυτοί έχουν να κάνουν με περισσότερους περιορισμούς, όπως χρονικά παράθυρα και έξτρα περιορισμούς στο πως φορτώνονται τα διαμερίσματα. Επιπλέον, επιτρέπουν πολλαπλά ταξίδια ανά όχημα. Για την αξιολόγηση μας κρατήσαμε μόνο τους περιορισμούς που σχετίζονται με το πρόβλημα μας. Αυτό που μας δυσκόλεψε περισσότερο να αφαιρέσουμε ήταν τα πολλαπλά ταξίδια, μιας και ο αλγόριθμος μας απέτυχε να παράξει την αρχική λύση σε κάποιες περιπτώσεις. Παρ' όλα αυτά, καταφέραμε να πάρουμε αποτελέσματα για πέντε παραδείγματα, τα οποία και θα παρουσιάσουμε.

Συγκρίνουμε τα αποτελέσματα μας με αυτά που παρήγαγε ένας αλγόριθμος, που είχε αναπτυχθεί από μια προηγούμενη ομάδα έρευνας για την εταιρεία. Εκείνος ο αλγόριθμος, χρησιμοποιούσε τα OR-Tools της Google για να αναθέσει τους πελάτες σε οχήματα. Επειδή τα OR-Tools δεν υποστηρίζουν διαμερίσματα, προσπάθησαν να τα προσομοιώσουν χρησιμοποιώντας κάποιους ασθενέστερους περιορισμούς. Αυτό έχει σαν αποτέλεσμα οι περισσότερες από τις διαδρομές που παράγει ο αλγόριθμος να είναι ανέφικτες στα οχήματα τους. Για να ξεπεράσουν αυτό το εμπόδιο, χρησιμοποιήσαν μια τεχνική ταιριάσματος (matching) των διαδρομών σε εφικτά οχήματα. Στο ταίριασμα οι διαδρομές και τα οχήματα παριστάνονται σαν κορυφές ενός διμερούς γραφήματος. Μια ακμή (r, v) στο γράφημα συμβολίζει ότι η διαδρομή r είναι εφικτή στο όχημα v. Ο στόχος είναι να βρεθεί ο μέγιστος αριθμός ακμών χωρίς κοινές κορυφές. Αυτό βοήθησε να αντιστοιχιστούν αρκετές διαδρομές, αλλά σχεδόν σε όλες τις περιπτώσεις παρέμειναν ορισμένες ανέφικτες.

Επειδή, ο αλγόριθμος τους παράγει ανέφικτες διαδρομές, τις οποίες μάλιστα αφερεί μετά το ταίριασμα, περιμένουμε ο αλγόριθμος τους να έχει χαμηλότερα κόστη. Στον πίνακα 0.3 παρουσιάζουμε τα κόστη που βρίσκει ο αλγόριθμος μας και ο αλγόριθμος τους πριν και μετά το ταίριασμα. Επίσης συμπεριλαμβάνουμε πόσο μεγαλύτερο είναι το δικό μας κόστος σε κάθε περίπτωση. Ακόμα έχουμε το ποσοστό των διαδρομών που υπολογίζει ο αλγόριθμος τους και είναι εφικτές μετά το matching. Τέλος, συμπεριλαμβάνουμε τους χρόνους που χρειάστηκαν οι δύο αλγόριθμοι για να υπολογίσουν τη λύση.

Παράδειγμα	Tabu search	Χρόνος	OR-Tools	Διαφορά%	Matching	Διαφορά%	Εφικτές%	Χρόνος
1	10115085	14m59s	9803947	3.2	9036121	11.9	90.9	17m40s
2	10399898	23m15s	9984231	4.2	9053257	14.9	92.2	15m3s
3	7550051	13m41s	7336853	2.9	7002968	7.8	93.8	16m35s
4	5214640	15m36s	5989852	-12.9	5245078	0.0	90.8	17m22s
5	5120051	1m20s	5095335	0.0	5095335	0.0	100.0	1m40s

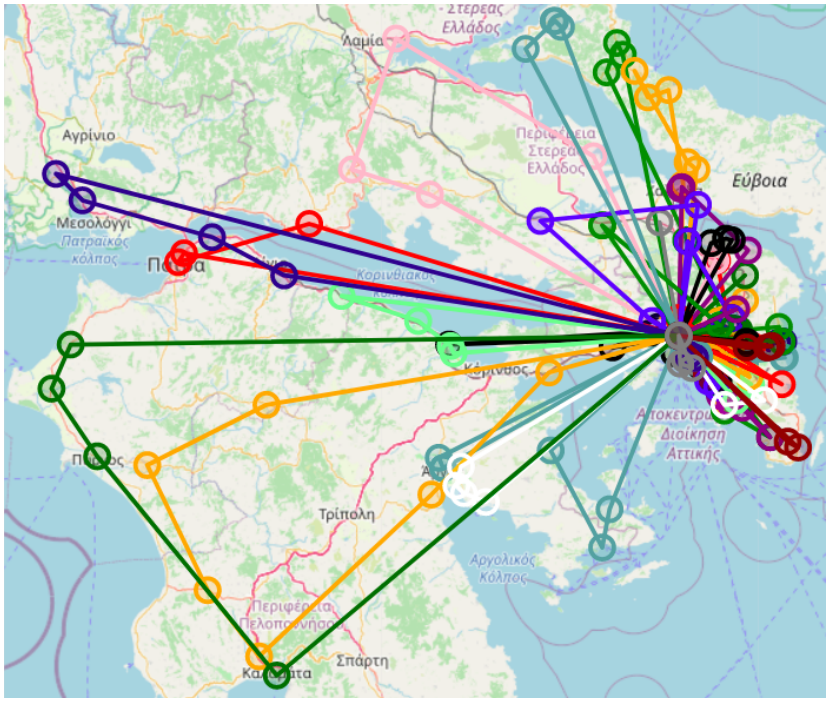
Πίνακας 0.3: Αξιλόγηση σε πραγματικά δεδομένα

Παρά το γεγονός ότι ο αλγόριθμος μας εξετάζει μόνο εφικτές διαδρομές σε καμία περίπτωση δεν έχουμε πάνω από 4.2% χειρότερο κόστος από ότι τα OR-Tools, που έχουν ασθενέστερους περιορισμούς και συμπεριλαμβάνουν ανέφικτες διαδρομές. Ο OR-Tools/Matching αλγόριθμος καταφέρνει μόνο σε ένα περιστατικό να παράγει εφικτές διαδρομές, το οποίο είναι μια ειδική περίπτωση μικρότερης διάστασης με μόνο επαρχιακούς πελάτες. Όσον αφορά την αξιολόγηση της ευρετικής μεθόδου που χρησιμοποιούμε για τη φόρτωση βρήκαμε ότι σε καμία περίπτωση δεν χρειαστήκαμε το CPLEX

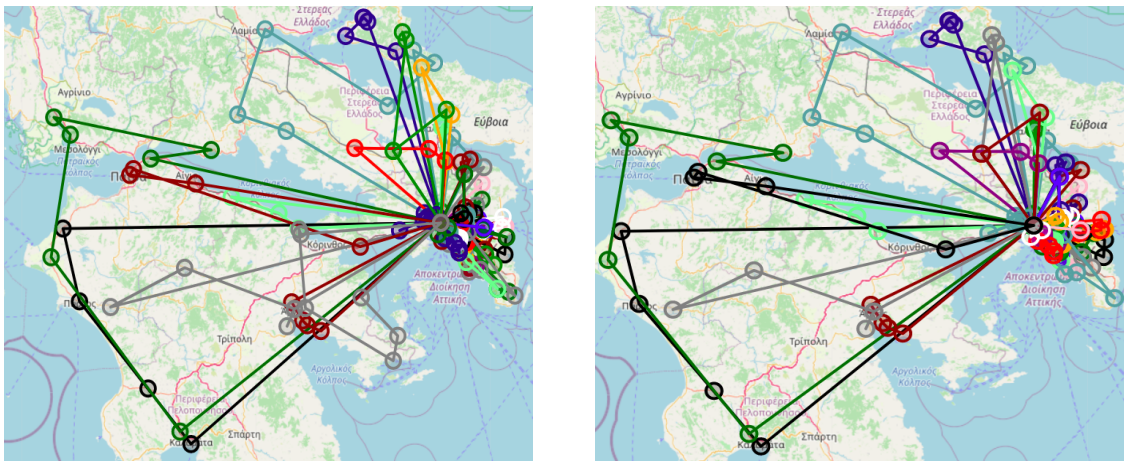
για πάνω από το 0.3% των περιπτώσεων.

Η πιο αξιοσημείωτη περίπτωση από τις παραπάνω είναι σίγουρα το παράδειγμα 4, στο οποίο είμαστε 12.9% καλύτεροι, παρά το γεγονός ότι μόλις το 91% των διαδρομών τους είναι εφικτές. Ακόμα και να αφαιρέσουμε όλες τις ανέφικτες διαδρομές, ο αλγόριθμος μας εξακολουθεί να έχει ελαφρώς μικρότερο κόστος. Στις επόμενες σελίδες, θα παρουσιάσουμε ενδεικτικά τις διαδρομές που υπολογίζει ο αλγόριθμος μας στα παραδείγματα 2 και 4 και θα εξηγήσουμε τα δυνατά σημεία του.

Παράδειγμα 2



Σχήμα 0.7: Διαδρομές που βρήκε ο αλγόριθμος μας στο παράδειγμα 2

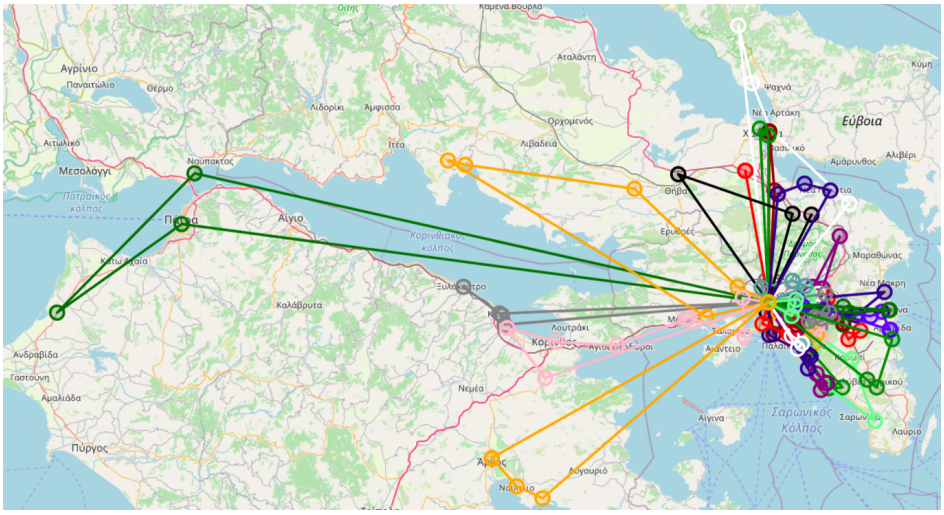


(a) Πριν το ταίριασμα (b) Μετά το ταίριασμα

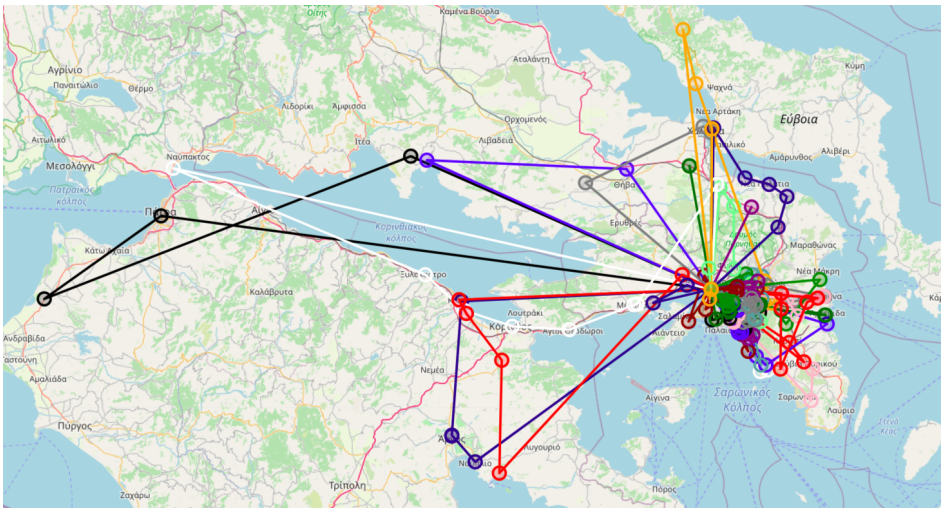
Σχήμα 0.8: Διαδρομές που βρήκαν τα OR-Tools στο παράδειγμα 2

Σε αυτό το παράδειγμα, τα OR-Tools έχουν βρει ένα υψηλό ποσοστό διαδρομών (7.8%) που δεν έχει μπορέσει να αντιστοιχιστεί σε κάποιο όχημα. Μια μεγάλη επαρχιακή διαδρομή που περνάει από την Επίδαυρο και το Κιάτο είναι ανέφικτη, κάτι που έχει μειώσει σημαντικά το συνολικό κόστος που υπολογίζουν. Πέραν αυτού, οαλγόριθμος μας πιθανότατα θα έπρεπε να έχει χωρίσει κάπως διαφορετικά τους πελάτες της πράσινης και πορτοκαλί διαδρομής της Πελοποννήσου και να απέφευγε τη μεγάλη ακμή μεταξύ Καλαμάτας και Πύργου. Παρατηρούμε, όμως, ότι το ίδιο πρόβλημα υπάρχει και στις διαδρομές των OR-Tools.

Παράδειγμα 4



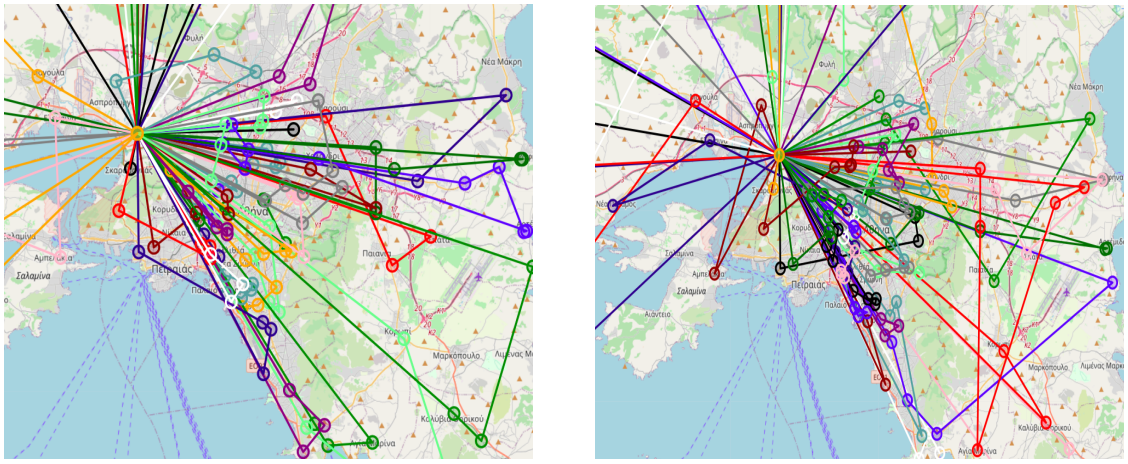
Σχήμα 0.9: Διαδρομές που βρήκε ο αλγόριθμος μας στο παράδειγμα 4



Σχήμα 0.10: Διαδρομές που βρήκαν τα OR-Tools στο παράδειγμα 4

Από τα σχήματα 0.9 και 0.10 είναι αυταπόδεικτο ότι ο tabu search αλγόριθμος έχει δρομολογήσει τους επαρχιακούς πελάτες πιο αποδοτικά αφού αποφεύγει να ταξιδέψει στη Δύση δύο φορές, κάτι που έχει σαν αποτέλεσμα να διανυθεί πολύ λιγότερη απόσταση συνολικά. Αυτό είναι εντυπωσιακό αν σκεφτούμε ότι το 9.2% των διαδρομών που παράγουν τα OR-Tools είναι ανέφικτες. Παρουσιάζουμε επίσης στο σχήμα 0.11 τις διαδρομές που υπολογίστηκαν για τους πελάτες εντός Αττικής. Εκεί

παρατηρούμε ότι έχουμε χωρίσει πολύ καλύτερα τους πελάτες της Ανατολικής Αττικής.



(a) Tabu search (b) OR-Tools

Σχήμα 0.11: Διαδρομές εντός Αττικής στο παράδειγμα 4

Κείμενο στα αγγλικά

Chapter 1

Introduction

In this thesis we investigate the Vehicle Routing Problem (VRP). The VRP is a classic combinatorial optimization problem that was firstly introduced in 1959 [Dant59] and has been studied extensively ever since. In the Vehicle Routing Problem, we are given a set of nodes, with one of them being the depot and the rest being the customers. Every customer demands a quantity q_i to be delivered to him. For this purpose, an unlimited homogeneous set of vehicles with capacity C is used. The objective is to find the routes that minimize the total distance travelled with respect to the vehicles' capacity constraints.

The VRP's numerous practical applications have made it one of the most studied problems in the field of operations research. This is evident from the great number of surveys [Eksi09, Brae16] on the various solution approaches and variations of the VRP. Even books have been written specifically for the Vehicle Routing Problem [Toth02a, Gold08, Toth14].

The Vehicle Routing Problem is an NP-hard problem as it is a generalization of the Travelling Salesman Problem. As a consequence, large dimension instances cannot be solved optimally in reasonable computing times and heuristic methods have been developed to try and approximate the optimal solution. Nonetheless, state of the art approaches have achieved great results in the classic Vehicle Routing Problem by using local search metaheuristics [Perr] or a combination of local search with integer programming [Subr13].

The VRP also has a huge amount of variations with additional features and constraints on the routing or the loading of the vehicles. Some of the most notable of these are fixed heterogeneous fleets, time windows, split deliveries and multiple depots. These characteristics arise from real-life challenged as different industries may require a different combination of constraints to be respected.

In the problem we target to solve in this thesis we have a heterogeneous fixed fleet of vehicles, each of which is split into compartments. Each customer may have multiple demands for different product types. Compartments can only host one product type (it is not predetermined) as mixing different products in the same compartment will alter their characteristics. In addition, compartments are not equipped with flow meters and thus cannot store multiple orders even if they are of the same type. This problem is a really important one, as these are constraints that are typically met in the distribution of liquid products and especially oil products. This has led this VRP variation to be also known as the Petrol Station Replenishment Problem (PSRP).

Despite the huge significance of the PSRP, only a few papers that study it have been published and the approaches they used could not apply to the instances we have in our possession. The algorithm we developed draws inspiration from local search procedures used in the classic VRP literature and select some of the moves that would better translate into the compartmentalized variation. The determination of the feasibility of the routes is a much harder task in our case, but we managed to embed it in our local search without introducing a prohibitive overhead.

Local search has had tremendous success in tackling various VRPs and that is why we based our algorithm on it. Generally, local search is a heuristic method that starts on an initial solution and iteratively moves to other solutions with better characteristics. The procedure stops when a stopping criterion is met which usually is a time limit. The solutions that are examined in each iteration are the ones near the candidate solution s and are also called the neighbors of s. This requires a neighborhood relation to exist so as to define what solutions are near another. Local search has had great results in a huge variety of problems.

The simplest form of local search is the hill climbing algorithm. In this algorithm we start from an initial candidate solution and in each iteration we move to the solution in the candidate solution's neighborhood that improves the objective function the most. If no improving neighbor is found the search stops and the last solution is returned.

This procedure is great for finding the optimal solution in convex spaces, but suffers in non-convex problems due to the existence of local optima. Many methods have been used in order to escape local optima, some of the more popular of which are based on restarting the search from a different initial solution. If the new initial solution doesn't take into account the solutions previously explored, it is called repeated local search. In practice, it is common for local optima to not be too far off each other. For this reason, instead of restarting from a completely random solution it is usually better to restart from a perturbation of the current local optimum. This method is called iterated local search. Another technique, with great success in various fields and the VRP especially, which is the one we ultimately used is based on using memory schemes in the local search and is called tabu search.

Tabu search uses a memory structured called the tabu list in order to keep avoid revisiting the last visited solutions. This would be pointless in the previous examples as we avoided cycles by only going to solutions with improving objective value, but this is not the case in tabu search. In tabu search the best neighboring solution not in the tabu list is always picked regardless if it improves the total cost. This helps us escape local optima, as more regions of the search space are explored. Each solution visited is placed in the tabu list for the next θ iterations, with θ being called the tabu tenure. The stopping criterion now is obviously different and usually is a maximum number of iterations without improving the best objective value found or a time bound. Restarts may also be used in the tabu search as a way to intensify the search near the best solution and diversify it by using random perturbations.

It is clear from the above definitions that for our algorithm we needed to create a method to construct the initial solution and define a neighborhood relationship that would apply in our problem. In addition, we had to modify the tabu list so as not to store the whole solutions but rather just some of the moves' distinctive characteristics, because storing the whole solution would drastically increase the memory required and the time needed to search the tabu list, which is something we do often and thus have to do fast.

For the generation of the initial solution we used a greedy route construction heuristic. We started from the first vehicle and always appended to its route the not routed customer with the smallest distance from the last node of the route and was also feasible in the vehicle. The last node is the last customer added or the depot if no customer has been added. If no feasible customer exists we close the route and move to the next vehicle. This way we put nodes that are near each other in the same route and provided a good initialization.

As for the neighborhood structure we used five moves to hop from one solution to the next. Shift (1, 0) moves a customer from one route to another. Swap (1, 1) interchanges the positions of a customer of route i with a customer of route j. Shift (2, 0) moves two consecutive customers from one route to another. Swap (2, 1) exchanges the positions of two consecutive customers of route i with a

customer of route j. Finally, in the Cross move routes i and j, stay the same until their k_1th and k_2th customer, respectively, and exchange their next customers. After a move is applied we try to improve the altered routes even further by solving a TSP on them. All of the above moves are considered in each iteration as long as the resulting solution is feasible.

To determine the feasibility of a route we use a heuristic method that places each order in the compartment with size just above the order size. If no compartment fits the order we use the largest compartment and fill it up to its capacity and put the rest of the order back in the orders that haven't been assigned to a compartment. If this heuristic fails to find an allocation of orders to compartments we use a mixed integer programming solver to determine the feasibility with certainty.

As for the tabu list instead of storing the whole solutions, we store the edges that were recently disconnected so as not to connect them again in the near future. To intensify and diversify the search we also used restarts when the local search was stuck. The best found solution was perturbed by performing randomly one of the moves described above.

To evaluate our algorithm we used three sets of instances. In the first (set A of Augerat [Auge95]) we wanted to see how well our algorithm performs in small instances if we ignore compartment constraints. We found out that our algorithm's solutions have a cost less than 2% compared to the optimal. In the second (set X of Uchoa et al. [Ucho17]) we used a set of benchmark instances for the classical VRP and modified them by adding compartments to the vehicles. The optimal costs for the modified instances are unknown, so we used this set to get an understanding of how well our algorithm loads up the vehicles. Lastly, we used a set of real instances from a fuel distributor and that our algorithm finds better solutions compared to an algorithm that was developed from a previous research team.

In the next chapters we will dive into the details of the topics mentioned in this introduction. In chapter 2 we will give a full formal definition on the Vehicle Routing Problem, see some of its variations and emphasize on the Petrol Station Replenishment Problem. In chapter 3 we investigate local search algorithms and analyze how tabu search has been used in the VRP. In chapter 4 we give a detailed description of our algorithm. In chapter 5, we evaluate our algorithms performance on the three sets of instances mentioned above. Finally, in chapter 6 we conclude this thesis and provide future directions for this line of work.

Chapter 2

The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a generalization of the Travelling Salesman Problem, that was first proposed in 1958 by George Dantzig and John Ramser[Dant59], mentioned then as the Truck Dispatching Problem. The Vehicle Routing Problem is a classical combinatorial optimization problem that is still studied in great extend because of its challenging nature and its numerous real-world applications. As it is implied by its name, the goal is to find the optimal routes for a fleet of vehicles so that a set of locations is visited. The routes are considered optimal if they minimize the objective function, which usually is the sum of the distances covered by the routes.

There are plenty constraints that are commonly found on VRP variations that can be added on the vehicles, the routes and the nodes. If no constraints are added, the optimal solution would consist of only one vehicle and so the problem would be equivalent to the Travelling Salesman Problem. The most common constraint on the vehicles is that all of them have a maximum capacity C of items they can carry. This variation is known as the Capacitated Vehicle Routing Problem (CVRP). In the rest of the thesis we will use the terms classic VRP and CVRP interchangeably. The vehicle set can be either fixed or infinite. The fleet can also be either homogeneous (all vehicles have the same characteristics) or heterogeneous. One example of a constraint on the routes, is that they must not exceed a maximum length L. Finally, the nodes often have to be visited within certain time windows.

2.1 Formal Definition

The Heterogeneous Fleet VRP (HFVRP) with fixed fleet is defined as follows:

A (directed) graph $G(V, E)$ is given, where $V = 0, 1, \ldots, N$ is the node set. Node 0 represents the depot, while the rest are the nodes to be visited (customers). The customer set is also symbolized as $V' = V \setminus 0$ Each customer demands a quantity q_i to be delivered to him from the depot ($q_0 = 0$). Every customer must be served fully by only one vehicle.

The vehicle set M consists of m vehicles. Each vehicle has a capacity Q_i associated with it, alongside a fixed cost F_k and a variable cost λ_k. In addition, a distance matrix is given, which contains the distance d_{ij} for each edge $(i, j) \in E$. The cost of the route r on vehicle k is: $F_k + \sum_{(i,j)\in r} \lambda_k d_{ij}$ ($\lambda_k d_{ij}$ is also symbolized as c_{ij}^k).

Two integer programming formulations have been mainly used to describe the Vehicle Routing Problem variations including the HFVRP: the vehicle flow formulation and the set partitioning formulation. The vehicle flow formulation of the problem is:

$$\min \quad \sum_{k\in M} F_k \sum_{j\in V'} x_{0j}^k + \sum_{k\in M}\sum_{i,j\in V} c_{ij}^k x_{ij}^k \tag{2.1}$$

$$\text{s.t.} \quad \sum_{k\in M}\sum_{i\in V} x_{ij}^k = 1, \quad \forall j \in V' \tag{2.2}$$

$$\sum_{i\in V} x_{ip}^k - \sum_{j\in V} x_{pj}^k = 0, \quad \forall p \in V', \forall k \in M \tag{2.3}$$

$$\sum_{j\in V} x_{0j}^k = 1, \quad \forall k \in M \tag{2.4}$$

$$\sum_{j\in V} x_{j0}^k = 1, \quad \forall k \in M \tag{2.5}$$

$$\sum_{i\in V}\sum_{j\in V} x_{ji}^k q_i \geq Q_k \quad \forall k \in M \tag{2.6}$$

$$x_{ij}^k \in \{0,1\}, \quad \forall i,j \in V, i \neq j, \forall k \in M \tag{2.7}$$

In the above formulation, the objective function (2.1) expresses the minimization of the total costs (fixed and routing). Constrains (2.2) ensure that each customer is visited exactly once by one vehicle. Constraints (2.3) guarantee that if a vehicle visits a customer, it must also depart from it. Constraints (2.4) and (2.5) make certain that each vehicle starts and finishes at the depot. With constraints (2.6) we are sure that the vehicle capacity is never exceeded. Finally, constraints (2.7) are the integrality constraints.

In the literature the set partitioning formulation is found more often. In this formulation the routes are the main entity used. Submaranian et al. (2013) [Subr12] define the problem in a similar way with the following.

Let R be the set of all feasible routes of all vehicles, $\mathcal{R}_i \subseteq R$ be the subset of routes that contain customer $i \in V'$, and $R_k \subseteq R$ be the set of routes associated with vehicle $k \in M$. Define y_j as the binary variable associated to route $j \in R$, and c_j as its cost. The Set Partitioning formulation for the HFVRP can be expressed as follows:

$$\min \quad \sum_{j\in R} c_j y_j \tag{2.8}$$

$$\text{s.t} \quad \sum_{j\in \mathcal{R}_i} y_j = 1 \quad \forall i \in V' \tag{2.9}$$

$$\sum_{j\in R_k} y_j \leq 1 \quad \forall k \in M \tag{2.10}$$

$$y_j \in \{0,1\}. \tag{2.11}$$

The objective function (2.8) minimizes the sum of the costs by choosing the best combination of routes. Constraints (2.9) state that a single route from the subset $\mathcal{R}_i$ visits costumer $i \in V'$. Constraints (2.10) specify that each vehicle must be used at most once. Constraints (2.11) define the domain of the decision variables.

2.2 Variations

The Vehicle Routing Problem has numerous real-world applications. In many of these the constraints are slightly different and so there have been created plenty VRP variations. The most important of

these are:

VRP with time windows

Each customer must be served within a specific time window. Each customer may also be associated with a service time (the duration his service will take). Vehicles may also have time windows within which they operate. Time windows can also be either soft or hard. Soft time windows allow a vehicle to arrive outside the customers time window, but the total cost receives a penalty. Hard time windows, which are the most common, do not allow late arrivals. If a vehicle arrives before the customer is ready to begin service, it may be allowed to wait until that time. [Cord00]

VRP with backhauls / VRP with pickups and deliveries

In the Vehicle Routing Problem with backhauls customers are partitioned into two sets. Linehaul customers demand a quantity to be delivered to them, like in the classic VRP. Backhaul customers have a quantity of product which must be picked up and returned to the depot. In many applications, backhauls customers must be visited after linehaul customers. [Toth02b]

In the Vehicle Routing problem with pickups and deliveries, the goods are picked up at some locations and delivered to others. Pickup and delivery vertices are usually paired meaning that the item picked up at pickup vertex i must be delivered to delivery vertex j and thus i and j must be on the same route. There are cases, however where the product is considered homogeneous and all backhaul customers can serve all linehaul customers. [Parr08]

Multi-depot VRP

The node set contains more than one depot, each of which has its own vehicles.

Periodic VRP

Deliveries are made over multiple days. Each node requires a particular quantity to be delivered to him on a fixed number of visits. [Fran08]

Open VRP

The vehicles do not return to the depot after serving the last customer.

Multi-trip VRP

Each vehicle can make multiple trips. It is combined with time window constraints.

Multi-compartment VRP

In this variation, the products delivered have multiple types. Each vehicle contains compartments and each compartment can only contain products of the same type. The customers demands are now vectors with their size being equal to the different types of products. Customers, of course, are not obligated to have demands for every type (some of the values of the vector can be 0).

A special case of this is the Petrol Station Replenishment Problem, which we are studying in this thesis. In the Petrol Station Replenishment Problem different orders are not allowed to be put in the same compartment, so we could consider each order to define a new product type. More about this problem in the next section.

2.3 The Petrol Station Replenishment Problem

The Petrol Station Replenishment Problem is a variant of the Capacitated Vehicle Routing Problems with more constraints on the loading of the vehicles. As it is implied by its name those constraints are very common on the delivery of oil products, but they are also met in the distribution of other products.

In the problem we are solving, a heterogeneous fleet of vehicles is assigned with the service of N customers. All vehicles are compartmentalized and they are initially located at a central depot, where they are loaded before they start their route and must return there at the end. Each customer has requested the delivery of a number of orders of different product types. Products of different types must not be mixed and thus they are loaded into different compartments. Compartments do not possess flow meters and therefore can only be used to deliver one order. The value to be minimized is the total distance of the routes covered by the vehicles (we do not consider fixed and variable costs of the vehicles).

The main difficulty of this problem that distinguishes from the CVRP is the loading of the vehicles. The loading problem is equivalent to the decision problem of the variable-sized bin covering problem. In the variable-sized bin covering we have a collection of items and bins of different sizes. The items must be packed into the bins, in a way that maximizes the number of bins that are filled. A bin is filled if the combined size of the items packed in it is greater than or equal to the size of the bin. The decision problem of this is if we can fill at least K bins with the items given.

In our case we want to find if a number of orders can fit into the compartments of the vehicles. This is essentially, associating every compartment with an order so that the size of the order is greater than or equal to the size of the compartments assigned to it. So it is equal to the decision problem of the variable-sized bin covering, with the orders being the bins, the compartments being the items and K being equal to the total number of bins.

One of the first publications that dealt with the PSRP was written by Brown and Graves [Brow81]. They created a dispatch system for the distribution of orders that consist of multiple different products. Apart from the standard compartment constraints they also considered overtime costs for vehicles that are routed for many hours, as well as undertime costs.

More recently Cornillier et al. (2008) [Corn08] developed an exact algorithm for the PSRP that arises on the Canadian Province of Quebec. In their case each truck contained three to six compartments, which allowed them to only consider routes that visit up to two customers and ignore all others. This could not be applied in our case as vehicles with 10 compartments are common. Their objective function was to minimize the total cost, instead of total distance. Costs for each route are the sum of a term proportional to the distance covered plus the fixed costs of using the vehicle. In addition, the quantity of each order was not fixed but was between a lower and upper bound. Interestingly, they didn't take the actual quantity delivered into account in the objective function. The fleet in their case is heterogeneous and unlimited and each truck consists of one or two trailers with compartments, where for each trailer the front compartment must be the last to be emptied. This is a constraint that is typical in the industry for reasons related to the stability of the vehicle, but we ignored it in our case. Finally, they consider an upper limit for the duration of each route.

Their approach of solving the problem was to use the set partitioning formulation we mentioned earlier. This would be impossible in most cases, as it requires an exponential number of variables (one for each feasible route). However, since they only consider routes of at most two customers, the number of feasible routes is limited to at most $(n^2 + n)/2$, which makes it pretty easy for an ILP solver to deduce the optimal solution. The set partitioning problem is a little different from the one we mentioned earlier, since the fleet is now unlimited, and they can always select the vehicle that minimizes the route's cost. The only difficulty then is to find for each of those routes the vehicle

that minimizes the route's cost, unless, of course, the route is impossible on all vehicles. In order to determine, if a route is feasible on a vehicle (they called this the Tank Truck Loading Problem) they used a heuristic, which we are not going to list here. If the heuristic failed to provide an optimal solution the following Integer Program is solved:

$$\max \quad \sum_{t=1}^{T} x_t \tag{2.12}$$

$$\text{s.t} \quad a_t \leq x_t \leq b_t \qquad (t \in \{1, \ldots, T\}) \tag{2.13}$$

$$x_t \leq \sum_{c=1}^{C} Q_c y_{tc} \qquad (t \in \{1, \ldots, T\}) \tag{2.14}$$

$$\sum_{t=1}^{T} y_{tc} \leq 1 \qquad (c \in \{1, \ldots, C\}) \tag{2.15}$$

$$y_{tc} = 0 \text{ or } 1 \qquad (t \in \{1, \ldots, T\}; c \in \{1, \ldots, C\}) \tag{2.16}$$

In the above formulation, the constants a_t and b_t are the lower and upper bound of order t respectively, while Q_c represents the capacity of compartment c. The variables x_t are the quantity delivered for order t, while variables y_{tc} tell us if compartment c is used for the delivery of demand t. The objective function is to maximize the total quantity delivered. Constraints (2.13) bound the amount delivered for each order between the lower and upper limit. Constraints (2.14) specify that the capacity of the compartments assigned for order t must be sufficient. Constraints (2.15) impose that each compartment can only be used for a single order.

The only way the vehicle impacts a route's cost (assuming the route is feasible on the vehicle) is through the fixed costs. So in order to find the optimal routes to use in the set partitioning problem they just had to find the vehicle with the least fixed costs that can accommodate the demands of each route. For this purpose, two strategies were used. The first one was to iterate for each route over all vehicles sorted in ascending fixed costs order until a feasible one is found. The second one was a column generation scheme, which we are not going to analyze further.

In their follow-up article Cornillier et. al [Corn09] studied the Petrol Station Replenishment Problem with time windows and multi-trips. This time a limited fleet was used. The objective function was also changed to the maximization of the profits instead of the minimization of the cost. This makes sense, since the quantities to be delivered are not known in advance, but are computed by the algorithm. The final profit is calculated by the revenue generated from the delivered quantities minus the transportation costs and the drivers' wages. Drivers have two different wage rates: regular wages per hour for the first H working hours of the day and overtime for the next H' hours. This time they considered routes that contain up to four customers, which makes it significantly harder for an exact algorithm to be efficient. For this reason, they developed two heuristics based on arc and route preselection.

The mathematical formulation they used for solving the problem follows. The constants used are:

- ϕ regular wage per hour
- ϕ' overtime wage per hour
- α_r earliest departure time for route r
- β_r latest departure time for route r
- λ_r minimum duration for route r (including waiting time if any)
- a_{sr} a binary parameter equal to 1 if and only if station s is delivered within route r
- ρ_{rk} the profit of route r if performed by truck k. This parameter is equal to $-\infty$ if truck k is unable to carry out route r

The decision variables are:

x_{rkv} a binary variable equal to 1 if an only if route r corresponds to trip v of truck k
d_{kv} the departure time of truck k for trip v
h_k the number of regular working hours of truck k
$h_k^{'}$ the number of overtime hours of truck k

The Integer Linear Programming formulation is:

$$\max \quad \sum_{(r,k,v} \rho_{rk} x_{rkv} - \phi \sum_k h_k - \phi' \sum_k h_k^{'} \tag{2.17}$$

$$\text{s.t} \quad \sum_{(r,k,v)} a_{sr} x_{rkv} = 1 \quad \forall s \tag{2.18}$$

$$\sum_r x_{rkv} \leq 1 \quad \forall (k,v) \tag{2.19}$$

$$\sum_r x_{rk,v+1} \leq \sum_r x_{rkv} \quad \forall (k,v) \tag{2.20}$$

$$\sum_r \alpha_r x_{rkv} \leq dkv \leq \sum_r \beta_r x_{rkv} + M(1 - \sum_r x_{rkv}) \quad \forall (k,v) \tag{2.21}$$

$$d_{k,v+1} \geq d_{kv} + \sum_r \lambda_r x_{rkv} \quad \forall (k,v) \tag{2.22}$$

$$d_{kv} - d_{k,1} + \sum_r \lambda_r \leq h_k + h_k^{'} \quad \forall (k,v) \tag{2.23}$$

$$h_k \leq H \quad \forall k \tag{2.24}$$

$$h_k^{'} \leq H' \quad \forall k \tag{2.25}$$

$$d_{kv} \in \mathbf{R}^+ \quad \forall (k,v) \tag{2.26}$$

$$h_k \in \mathbf{R}^+, h_k^{'} \in \mathbf{R}^+ \quad \forall k \tag{2.27}$$

$$x_{rkv} \in 0,1 \quad \forall (r,k,v) \tag{2.28}$$

As we mentioned before the objective function maximizes the total profit. Constraints (2.18) make certain that every customer node is visited once and only once. Constraints (2.19) ensure that at most one route is assigned as the vth trip of vehicle k. Constraints (2.20) specify that trip $v+1$ of a truck may exist if and only if trip v exists as well. Constraints (2.21) enforce the compliance with the time windows. Constraints (2.22) state that a vehicle can only begin a trip after it has returned from the previous one. Finally, constraints (2.23), (2.24), (2.25) calculate the total working hours and ensure they are in the allowable limits.

It is practically impossible to optimally solve the above formulation for big instances, since it requires consideration of all feasible routes. The authors proposed ways to generate these routes quicker eliminating some infeasible routes and they also ignored routes that were dominated by others, but it still was impossible to solve cases with more than 15 customers. This led them to develop two heuristic methods that greatly reduce the number of routes considered.

The first heuristic was an arc preselection heuristic. Two approaches were considered in order to determine which edges to keep. In the first version, the arcs that connected each node with its η nearest neighbors were selected. In the second version, the arc subset included the arcs of the first v minimum spanning trees. The results showed that the minimum spanning tree approach led to greater objective function values. However, this heuristic didn't reduce the solution space as much as needed in order to solve big instances.

The second heuristic was based on route preselection. The geographical space is successively partitioned into sectors of 5 to 10 station, where the problem is solved optimally, until κ routes are selected. The sectors are centered at the depot and are randomly generated each time. Sectors cannot reappear in different partitions. This approach led to worse results in 15 node instances than the arc preselection one, but was more scalable and could be applied to instances with 50 nodes.

The inability of this algorithm to scale with good results, alongside the limitation of only four customers per route make it a bad fit for our case where vehicles with 10 compartments are typical and more than 200 nodes expect deliveries. This forces to explore other directions, like local search algorithms who are proven to have great results in the other VRP variants.

Another notable study of the Petrol Station Replenishment Problem with time windows was made by Benantar et al. [Bena16]. They used a tabu search metaheuristic, which we are going to list in the next chapter, after we have given an introduction on of local search algorithms and especially tabu search.

Chapter 3

Tabu search

3.1 Local search algorithms

Local search is a heuristic method used in many optimization problems. Local search algorithms begin with an initial solution and iteratively try to improve the cost by searching the neighboring candidate solutions of the current solution. This procedure continues, until a stopping criterion is met (e.g. a local optimum is found or a time limit is reached).

From the above definition it becomes clear, that a neighborhood relation must be defined on the search space. We use the following definition from [Aart03] for the neighborhood function.

Definition 3.1.1. Let (S, f) be an instance of a combinatorial optimization problem, where S is the set of feasible solutions, and f is the cost function ($f : S \rightarrow \mathbf{R}$). A neighborhood function is a mapping $N : S \rightarrow 2^S$, which defines for each solution $i \in S$ a set $N(i) \subseteq S$ of solutions that are in some sense close to i. The set $N(i)$ is the neighborhood of solution i, and each $j \in N(i)$ is a neighbor of i.

In more recent works infeasible solutions may be considered as well with a penalty factor, so S could also be defined as the set of all solutions and not just the feasible ones. The selection of a good neighborhood function is of critical importance in the finding of a good solution and it is always one of the challenges of local search as there is not a specific neighborhood function that could be applied to all problems with good results.

The most popular local search heuristics in the Vehicle Routing Problem literature are:

Genetic algorithms

Genetic algorithms are a metaheuristic belonging in the wider class of evolutionary algorithms. Evolutionary algorithms, mimic the way species evolve and adapt to their environment, according to the Darwinian principle of natural selection. Genetic algorithms are evolutionary algorithms that model sexual reproduction, also called genetic crossover. In GAs a population of solutions evolves from one generation to the next through the application of the operators *selection*, *crossover* and *mutation*. This is different than classic local search algorithms where a single solution's neighborhood is examined in each iteration. In addition, solutions are encoded into chromosomes, which are usually bit or integer strings. The genetic operators are applied on the chromosomes instead of the solutions. Starting from an initial population, the genetic algorithm evolves it and yields a population of the same size at each iteration. In the crossover, two parent solutions are selected and create an offspring which carries their best characteristics. In each iteration, multiple crossovers take place combining the characteristics of the best individuals. The children replace the bad solutions in the next generation. Finally some of the offsprings are mutated [Pirl96]. In the end the best found solution is returned. When applied to vehicle routing problems, the classical GA solution scheme is often modified. In particular, the encoding of solutions into chromosomes is either completely ignored (by applying the various operators directly on the solutions) or designed in a very particular way to take advantage of specialized crossover and mutation operators. [Gend08]

Simulated Annealing

Simulated annealing [Kirk83] [Cern85] is a randomized metaheuristic. It draws its name from materials science annealing, which is a process that alters the physical properties of materials, with the purpose to transforming them to a low energy state. It is achieved by heating the material until it melts, maintaining it for an amount of time and finally cooling it gradually. In each iteration, a neighboring solution s_n of the current solution s is selected at random. The selected solution becomes the current solution with a probability that depends on the current solution's energy (objective value), the selected solution's energy and the current temperature. Often improving solutions are accepted by default (probability of 1). The temperature like in the metallurgy annealing begins from a high value and is cooled down in the process. When the temperature is low, decreasing solution are almost never accepted. The algorithm usually terminates after the best found cost hasn't been improved in a fixed number of iterations.

Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a local search method. In Variable Neighborhood Search multiple neighborhood structures are considered as suggested by its name. VNS is based on three facts. First, a local minimum with respect to a neighborhood structure is not necessarily a local minimum with another. Second, a global optimum is a local optimum with respect to all neighborhood structures. Third, for many problems a local optimum with respect to one or more neighborhood structures is close to local optima with other neighborhood structures. VNS consists of two phases that are repeated for a number of times. In the first phase, a descent is performed in order to reach a local optimum with respect to the first neighborhood structure. This is usually achieved by the best improvement (highest decent) heuristic, where the neighbor that improves the solution the most is always selected. Then a perturbation is performed on the current solution and the next neighborhood structure is considered. If a solution is found that is better than the current best solution, the process is restarted from the first neighborhood structure. When the next changing of neighborhood happens in a deterministic way, the method is also called Variable Neighborhood Decent (VND).

Tabu search

Tabu search [Glov98] is another local search metaheuristic that has been extensively used for solving VRPs. Like simulating annealing it also accepts moves that are worse than the current one in order to escape from bad local optima. Tabu search uses a short memory, called tabu list, where it stores the recently visited solutions and forbids them to be selected (marks them as tabu) for a period, called tabu tenure. The reasons this list is kept is to avoid cycles in the solutions visited and, mainly, to explore regions that have not been explored. A simplified version of the tabu search algorithm is presented on Algorithm 1.

Of course, storing all recent solutions and searching in this memory could be a hard task for many problems, so a lot of times some of the characteristics of the moves are stored instead. Such a tabu list could fail to prevent cycling at some cases, but it can make it less frequent, while maintaining, and perhaps even enhancing, its main role of diversifying the search. However, it also has the risk of rejecting solutions that haven't been explored before that could lead to better costs. For this reason, it is common practice to bend the tabu rule if a solution has good characteristics (e.g. it has a lower cost than the current best solution). This is called the aspiration criterion. Another strategy that is sometimes used, is to penalize moves that are in the tabu list according to their frequency there instead of completely forbidding them. Apart from the tabu list (short-term) memory, intensification and diversification strategies are also often used, also seen as intermediate-term and long-term memories respectively. Intensification rules direct the search towards promising areas. This can be achieved by various methods, like forbidding solutions with certain characteristics or adjusting the length of the

Algorithm 1 Tabu search

```
1: initalSolution ← generateInitialSolution()
2: currentSolution ← initialSolution
3: best ← initialSolution
4: tabuList ← []
5: tabuList.add(initialSolution)
6: while not stoppingCriterion fulfilled do
7:     neighborhood ← getNeighbors(currentSolution)
8:     bestNeighbor ← neighborhood[0]
9:     for neighbor ∈ neighborhood do
10:        if (neighbor ∉ tabuList) and cost(neighbor) < bestNeighborCost) then
11:            bestNeighbor ← neighbor
12:        end if
13:    end for
14:    if cost(bestNeighbor) < cost(best) then
15:        best ← bestNeighbor
16:    end if
17:    currentSolution ← bestNeighbor
18:    tabuList.add(currentSolution)
19:    if tabuList.length > TABU_TENURE then
20:        tabuList.removeFirst()
21:    end if
22: end while
23: return best
```

tabu list. Diversification rules take place when the search doesn't encounter improving costs for a number of iterations and restart the search from a random position. We chose to base our algorithm on tabu search, because it has been successfully applied to several variations of the Vehicle Routing Problem. In the next section we present how it is used in practice for VRPs.

3.2 Applications on the Vehicle Routing Problem

In this section we examine how Jose Brandao [Bran11], El Fallahi et al. [ElFa08] and Benantar et al. [Bena16] used tabu search to solve the heterogeneous fixed fleet VRP, the homogeneous fixed fleet multi-compartment VRP and the petrol station replenishment problem. We will focus on the: objective function, generation of initial solution, neighborhood structure, tabu list, intra-route optimizations, intensification, divesification and stopping condition. Before we examine each of those characteristics we should mention the differences of the problems we cite with their equivalent on section 2.2. More specifically, El Fallahi et al. allowed customers to be served by multiple vehicle as long as each product they requested was only delivered by one vehicle. In addition, each vehicle had the same number of compartments with number of different products and compartment i could only be used for product i. Benantar et al. also considered time windows, adjustable orders and labeled vehicles. The last one means that some vehicles are not allowed to visit certain customers, while they can visit others with a bigger cost. They also allowed a customer to be visited by different vehicles for different products. These differences make the last algorithm not directly applicable in our problem.

3.2.1 Objective function

In all three of the papers, the authors allowed routes that violate the capacity, route length and/or time windows constraints, with a penalty factor added to the objective function, in order to explore more regions of the search space. Brandao used the following objective function.

$$f(S^{'}) = \sum_{i=1}^{r}(c_i + Pl_i) \tag{3.1}$$

where $S^{'}$ is the candidate solution, r is the total number of routes, c_i is the cost of the ith route, P is a penalty term and l_i is the load excess in route i.

El Fallahi et al. define the objective function as follows:

$$cost^{'}(S) = cost(S) + \delta Q(S) + \zeta D(S) \tag{3.2}$$

where $cost(S)$ is the routing cost of solution S, $Q(S)$ is the sum of the capacity violations for all routes, $D(S)$ is the total violation in route length, while δ and ζ are the penalty factors. The capacity excess for one route is the sum of the excesses of all of the vehicle's compartments.

Benantar et al. define the objective function on a similar way. Here instead of route length constraints, time window constraints are considered (we won't go into detail on how their violation is computed). The objective function is:

$$F^{'}(S) = F(S) + \alpha C(S) + \beta T(S) \tag{3.3}$$

3.2.2 Generation of initial solution

Brandao begins the construction of his initial solution, by generating a TSP tour with all customers. To create this tour, we start from the depot and each customer is inserted in the order defined by the GENI algorithm [Gend98]. Then this route has to be partitioned into several feasible routes. The partitioning algorithm is the following: The node after the depot is selected. Then we find the free vehicle with the smallest capacity that can hold the node's demands and start a new route with the selected node as the first node (after the depot). The route will also include the next customers of the original tour, as long as they don't violate the capacity constraints. The customers of the new route are removed from the TSP tour and the process is repeated, until all customers have been assigned to a route or when no more vehicles are available. In the latter case, the unrouted customers are inserted, one by one, in the route where the insertion cost is lower, ignoring the capacity violations.

El Fallahi et al. divide the problem to m different VRPs, where m is the number of total different product types. They solve each of these subproblems by using the Clarke and Wright savings heuristic [Clar64] and finally merge them to get an MCVRP solution. Let $E_1, E_2, \ldots, E_m$ be the solutions for each of the m VRP subproblems. To get the MCVRP solution E_1 will be merged with E_2, the result will be merged with E_3, and so on. To merge solutions A and B, each route T of B will be concatenated to its nearest route $T^{'}$ in A. The customers of T are inserted one by one with a least cost insertion. Merging the solutions is always feasible, because each product goes to a predetermined compartment. Thus, this approach can't be applied in our case. The distance between two routes is calculated by the following equation.

$$d(T, T^{'}) = \sum_{x \in T} dist(x, T^{'}) \quad \text{where} dist(x, T^{'}) = \min_{y \in T^{'}} c_{xy} \tag{3.4}$$

Benantar et al. split the construction of the initial routes in two phases: a) loading and b) routing. In the loading phase, a 0-1 integer linear program is solved to assign each order to a set of compartments. The objective function to be minimized is the total number of compartments used. In the routing phase, the orders are already assigned to vehicle and what needs to be determined is the order the customers are visited. For each vehicle the first node visited, is the one with the earliest opening time and every other node after is the one that is closest to the previous and hasn't been visited yet.

3.2.3 Neighborhood structure

Brandao's neighborhood consisted of four different moves: single insertion, double insertion, triple insertion and swap. In order to reduce the neighborhood size, he allowed a customer to be moved into another route only if it contained one of its δ nearest neighbors. In addition, not all moves were examined at each iteration. Parameters F_I, F_D, F_T and F_S were used instead, denoting the frequency single, double and triple insertion and swap moves, respectively.

In the single insertion, a customer is removed from its current route r and is inserted in one of the routes containing one of its δ-nearest neighbors. A new route can be created only if r contains 2 or more customers.

In the double insertion, two customers (a, b) of the same route r are removed and they are inserted in a route r' containing at least one of the δ-nearest neighbors of a or b. The insertion of b takes into consideration the previous insertion of a. A new route can be created only if r contains 3 or more customers.

In the triple insertion, three consecutive customers (a, b, c) of the same route r are removed and they are inserted in a route r' containing at least one of the δ-nearest neighbors of a, b or c. They are inserted either in the same order (a, b, c) or in the reverse c, b, a. A new route can be created only if r contains 4 or more customers.

The vehicles of the routes involved in the move can also change to reduce the cost of the objective function more. Specifically, in insertion moves if the route from where the customers were removed is feasible in its vehicle can get switched to a smaller vehicle that fits its demands, if one is available, whereas if the other route which got the new customers is infeasible it switches to a bigger free vehicle. Similarly for the swap if the new routes are feasible or infeasible respectively.

In the swap move, two customers a and b, belonging into two different routes r_a and r_b respectively, exchange positions. For this move to be allowed, r_b has to contain at least one of the δ-nearest neighbors of customer a and r_a or r_b has to have more than one customer.

El Fallahi et al. considered three neighborhood moves: 2-OPT, relocate and 1-interchange.

2-OPT is a classic TSP local search move. Let i and j be two nodes of route r and $succ(i)$ and $succ(j)$ be their successors in the route. 2-OPT consists of replacing the edges $(i, succ(i))$ and $(j, succ(j))$ with the edges (i, j) and $(succ(i), succ(j))$. El Fallahi et al., also allow i and j to belong to different routes (also seen as 2-OPT* or cross [Tail97] in the literature).

Relocate is similar to the single insertion of Brandao with the exception that all customers that belong to different routes are examined and not just the δ-nearest.

Similarly, 1-interchange is the equivalent of the swap explained before without the δ-neighborhood constraint.

Benantar et al. use the same moves with El Fallahi et al, with two differentiations. Firstly, they use 2-OPT* instead of 2-OPT, excluding intra-route moves. Secondly, they examine a reduced neighborhood with each customer i only being able to move in a route containing one of its neighbors, whose distance from him is less than A_i. In order to calculate A_i for each customer they used the Kolmogorov-Smirnov distance criterion.

3.2.4 Tabu list

Brandao uses the tabu list to prohibit customers to return to a route they left in the previous θ iterations. Moreover, the customers that created a new route cannot leave it for θ iterations. The aspiration criterion is also applied, when an infeasible move that is tabu has a better cost than the best infeasible solution found before, or a feasible solution is found that has a better cost than the best feasible solution yet found.

El Fallahi et al. implement the tabu list as square matrix L of $n \times n$ dimension. The element $L[i, j]$ indicates the iteration in which the edge (i, j) was last removed from the solution. Each edge cannot be reinserted to the solution for the next θ iterations.

Benantar et al. implement the tabu list as an upper triangular matrix L of $K \times K$ dimension, where K is the vehicle set. Each element $L(k, k^{'})$ of the matrix associates route R_k with route $R_{k'}$ and contains the following characteristics: $(R_k, R_{k'}, i, position\ i, j, position\ j, F(S), t.\ R_k$ and $R_k{'})$ are the two routes at the time the move took place, i and j are customers of the route R_k, with $position\ i$ and $position\ j$ being their respective orders in the route. $F(S)$ is the objective value of the solution, and t is the iteration, when arc (i, j) was moved from R_k to $R_{k'}$. The arc cannot be reinserted until iteration $t + \theta$.

3.2.5 Intra-route optimizations

It is common in tabu search approaches to try and improve the modified routes after each iteration. Brandao achieves this with one of three methods: (1) insert the moved customers in their new routes using GENI; (2) US [Gend92]; (3) 2-OPT. Which one will be selected depends on the instance. If the new solution has a better cost than the best feasible solution US is applied to all route to try and improve the cost even better.

El Fallahi et al. don't use any post-optimization technique, but they allow intra-route moves, as we saw earlier with the 2-OPT. Benantar et al. don't use intra-route optimizations.

3.2.6 Intensification and diversification

In order to intensify the search near good regions, Brandao restarts from the best found solution after the objective value hasn't been improved over a number of iterations. In order to diversify the search Brandao modified his objective function (3.1) in the form of the next equation.

$$f_1(S^{'}) = \begin{cases} f(S^{'}), & f(S^{'}) \leq f(S) \\ f(S^{'}) + \beta D\sqrt{rn}\rho/K, & f(S^{'}) > f(S) \end{cases} \tag{3.5}$$

where S is the current solution, β is a scaling parameter, D is the largest absolute difference in $f(S)$ between two consecutive iterations, n is the total number of nodes, ρ is the number of times the customer has been moved (if more than one customer moved, the one with the most moves is considered) and K is the number of iterations executed so far. In addition the tabu tenure, the size δ of the neighborhood and the frequency of moves are changed in a way that diversifies the search. The strategic oscillation, with infeasible solutions also being allowed helps explore regions that wouldn't be explored otherwise and can lead to reduced costs. Finally, they also apply a shaking technique after a large number of iterations.

El Fallahi et al. achieve intensification by adjusting the tabu tenure θ. After a move improves the objective value θ becomes $\max(\theta - 1, \theta_{min})$. Otherwise, if the cost hasn't been reduced in the last ϕ_{LT} iterations θ becomes $\min(\theta + 1, \theta_{max})$. As Brandao, they also diversify the search by admitting

infeasible solutions in the search space. Another diversification strategy is to restart the search from the best solution that has been evaluated but not visited yet, when the local search is stuck in a local minimum. Finally, they also split the route with the maximum feasibility violations into two routes, if the solation is infeasible for ϕ_{viol} consecutive iterations.

Benantar et al. use the same intensification technique regarding the tabu tenure with El Fallahi et al., as well as their restart from the best evaluated solution not yet explored. They also allow infeasible solutions to be explored as both Brandao and El Fallahi et al. Finally, they shake a solution by performing a number of random moves.

3.2.7 Stopping condition

Brandao's algorithms is run as a set of phases with multiple cycles. A cycle ends after K_{BL} iterations without improving the best feasible or infeasible solution. Six phases are used in the small instances and three phases in the larger ones. Each phase apart from the first one begins with the best solution found yet. A phase is terminated if more than K_L iterations have passed and the best feasible solution found hasn't been improved in the last four cycles.

Both El Fallahi et al. and Benantar er al. have the same termination criteria. They stop the local search procedure and restart when γ_{max} iterations have passed without improvement or ν_{max} iterations in total. The maximum number of restarts is η_{max} but the process can stop earlier if χ_{max} restarts without any improvements.

Chapter 4

Our algorithm

In this chapter we present the algorithm we developed for the purpose of my thesis. We propose a tabu search algorithm that is targeted to the Petrol Station Replenishment Problem, but can be also applied to the classical Capacitated Vehicle Routing Problem. It is also easy to extend to include many more VRP variations.

4.1 Algorithm outline

The algorithm presented below is the high level description of the algorithm we developed.

Algorithm 2 Our tabu search algorithm

```
1:  currentSolution ← generateInitialSolution()
2:  bestSolution ← currentSolution
3:  restartNumber ← 0
4:  iteration ← 0
5:  tabuList ← initializeTabuList()
6:  while True do
7:      bestMove ← findBestNeighbor(currentSolution)
8:      currentSolution ← currentSolution.applyMove(bestMove)
9:      tabuList.update(bestMove)
10:     currentSolution.optimizeChangedRoutes()
11:     if currentSolution.cost < bestSolution.cost then
12:         bestSolution ← currentSolution
13:         iteration ← 0
14:     else
15:         iteration ← iteration + 1
16:     end if
17:     if iteration = maxIterations then
18:         if restartNumber = maxRestarts then
19:             return bestSolution
20:         end if
21:     else
22:         restartNumber ← restartNumber + 1
23:         iteration ← 0
24:         currentSolution ← restartNearBest()
25:     end if
26: end while
```

Lines 1-5 initialize the search by constructing an initial solution through a heuristic. The while loop of lines 6-26 is the core of the program. In line 7, the neighborhood of the current solution is searched and the best feasible non-tabu move is selected. In the next two lines (8-9) the selected moves

are applied and the tabu list is updated (both insertion of new elements and deletion of old). Lines 11-16 express that if the current solution corresponds to a lower cost than that of the best solution found so far, the current solution becomes the best solution and the iteration counter resets to 0. Otherwise the iteration counter is incremented by 1. If the iteration counter reaches its maximum allowed values there are two possible ways forward (lines 17-25). If the maximum number of restarts is reached, then the process stops and the best found solution is returned (lines 18-20). Otherwise, the process restarts with a solution in the neighborhood of the best found solution (lines 21-25).

4.2 Generation of initial solution

For the construction of the initial solutions we use the following algorithm:

Algorithm 3 Construction of initial solution

```
1:  notRoutedCustomers ← set(customers)
2:  vehicleIndex ← 1
3:  currentVehicle ← vehicles[vehicleIndex]
4:  currentVehicle.startRoute()
5:  cost ← 0
6:  last ← depot
7:  while notRoutedCustomers ≠ ∅ do
8:      minimumCost ← +∞
9:      selectedCustomer ← null
10:     for customer ∈ notRoutedCustomers do
11:         if currentVehicle.fits(customer) and distance(last, customer) < minimumCost then
12:             minimumCost ← distance(last, customer)
13:             selectedCustomer ← customer
14:         end if
15:     end for
16:     if selectedCustomer ≠ null then
17:         currentVehicle.route.append(selectedCustomer)
18:         cost ← cost + distance(last, selectedCustomer)
19:         last ← selectedCustomer
20:         notRoutedCustomers ← notRoutedCustomers - {selectedCustomer}
21:     else
22:         currentVehicle.closeRoute()
23:         cost ← cost + distance(last, depot)
24:         vehicleIndex ← vehicleIndex + 1
25:         if vehicleIndex > numberOfVehicles then
26:             failure to get a solution
27:         else
28:             currentVehicle ← vehicles[vehicleIndex]
29:             currentVehicle.startRoute()
30:             last ← depot
31:         end if
32:     end if
33: end while
34: for vehicle ∈ vehicles do
35:     vehicle.optimizeRoute()
36: end for
```

We use a simple greedy algorithm, which aims to put nodes that are close to each other in the same route. For the algorithm, we keep a set of the customers that haven't been routed and assign customers to routes one by one removing them from the set. As with most tabu search approaches, we rely on the local search procedure to improve the initial results considerably and don't expect the initialization to be close to optimal. The main reason we want good initializations is to reduce the time it takes the tabu search to reach good regions of the solution space.

At the beginning of the algorithm 3 all customers are not routed, so the set contains every customer (line 1). We start from the first vehicle (lines 2-4). The main loop (lines 7-33) tries to append to the current vehicle's route the customer that is closest to the last node of the route and its demands are feasible in the vehicle alongside the demands of the previous customers (lines 10-20). If no customer is feasible in the vehicle we proceed to the next vehicle where the same procedure is applied (21-14, 27-31). If we run out of vehicles, the algorithm has failed to provide a solution (lines 25-26). Finally, each of the generated routes is optimized by means of the post-optimization technique we describe on a later section (lines 34-36).

This algorithm provides good initial solutions, upon which the local search can build. More importantly, it succeeded in finding a solution for almost all cases.

4.3 Feasibility

As we have mentioned before our algorithm can be used both for the Capacitated Vehicle Routing Problem and for the Petrol Station Replenishment Problem. The main difference between those two problems lies in the vehicles available and the loading of them. For this reason, we use two different methods to determine if a route is feasible on a vehicle.

In the case of the classical VRP, a route is feasible on a vehicle if the sum of the demands of the route's customers is not bigger than the vehicle's capacity. This way, if we store the current load of a vehicle, we can deduce in constant time whether a customer can be added to the vehicle's route.

For the case of the PSRP things are a bit trickier, since vehicles are compartmentalized and each order of a product must not be put in the same compartment with others. As we remember from chapter 2, the loading problem is equivalent to then variable sized bin covering, which is a known NP-hard problem and thus can't be solved optimally in polynomial time. However, in practice vehicle rarely contain more than 10 compartments, which makes it easy for a mixed integer programming solver to solve in reasonable time. To make the procedure even faster, we use a heuristic which falls back to the integer program if it can't resolve the feasibility of the route with certainty.

The heuristic we use is presented on Algorithm 4. The demands of the customers are stored in a list or array (line 2), while we also keep a multi-set with the compartment sizes (line 3). If the requested orders are more than the vehicle's compartments or the sum of the demands is bigger than the sum of compartment capacities the route is infeasible on this vehicle (lines 5-6). If we couldn't determine the infeasibility by this condition we do the following for each order (lines 8-24). We find the compartment the smallest compartment that is big enough to hold the order and assign the order to it (lines 14, 19-22). If no such compartment was found we assign a part of it to the biggest compartment (lines 15-18) and repeat the process for the rest. If we run out of compartments, we try to solve the problem using the MIP solver (lines 11-13). Otherwise, if we have successfully linked each order to a compartment we answer that the route is feasible on the vehicle. For the multi-set we use a binary search tree since all desired operations (insert, delete, ceiling, maxElement) have O(logN) time complexity, where N is the size of the tree. The total complexity then of the heuristic (assuming it solves it without the MIP solver) if we have N compartments and M orders is O(M + NlogN).

Algorithm 4 Feasibility heuristic

```
1:  procedure feasible(route, vehicle)
2:      orders ← list of route customer's order demands
3:      comps ← multi-set of vehicle compartments' sizes
4:      totalDemands ← sum(orders)
5:      if (orders.size > comps.size) or (totalDemands > vehicle.capacity) then
6:          return false
7:      end if
8:      for order ∈ orders do
9:          filled ← 0
10:         while filled < order do
11:             if comps.size = 0 then
12:                 solve with MIP solver
13:             end if
14:             ceil ← comps.ceiling(order - filled)
15:             if ceil = null then
16:                 maxComp ← comps.maxElement()
17:                 filled ← filled + maxComp
18:                 comps.remove(maxComp)
19:             else
20:                 filled ← filled + ceil
21:                 comps.remove(ceil)
22:             end if
23:         end while
24:     end for
25:     return true
26: end procedure
```

The Integer Program, that we solve if the heuristic fails to resolve the feasibility of the route is listed below algorithm 4. Decision variables y_{ij} determine whether order i is placed on compartment j. Let also O be the set of orders and C the set of compartments, with d_i being the demand of order i and q_j the capacity of compartments j.

$$\min \quad \sum_{i \in O} \sum_{j \in C} y_{ij} \tag{4.1}$$

$$\text{s.t} \quad d_i \leq \sum_{j \in C} y_{ij} q_j \quad \forall i \in O \tag{4.2}$$

$$\sum_{i \in O} y_{ij} \leq 1 \quad \forall j \in C \tag{4.3}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i, j) \tag{4.4}$$

where objective function (4.1) specifies the minimization of compartments used. Constraints (4.2) ensure that the compartments assigned for the loading of one order have sufficient capacity. Constraints (4.3) make certain that each compartment is used only once. Finally constraint (4.4) specify that variables y_{ij} are binary.

4.4 Neighborhood structure

In order to determine what the best solution in the neighborhood of another solution is we use the following algorithm:

Algorithm 5 Find best neighbor

```
1:  bestMove ← dummyMove()
2:  for vehicle1 ∈ vehicles do
3:      route1 ← vehicle1.route
4:      for index1 ← 2 to route1.size - 1 do
5:          for vehicle2 ∈ vehicles do
6:              if vehicle1 = vehicle2 then
7:                  continue
8:              end if
9:              route2 ← vehicle2.route
10:             for index2 ← 1 to route2.size - 1 do
11:                 for move ∈ candidateMoves do
12:                     neighbor ← move(route1, index1, route2, index2)
13:                     if neighbor.cost < bestMove.cost then
14:                         if neighbor.isFeasible() then
15:                             bestMove ← neighbor
16:                         else if neighbor.isTransferFeasible() then
17:                             bestMove ← neighbor
18:                         end if
19:                     end if
20:                 end for
21:             end for
22:         end for
23:     end for
24: end for
25: return bestMove
```

At first (line 1) we initialize the best neighborhood move with a dummy move, which is a move with infinite cost. Then we iterate over all vehicle routes and (almost) all positions in them, excluding the case where the two vehicles are the same as we don't want intra-route moves. For each (route1, index1, route2, index2) quadruplet we evaluate all candidate moves (line 12). The variables index1 and index2 indicate that the node that has this position in the route will be moved or another node will be moved after him (which of the two depends on the move). For this reason, the last station (depot) is never considered as it cannot be removed from a route and no node can be added after it.

When we evaluate the move, we don't check feasibilty. We only calculate its cost and check its tabu status. We check if the move is feasible (line 14), only if we already know that it improves the cost of previously found best neighborhood move (line 13). The reason we do this, is that we can calculate the cost of each move in O(1) time, while the feasibility check requires more computational effort for compartmentalized vehicles. If a move is both feasible and results to a lower cost than the previous best neighbor, we set it as the current best neighborhood move. When we know that a move is best in terms of cost, but the new routes are not feasibles in their current vehicles, we check if the infeasible routes can be transferred in a free vehicle that can fit their demands (lines 16-18). Finally, the best found neighborhood move is returned.

Most of the candidate moves we examine are based on the λ-interchanges defined by Osman in 1993 [Osma93]. A λ-interchange is symbolized as (i, j) and implies that i customers are moved from

route a to route b and j customers are moved from route b to route a. The parameters i and j must not exceed λ. In our case we limited λ to 2 and also considered only consecutive customers, as we want to calculate the potential cost of each move in O(1) time. If $i = 0$ or $j = 0$, the move is called a shift. Otherwise it is called an interchange or a swap.

The full list of the neighborhood moves we use in our algorithm is listed below. We also include images that show the new routes after a move is applied to the following routes and their colored indices. We also assume that each move is applied to the quadruplet (r_1, i_1, r_2, i_2) as indicated in algorithm 5.

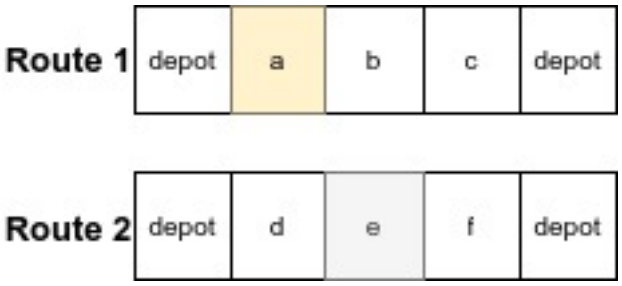


Figure 4.1: Initial routes and selected indices

4.4.1 Shift (1, 0)

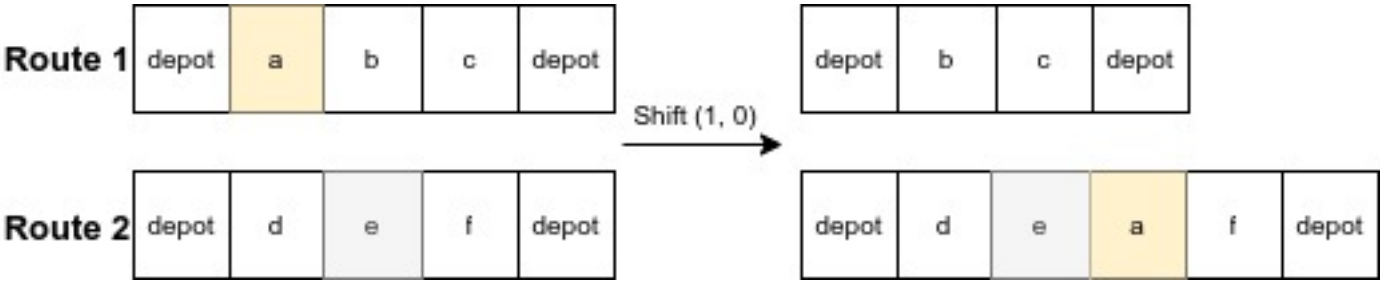


Figure 4.2: Shift (1, 0)

Shift (1, 0) moves the node positioned at index i_1 of route r_1 to the position after i_2 of route r_2, as show in figure 4.2. We don't allow a node that is the only customer of one route to be moved to an empty route. This is a classic move that almost all local search algorithms use.

For the example above, the cost c' of the right solution after the move can be computed as (assuming the cost of the left solution is c):

$$c' = c - distance(depot, a) - distance(a, b) + distance(e, a) + distance(a, f)$$

4.4.2 Swap (1, 1)

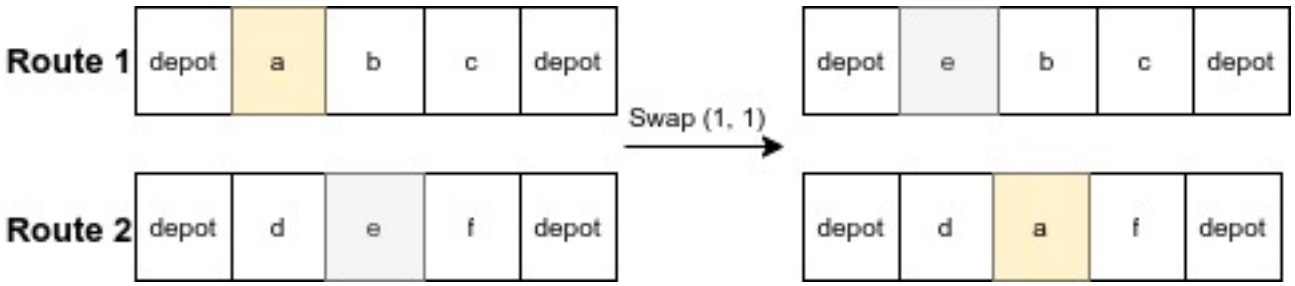


Figure 4.3: Swap (1, 1)

Swap (1, 1) interchanges the i_1th node of route r_1 with the i_2th node of route r_2, as shown in figure 4.3. We don't allow a node that is the only customer of one route to be swapped with another that is alone in its route. In addition the value of i_2 must be greater than 0. This is also a move very often used. It was used by all three algorithms we analyzed in the previous chapter.

For the example above, the cost c' of the right solution after the move can be computed as (assuming the cost of the left solution is c):

$$\begin{aligned} c' = c &- distance(depot, a) - distance(a, b) + distance(d, a) \\ &+ distance(a, f) - distance(d, e) - distance(e, f) \\ &+ distance(depot, e) + distance(e, b) \end{aligned}$$

4.4.3 Shift (2, 0)

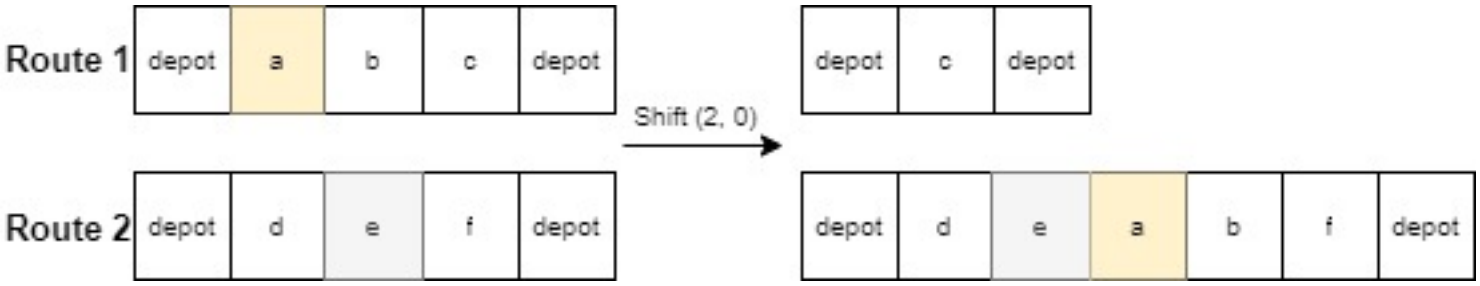


Figure 4.4: Shift (2, 0)

Shift (2, 0) moves the node positioned at index i_1 of route r_1 alongside its following node after i_2th node of route r_2, as shown in figure 4.4. We don't allow two nodes that are the only customer of one route to be moved to an empty route. Also, i_1 must correspond to a position before the pre-last of r_1. This move is equivalent to the application of two consecutive (1, 0) moves and we expect it to have a good impact as close customers are usually consecutive in their routes.

For the example above, the cost c' of the right solution after the move can be computed as (assuming the cost of the left solution is c):

$$c' = c - distance(depot, a) - distance(b, c) + distance(e, a) + distance(b, f)$$

4.4.4 Swap (2, 1)

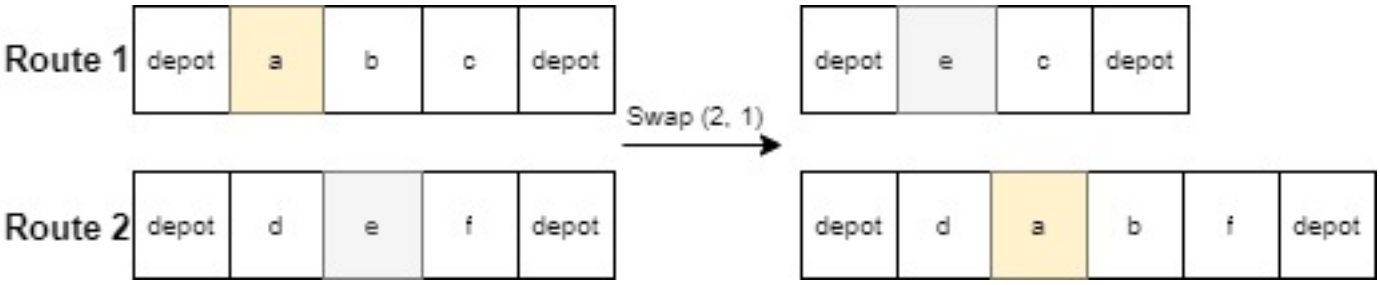


Figure 4.5: Swap (2, 1)

Swap (1, 1) interchanges the i_1th node of route r_1 with the i_2th node of route r_2. The $(i_1 + 1)$th node of r_1 is also moved alongside the i_1th. We don't allow two nodes that are the only customers of one route to be swapped with another that is the alone in its route. In addition, i_1 must correspond to a position before the pre-last of r_1 and the value of i_2 must be greater than 0. This moves combines the nice characteristics of a (2, 0) move with those of the (1, 1).

For the example above, the cost c' of the right solution after the move can be computed as (assuming the cost of the left solution is c):

$$\begin{aligned} c' = c &- distance(depot, a) - distance(b, c) + distance(d, a) \\ &+ distance(b, f) - distance(d, e) - distance(e, f) \\ &+ distance(depot, e) + distance(e, c) \end{aligned}$$

4.4.5 Cross

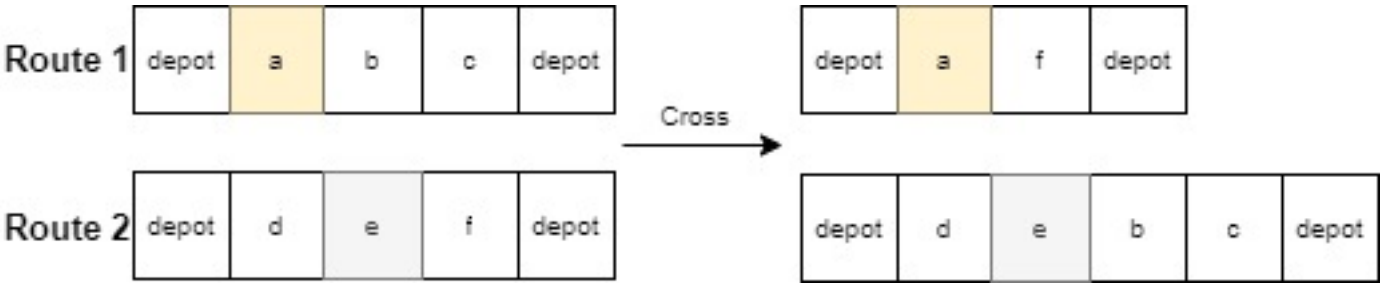


Figure 4.6: Cross

The cross move is equivalent to the 2-OPT* we saw on the previous chapter. The routes r_1 and r_2, stay the same until the i_1th and i_2th customer, respectively, and exchange their next customers. This is illustrated in figure 4.6. We don't allow i_1 and i_2 to be the pre-last positions of their routes simultaneously, as then only the depots would be crossed. This is the most distinctive route of the ones we used and we expect it to have a really good impact as it is the only move that doesn't limit the number of customers moved.

For the example above, the cost c' of the right solution after the move can be computed as (assuming the cost of the left solution is c):

$$c' = c - distance(a, b) - distance(e, f) + distance(a, f) + distance(e, b)$$

4.5 Tabu list

We implement the tabu list as a square matrix of $n \times n$ dimensions, where n is the number of nodes including the depot. A positive value of the cell (i, j) indicates that every move that connects i and j is considered tabu and thus not allowed. If that value equals t then this restriction applies for the next t iterations.

As El Fallahi et al., we also store insert in the tabu list the edges that are removed in each move. For example, in the move of figure 4.5 the cells that get a positive value will be the $(depot, a)$, (b, c), (d, e) and (e, f). If $(depot, e)$, (e, c), (d, a) and/or (b, f) have a possible value during the evaluation of the move then the move would have been tabu and the call of swap(2, 1) with those arguments would return a dummy move with infinite cost.

The $tabuList.update(bestMove)$ call on line 9 of algorithm 2 consists of two phases. Firstly, all positive values of the matrix are decreased by 1, since an iteration has passed. Then, for each of the edges that are removed in the application of $bestMove$ are inserted in the tabu list with their value being equal to a tabu tenure. The tabu tenure is calculated differently for each edge and it is a sum of two terms: the first term is a constant lower bound for the tabu tenure and the second is a random number that is upper bounded. The reason we use the random term is to diversify the search.

4.6 Optimizations

For intra-route optimizations most VRP researchers use the classic TSP moves, like 2-OPT and 3-OPT [Lin65]. These provide good solutions for the general case, but of course they are heuristic methods that don't provide the optimal solution. We want to take advantage of the fact that in our case the number of compartments is an upper limit to the number of customers of the route. So, the maximum number of nodes in a routes is equal to the number of compartments plus one (the depot). In practice, the number of nodes of a route is much smaller as each node may have multiple orders for different product types and some orders require more than one compartment for their storage.

With that in mind, we decided to optimize each changed route by solving the Travel Salesman Problem its nodes define. For this we used the Held-Karp Dynamic Programming algorithm [Bell62, Held62], which has an $O(n^2 2^n)$ running time. This might seem a lot, after all it has an exponential complexity, but it is fast for cases with up to 10 nodes. If a route has more than 12 nodes, we do not do any optimizations. This never happens in the PSRP cases that we have, but because our algorithm can also be used for the classic VRP another procedure like 2-OPT could be added for those cases.

The recursive relationship used to solve the problem is on equation 4.5. The array DP is used for memorizing the costs of the partial solutions. A cell $DP[S, k]$ represents the optimal cost of the Hamiltonian path for the subset $S \cup 1$ of nodes if k is the last node and 1 is the first. The total number of those subsets is 2^{n-1} so $O(2^{n-1}n) = O(n2^n)$ space is required.

$$DP[S, k] = \begin{cases} distance(1, k), & S = \{k\} \\ \min_j DP[S \setminus k, j] + distance(j, k), & otherwise \end{cases} \tag{4.5}$$

The minimum solution cost then is:

$$minCost = \min_j DP[\{2, \ldots, n\}, j] + distance(j, 1)$$

The Held-Karp Dynamic Programming algorithm for the Travelling Salesman Problem is presented on Algorithm 6. The parent array can be used to retrieve the best tour.

Algorithm 6 TSP

```
1:  procedure TSP
2:      for k ← 2 to n do
3:          DP[{k}, k] ← distance(1, k)
4:          parent[{k}, k] ← 1
5:      end for
6:      allSubsets = {S ⊆ {2, . . . , n}} sorted by increasing size
7:      for S ∈ allSubsets do
8:          for current ← 2 to n do
9:              if current ∈ S then
10:                 continue
11:             end if
12:             minCost ← ∞
13:             for previous ∈ S do
14:                 cost ← DP[S \, previous] + distance[previous, current]
15:                 if cost < minCost then
16:                     minCost ← cost
17:                     bestParent ← previous
18:                 end if
19:             end for
20:             DP[S ∪ {current}, current] ← minCost
21:             parent[S ∪ {current}, current] ← bestParent
22:         end for
23:     end for
24:     minCost ← ∞
25:     for prelast ← 2 to n do
26:         cost ← DP[{2, . . . , n}, prelast] + distance[prelast, 1]
27:         if cost < minCost then
28:             minCost ← cost
29:             bestParent ← prelast
30:         end if
31:     end for
32:     return minCost
33: end procedure
```

4.7 Intensification and diversification

We have already explained above some of the diversification and intensification techniques we use. Firstly, we use a random tabu tenure for each element we store in the tabu list. This leads us to explore the search space in a non deterministic way and thus explore regions that we wouldn't ordinarily visit.

Another technique we use is that when we use a move that results in routes with better cost, but are infeasible in their current vehicles we check if any of the free vehicles can accommodate their needs and move them if one is found. This is similar to what Brandao did. In order to avoid checking all the free vehicles Brandao sorted them by capacity and checked the ones that are larger iteratively until one was found. Unfortunately, we cannot do this in the case of compartmentalized vehicles as loading doesn't depend only in the total capacity of the vehicle. What we did to reduce the amount of vehicles checked is that we categorized them by their compartments and didn't check vehicles if we had already checked others with the same compartments.

The third intensification and diversification technique we used was to restart near the best solution

found, when the local search is stuck. This is achieved by restoring the best solution and selecting randomly one of the moves described in section 4.4. Of course there is a high chance that the move we select is not feasible so we repeat the process, until a feasible move is found. This helps us to both intensify the search near the best solution and diversify it by selecting a move in a non-deterministic way.

Chapter 5

Computational results

For the evaluation of our work we used three sets of instances: 1) set A of Augerat [Auge95], 2) set X of Uchoa et al. [Ucho17] and 3) real instances from a fuel distributor. Our tabu search algorithm was coded in Java, and the experiments ran on an Intel Core i5-8250U CPU with 1.6 GHz frequency and 8 GB of RAM. The operating system we used for the evaluation was Ubuntu 20.04. We also used CPLEX 20.1 as our MIP solver. You can find the code we used in `https://github.com/jasonlazar/MCVRP`.

5.1 Evaluation on set A

In the first phase of our evaluation we wanted to test how well our algorithm behaves in the classic Vehicle Routing Problem. Because, our algorithm is primarily targeted to the Petrol Station Replenishment Problem and use TSP as our intra-route optimization technique the algorithm would not be able to scale for big instances.

With that in mind we chose to test our implementation on the classic set A of Augerat, whose instances have from 32 to 80 customers. The fleet is considered homogeneous and infinite with every vehicle having a capacity of 100. Our algorithm is targeted to fixed fleets, but we can simulate an infinite fleet by having as much vehicles as customers, as split deliveries are not allowed. In addition due to the triangle inequality it is always optimal to not leave any node alone in a route if it can join another route. These reasons led us to simulate cases with infinite fleets by using a fleet consisting of $n/3$ vehicles. On this particular set all optimal solution use less than $n/6$ vehicles.

For our experiments we rounded the distances of the nodes to the nearest integer. This is typically done in exact approaches, instead of heuristics but we chose to do it as we wanted to compare our solutions to the known optimal solutions. As for the parameters we considered that the local search stops after 10 iterations without improving the optimal cost and that it restarts 5 times. As for the tabu tenure the constant part was 7 in the instances with up to 45 nodes and 10 in the rest. The random part was up to 5 in all instances.

5.1.1 Comparison with optimal solutions

Modern state of the art approaches that use either branch-and-cut-and-price [Fuka06] or a combination of local search and integer programming [Subr13] have been successful in finding the optimal solutions for set A. We did not expect our algorithm to reach optimality on most instances, since it has not been targeted to this particular problem, but we definitely want to be close to optimal and as is shown in the following table we were able to achieve that.

As is evident from the table 5.1 our solutions are really close to the optimal. The geometric mean of the approximation ratios is less than 1.02, which means that on the average case our algorithm finds a solution that is less than 2% (1.8% in fact) worse than the optimal. Also, in 3 of the instances we

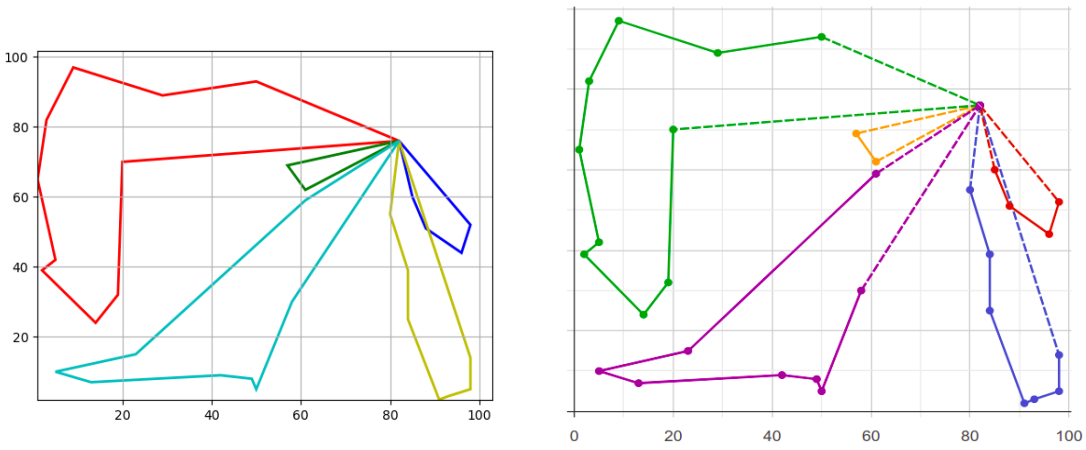
Instance	Optimal	Our	Approximation Ratio
A-n32-k5	**784**	**784**	1.00
A-n33-k5	**661**	**661**	1.00
A-n33-k6	**742**	743	1.00
A-n34-k5	**778**	787	1.01
A-n36-k5	**799**	**799**	1.00
A-n37-k5	**669**	678	1.01
A-n37-k6	**949**	960	1.01
A-n38-k5	**730**	743	1.01
A-n39-k5	**822**	825	1.00
A-n39-k6	**831**	833	1.00
A-n44-k7	**937**	976	1.04
A-n45-k6	**944**	955	1.01
A-n45-k7	**1146**	1179	1.03
A-n46-k7	**914**	927	1.01
A-n48-k7	**1073**	1116	1.04
A-n53-k7	**1010**	1064	1.05
A-n54-k7	**1167**	1168	1.00
A-n55-k9	**1073**	1112	1.04
A-n60-k9	**1354**	1368	1.01
A-n61-k9	**1034**	1076	1.04
A-n62-k8	**1288**	1328	1.03
A-n63-k9	**1616**	1632	1.01
A-n63-k10	**1314**	1357	1.03
A-n64-k9	**1401**	1425	1.02
A-n65-k9	**1174**	1187	1.01
A-n69-k9	**1159**	1170	1.01
A-n80-k10	**1763**	1848	1.05

Table 5.1: Evaluation on set A

find the optimal solution.

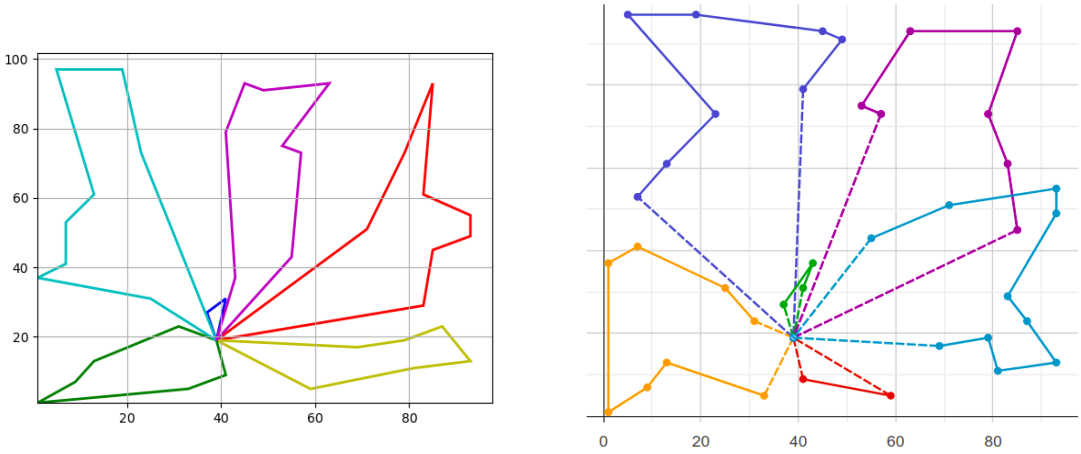
We also showcase 3 examples with both our routes and the optimal routes visualized to help the reader better understand our algorithm's performance.

In figure 5.1 our algorithm manages to find the exact same routes with those of the optimal solution. In figure 5.2 we depict the A-n39-k6, where our routes differ from those found in the optimal solution, but the cost is almost the same (833 vs 831). Finally figure 5.3 depicts the worst case for our algorithm in terms of approximation ratio, where our routes result in a cost 5% higher than the optimal routes. In the last two cases the routes had many differences than the optimal and they couldn't find a move that would lead to a better cost. One way to tackle this would be to either use more restarts to go to a region that would be worse at first but could lead to better results in the future or allow infeasible solutions to be visited.



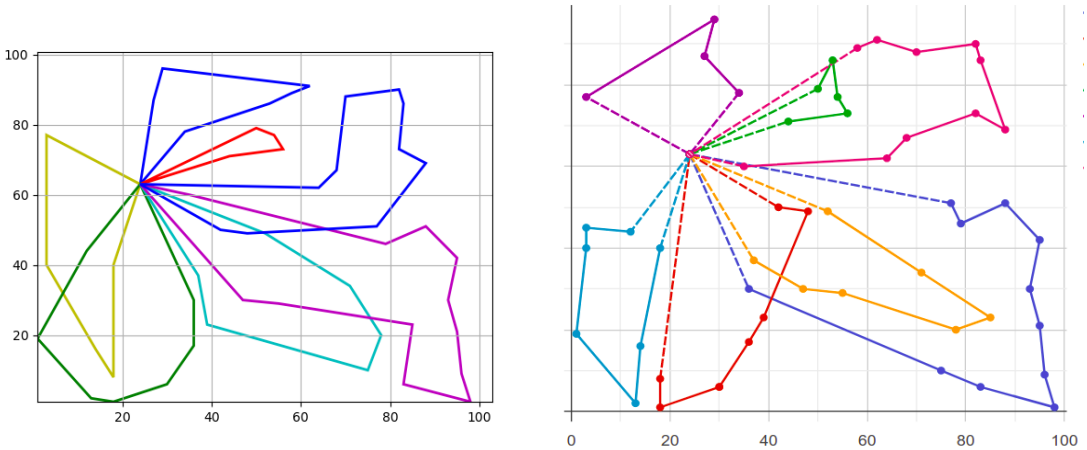
(a) Our solution (b) Optimal solution

Figure 5.1: A-n32-k5 instance



(a) Our solution (b) Optimal solution

Figure 5.2: A-n39-k6 instance



(a) Our solution (b) Optimal solution

Figure 5.3: A-n53-k7 instance

5.1.2 Evaluation of each move

In table 5.2 we show how each modification we made improved the total approximation ratio. We started based on this GitHub repository: `https://github.com/afurculita/VehicleRoutingProblem`. In this repository the author has implemented a tabu search algorithm for the VRP. We kept the tabu list structure and initialization. As for the neighborhood he only used the shift (1, 0) move, with the difference that he also considered intra-route moves. The first alteration we made is to stop examining intra-route moves and instead optimize the routes after each move. The first results are in the first row of table 5.2. We can see that they were 10% higher than the optimal on average.

The next move we added was the swap (1, 1) move and we can clearly see it had a big impact as the approximation ratio dropped from 1.106 to 1.089. This is expected as the size of the neighborhood is drastically larger when we go from one possible move to 2. Shift (2, 0) also had a great impact dropping the approximation ratio even further to 1.075. Swap (2, 1) also helped, albeit not as the previous moves. After adding the approximation ratio was 1.069. The last move we added was the cross move and it had a great impact as expected. The approximation ratio dropped to 1.050, which means that after this we were only 5% worse than optimal.

After we added all the moves, we also added the restarts from near the optimal solution which also helped by letting us escape some local minima. The approximation ratio further dropped to 1.042. All those previous configurations were initially tested with real distances. In order to have a better idea of how we compare with the optimal we rounded the distances to the nearest integer as this is what happens in the optimal solutions as well. With rounded distances the approximation ratio was 1.036. Finally, we also tuned our parameters to the values described in the previous section, instead of the default parameters.

Moves	**Approximation Ratio**
Shift(1, 0)	1.106
Shift(1, 0), Swap(1, 1)	1.089
Shift(1, 0), Swap(1, 1), Shift(2, 0)	1.075
Shift(1, 0), Swap(1, 1), Shift(2, 0), Swap(2, 1)	1.069
Above + Cross	1.050
Above + Shaking	1.042
Round Distances to Nearest Integer	1.036
Round + Parameter Tuning	1.018

Table 5.2: Improvements after addition of each move

5.2 Evaluation on set X

In this section we wanted to test how well our algorithm behaves on a benchmark set of instances if we also add compartment constraints on the vehicles. Set X of Uchoa et al. is also targeted to the homogeneous fleet CVRP so we had to modify it in order to fit our problem. The set contains instances ranging from 101 to 1001 nodes (depot included). The real instances we have in our possession never exceed 220 customers, but we tested on instances with up to 300 nodes to prove our algorithm can scale. For the generation of our heterogeneous fleet we used 5 categories of vehicles, which are described in the following table:

# of compartments	% of total capacity
4	[10, 20, 30, 40]
5	[10, 20, 20, 25, 25]
6	[10, 10, 10, 20, 20, 30]
8	[10, 10, 10, 10, 15, 15, 15, 15]
10	[5, 5, 5, 5, 10, 10, 10, 15, 15, 20]

Table 5.3: Vehicles used in our instances

The total vehicles we used for each instance were twice as much as those of the optimal solution without compartments. We also excluded instances where the ratio r of nodes to vehicles was above 15 as we only have vehicles with up to 10 compartments, which is also usually the case on the real instances. Each vehicle was selected randomly from the categories of table 5.3 and its total capacity was equal to the capacity of the vehicles of the initial instance. When r was > 6 we didn't use vehicles with 4 compartments. Same for 5 compartments when $r > 7$, 6 compartments for $r > 8$ and 8 compartments for $r > 12$.

5.2.1 Presentation of the results

In table 5.4 we present our results on the modified instances of set X. The results of the best known cost without compartments are also mentioned for the sake of completeness, even though they are not suited for comparison with our costs as they solve a much easier problem. At the time of the writing of this thesis all of them have been solved optimally except X-n294-k50. The number of vehicles used in our solutions are also in the table, alongside the time required for the computation and the percentage of the times cplex was used to determine if a loading was feasible.

The parameters we used for our experiments were 5 maximum iterations without improving the best cost and 2 restarts. The constant term of the tabu tenure was 10 and the random part was up to 5 iterations. The results presented are the results of one run, but in reality the program could be run more times and keep the best result, especially for the instances were the time was not much, which would further improve our results. We also did not round the distances.

As expected our solutions, are way behind the best known solutions for the classic problem. This is mainly caused by the many more vehicles used in our solutions, due to the compartment constraints. Especially in the cases where n/k is greater than 10 it would be impossible to recreate the optimal solutions as we only have vehicles with up to 10 compartments. In the next sections we will prove that our algorithm loads the vehicles in a really good way and that also initializing our routes on solutions based on the optimal doesn't produce better costs.

A very encouraging finding from our experiments is the fact that all instances were solved in a reasonable time. This means that our algorithm can be used in practice, as opposed to a theoretical model that would find better solutions but in a bigger time. Many of the instances ran in under 10 minutes which also means that in a real life scenario they could be reran, with possibly higher values for the $maxIterations$ and $maxRestarts$ parameters.

Instance	Best known	Our solution	Vehicles Used	Time	% CPLEX
X-n101-k25	27591	31255	31	1m33s	1.8
X-n106-k14	26362	30822	16	4m0s	8.4
X-n110-k13	14971	17953	17	3m10s	13.4
X-n115-k10	12747	18577	17	7m3s	14.9
X-n125-k30	55539	64370	36	6m29s	0.7
X-n129-k18	28940	38224	24	3m10s	3.0
X-n134-k13	10916	14835	18	2m8s	5.7
X-n139-k10	13590	18663	16	1m17s	11.3
X-n148-k46	43448	53208	59	0m12s	0.1
X-n153-k22	21220	27263	31	20m51s	6.1
X-n157-k13	16876	20221	16	0m3s	0.0
X-n162-k11	14138	20460	19	3m4s	15.7
X-n172-k51	45607	49530	58	11m11s	0.7
X-n176-k26	47812	61226	34	26m1s	4.2
X-n181-k23	25569	22414	19	0m3s	0.0
X-n186-k15	24145	35554	24	25m31s	23.5
X-n195-k51	44225	49985	58	12m43s	1.0
X-n200-k36	58578	69012	43	10m6s	1.4
X-n204-k19	19565	25285	26	3m49s	3.7
X-n209-k16	30656	44791	24	14m57s	15.5
X-n219-k73	117595	42614	22	0m6s	0.0
X-n223-k34	40437	50426	43	9m27s	1.7
X-n228-k23	25742	35984	34	18m30s	8.8
X-n233-k16	19230	27753	27	12m5s	9.1
X-n242-k48	82751	99987	63	4m34s	0.3
X-n247-k50	37274	44624	61	28m12s	1.5
X-n251-k28	38684	48877	36	28m44s	10.6
X-n266-k58	75478	86050	67	4m47s	0.2
X-n270-k35	35291	42869	45	19m20s	7.0
X-n275-k28	21245	22489	29	0m4s	0.0
X-n289-k60	95151	109580	71	20m38s	0.8
X-n294-k50	47161	54881	59	19m14s	1.5
X-n298-k31	34231	43473	41	10m49s	4.4

Table 5.4: Results on the modified set X

Finally, our heuristic to determine the feasibility of a loading (algorithm 4) seems to be performing really well, especially when n/k isn't big. The cases where CPLEX is never used are outliers that only contain unitary demands. These instances are good benchmark instances, but are not met in real life. Even if we ignore those three however the geometric mean of the column is just 3%, which means that on an average instance the heuristic successfully handles 97% of the calls.

5.2.2 Loading

In this subsection we will examine how well our algorithm loads up the vehicles. In the industry the loadings that fill over 60% of the vehicles total capacity are considered acceptable. We are not going to present each vehicle, as there are too many instances and each has many vehicles. Instead we are going to present the average loading for each instance set and analyze only a number of instances that are representative of the whole set.

Instance	Average loading %
X-n101-k25	81
X-n106-k14	82
X-n110-k13	76
X-n115-k10	50
X-n125-k30	82
X-n129-k18	86
X-n134-k13	66
X-n139-k10	64
X-n148-k46	92
X-n153-k22	67
X-n157-k13	88
X-n162-k11	55
X-n172-k51	86
X-n176-k26	75
X-n181-k23	97
X-n186-k15	57
X-n195-k51	88
X-n200-k36	82
X-n204-k19	69
X-n209-k16	64
X-n219-k73	99
X-n223-k34	89
X-n228-k23	66
X-n233-k16	57
X-n242-k48	94
X-n247-k50	77
X-n251-k28	81
X-n266-k58	93
X-n270-k35	78
X-n275-k28	91
X-n289-k60	85
X-n294-k50	85
X-n298-k31	81

Table 5.5: Average loading of each vehicle

The results of table 5.5 prove that our algorithm provides great loadings for the vehicles. We mentioned that a loading that fills over 60% of the vehicle is considered acceptable. Our algorithm finds excellent (above 80%) loadings for the average vehicle in 55% of the occasions. This number goes to 67% if we include average loadings of 75% or higher. We also observe that all except two (X-n153-k22 and X-n228-k23) of the rest have a n/k ratio over 10 and thus it would be impossible for us to load the vehicles more. So 22 out of 24 (92%) instances with a n/k ratio under 10 load more than 75% of the vehicle on average.

Let us now handpick and examine even further some of the instances to see how each vehicle is loaded and also see their routes visualized. In the following tables we present in detail the loading for each vehicle of those instances. The loading% column indicates the percentage of the total capacity filled for the vehicle, while # compartments is the total number of compartments of the vehicle. The last column shows how each individual compartment is loaded. The bold part is the compartment's capacity while the number in the parenthesis is the quantity of the product loaded into the compartment.

X-n115-k10

This is the instance were our algorithm had the worst average loading of (50%) which is unacceptable in real instance. However we should keep in mind that this is a benchmark instance and specifically designed to challenge even the best algorithms. In real life, the compartment sizes and the order quantities are related values. In this particular instance the n/k ratio is 11.5, so we used vehicles of 8 and 10 compartments. Out of the 20 vehicles available, we were unlucky that the random selection determined that only 6 of them should have 10 compartments. This forces us to use mostly 8 compartment vehicles in an instance were the optimal no-compartment solution has an average route length of 11.5.

Vehicle	Loading%	# Compartments	Compartments
6	82	10	**33**(33) **25**(25) **25**(25) **16**(9) **16**(10) **16**(10) **8**(3) **8**(5) **8**(8) **8**(5)
19	80	10	**33**(33) **25**(25) **25**(25) **16**(10) **16**(10) **16**(9) **8**(6) **8**(1) **8**(8) **8**(4)
13	77	8	**25**(25) **25**(25) **25**(25) **25**(24) **16**(10) **16**(5) **16**(10) **16**(2)
9	72	10	**33**(33) **25**(25) **25**(25) **16**(10) **16**(9) **16**(3) **8**(1) **8**(3) **8**(2) **8**(6)
3	71	8	**25**(25) **25**(25) **25**(25) **25**(21) **16**(8) **16**(2) **16**(4) **16**(7)
12	70	10	**33**(33) **25**(25) **25**(21) **16**(4) **16**(4) **16**(3) **8**(2) **8**(7) **8**(7) **8**(8)
7	68	8	**25**(25) **25**(25) **25**(25) **25**(23) **16**(5) **16**(5) **16**(3) **16**(1)
10	66	10	**33**(33) **25**(25) **25**(25) **16**(1) **16**(4) **16**(5) **8**(1) **8**(6) **8**(6) **8**(1)
16	63	8	**25**(25) **25**(25) **25**(24) **25**(8) **16**(1) **16**(5) **16**(9) **16**(6)
15	55	8	**25**(25) **25**(25) **25**(9) **25**(5) **16**(3) **16**(8) **16**(3) **16**(12)
5	52	8	**25**(25) **25**(25) **25**(1) **25**(4) **16**(9) **16**(1) **16**(5) **16**(15)
8	49	8	**25**(25) **25**(25) **25**(5) **25**(5) **16**(2) **16**(10) **16**(1) **16**(7)
14	40	10	**33**(5) **25**(7) **25**(4) **16**(10) **16**(9) **16**(9) **8**(8) **8**(5) **8**(6) **8**(2)
0	29	8	**25**(5) **25**(8) **25**(8) **25**(5) **16**(3) **16**(8) **16**(2) **16**(8)
1	23	8	**25**(8) **25**(0) **25**(0) **25**(0) **16**(7) **16**(6) **16**(6) **16**(10)
2	23	8	**25**(3) **25**(6) **25**(6) **25**(4) **16**(5) **16**(4) **16**(2) **16**(7)
11	21	8	**25**(6) **25**(3) **25**(1) **25**(1) **16**(8) **16**(2) **16**(8) **16**(5)
4	0	8	**25**(0) **25**(0) **25**(0) **25**(0) **16**(0) **16**(0) **16**(0) **16**(0)
17	0	8	**25**(0) **25**(0) **25**(0) **25**(0) **16**(0) **16**(0) **16**(0) **16**(0)
18	0	8	**25**(0) **25**(0) **25**(0) **25**(0) **16**(0) **16**(0) **16**(0) **16**(0)

Table 5.6: X-n115-k10 loading

On table 5.6 we can clearly see that all 10-compartment vehicles are fully used. Actually the only vehicle where not all of its compartments are utilized is vehicle 1 which only serves 5 customers, whose demands require just one compartment each. However, none of the other vehicles have any space to fit them. Therefore, although the percentage of the loading is really low on some vehicles that is not due to empty compartments, but because the vehicles are loaded with small orders.

In addition this instance is really special as 12 of the 114 customers have placed really large orders $(60 - 99)$, while the rest have placed really small orders (≤ 10). The total capacity of the vehicles is 169. In the optimal non-compartment solution 2 of the routes contain 2 of the first 12 customers. Compartment constraints would let us to place those customers in the same routes but they would have been the only customers of their route (the two pairs of orders had capacities 86, 65 and 74, 87 respectively), which would cause us to use even more vehicles with probably even greater cost. Even in the optimal solution, these routes are two of the three smallest. This has as an effect some of the rest of the routes to have way more than the mean of 11.5 customers. In fact the standard deviation of the optimal routes' length is 6.4, and there is one route with 19 customers.

In figure 5.4 we present the routes computed by our algorithm. We can see that they manage to ship close customers together, with a few exceptions that are probably cause by the compartment

constraints.

X-n134-k13

This is also an instance where the ratio of nodes to vehicles is above 10. It is in fact 10.3, so we again used only vehicles of 8 and 10 compartments selected randomly. The standard deviation of the routes' length on the optimal solution is again high (4.7), which means that the optimal solution again uses much longer routes. Here, however our average loading percentage is way bigger than in the previous example. Actually the average loading is 66% of the vehicle's capacity, which means that the loading of the average vehicle is acceptable. Another thing to note is that in this instance the distribution of customers' demands is normal, contrary to the X-n115-k10 where all orders where either extremely big or extremely small. On the following table we present the details of the vehicles' loading.

Vehicle	Loading%	# Compartments	Compartments
10	87	8	**96**(90) **96**(74) **96**(80) **96**(91) **64**(53) **64**(61) **64**(52) **64**(53)
1	85	8	**96**(69) **96**(80) **96**(95) **96**(92) **64**(64) **64**(58) **64**(35) **64**(53)
6	85	8	**96**(86) **96**(73) **96**(77) **96**(90) **64**(54) **64**(54) **64**(55) **64**(55)
4	84	8	**96**(69) **96**(78) **96**(95) **96**(73) **64**(64) **64**(64) **64**(59) **64**(36)
5	84	8	**96**(79) **96**(91) **96**(85) **96**(72) **64**(64) **64**(52) **64**(33) **64**(60)
9	84	8	**96**(85) **96**(89) **96**(67) **96**(78) **64**(51) **64**(53) **64**(59) **64**(54)
7	83	10	**128**(73) **96**(70) **96**(93) **64**(62) **64**(60) **64**(64) **32**(20) **32**(32) **32**(32) **32**(28)
3	82	8	**96**(79) **96**(74) **96**(95) **96**(85) **64**(64) **64**(57) **64**(64) **64**(7)
12	82	8	**96**(89) **96**(87) **96**(85) **96**(84) **64**(51) **64**(64) **64**(13) **64**(54)
17	81	8	**96**(72) **96**(83) **96**(74) **96**(74) **64**(56) **64**(63) **64**(64) **64**(30)
11	78	8	**96**(70) **96**(73) **96**(73) **96**(71) **64**(56) **64**(64) **64**(64) **64**(25)
8	77	10	**128**(76) **96**(73) **96**(67) **64**(64) **64**(64) **64**(64) **32**(25) **32**(25) **32**(32) **32**(3)
2	74	8	**96**(66) **96**(77) **96**(65) **96**(74) **64**(64) **64**(64) **64**(35) **64**(31)
14	63	10	**128**(93) **96**(93) **96**(10) **64**(56) **64**(42) **64**(46) **32**(7) **32**(12) **32**(23) **32**(22)
22	55	10	**128**(98) **96**(12) **96**(47) **64**(45) **64**(40) **64**(37) **32**(12) **32**(19) **32**(17) **32**(26)
20	53	10	**128**(90) **96**(26) **96**(29) **64**(34) **64**(39) **64**(33) **32**(28) **32**(29) **32**(12) **32**(20)
15	37	10	**128**(49) **96**(13) **96**(34) **64**(35) **64**(26) **64**(9) **32**(3) **32**(28) **32**(23) **32**(16)
0	10	8	**96**(66) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
13	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
16	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
18	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
19	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
21	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
23	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
24	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)
25	0	8	**96**(0) **96**(0) **96**(0) **96**(0) **64**(0) **64**(0) **64**(0) **64**(0)

Table 5.7: X-n134-k13 loading

As we can see on table 5.7 we were again unlucky that only 6 of the 26 vehicles have 10 compartments and the rest have 8. As with before, our algorithm cleverly routes all 10-compartment vehicles, which are fully utilized. In addition, most vehicles have good loadings (13/18 above 70% and 10/18 above 80%), but the overall percentage gets dragged down by a few outliers. Of the rest two instances have really bad loading percentages with 10 and 37% respectively. The second one is a classic example of a route containing only customers with small demands, as all compartments are used but none is full. The first is the most problematic as only one customer is served and their order isn't that big. However, we notice that all the other vehicles have used all of their compartments and so

that customer couldn't join any of the other routes. If we look, at figure 5.5, we can clearly see that our algorithm has made the perfect choice of which node to leave alone (it is the only one left of the depot).

X-n106-k14

In this instance our algorithm is very successful in finding a good loading for the vehicles. Here the n/k ratio is equal to 7.5, so we used vehicles with 6, 8 and 10 compartments. The standard deviation of the routes' length in the optimal no-compartment solution is also relatively small (1.8), which helps our algorithm find better loadings. The next table shows the loadings our algorithm produced in detail.

Vehicle	Loading%	# Compartments	Compartments
6	91	10	**120**(95) **90**(78) **90**(88) **60**(56) **60**(59) **60**(60) **30**(23) **30**(30) **30**(30) **30**(28)
26	89	10	**120**(98) **90**(88) **90**(76) **60**(60) **60**(60) **60**(38) **30**(27) **30**(30) **30**(30) **30**(28)
0	88	10	**120**(92) **90**(87) **90**(79) **60**(60) **60**(58) **60**(60) **30**(27) **30**(11) **30**(30) **30**(27)
20	86	8	**90**(82) **90**(78) **90**(81) **90**(75) **60**(60) **60**(38) **60**(60) **60**(39)
22	85	8	**90**(82) **90**(86) **90**(70) **90**(71) **60**(50) **60**(60) **60**(40) **60**(50)
2	84	10	**120**(97) **90**(84) **90**(65) **60**(52) **60**(54) **60**(51) **30**(30) **30**(30) **30**(30) **30**(10)
17	84	8	**90**(83) **90**(83) **90**(90) **90**(83) **60**(9) **60**(60) **60**(36) **60**(59)
13	83	8	**90**(89) **90**(90) **90**(81) **90**(83) **60**(60) **60**(56) **60**(7) **60**(34)
27	83	10	**120**(99) **90**(70) **90**(72) **60**(50) **60**(58) **60**(56) **30**(30) **30**(30) **30**(30) **30**(2)
21	82	8	**90**(66) **90**(65) **90**(80) **90**(63) **60**(58) **60**(60) **60**(58) **60**(39)
4	81	10	**120**(92) **90**(81) **90**(90) **60**(60) **60**(60) **60**(60) **30**(5) **30**(25) **30**(11) **30**(3)
14	79	10	**120**(67) **90**(81) **90**(86) **60**(53) **60**(59) **60**(60) **30**(20) **30**(30) **30**(20) **30**(0)
25	79	8	**90**(75) **90**(61) **90**(63) **90**(76) **60**(60) **60**(59) **60**(50) **60**(32)
11	78	10	**120**(64) **90**(67) **90**(72) **60**(53) **60**(51) **60**(60) **30**(15) **30**(30) **30**(30) **30**(26)
12	76	8	**90**(73) **90**(66) **90**(90) **90**(62) **60**(57) **60**(1) **60**(55) **60**(50)
15	63	6	**180**(77) **120**(85) **120**(65) **60**(53) **60**(60) **60**(36)
1	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
3	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
5	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
7	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
8	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
9	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
10	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
16	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
18	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
19	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
23	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)
24	0	6	**180**(0) **120**(0) **120**(0) **60**(0) **60**(0) **60**(0)

Table 5.8: X-n106-k14 loading

As we can see 13 vehicles have 6 compartments, 7 have 8 compartments and 8 vehicles have 10 compartments. The loadings are excellent as 11 of the 16 vehicles (69%) have more than 80% of their capacity filled and the lowest loading percentage is 63, which is still acceptable. Again, all 10-compartment vehicles have been used, as well as all the 8-compartment vehicles. The only non-used vehicles have 6 compartments. In figure 5.6 we show the routes our algorithm produced.

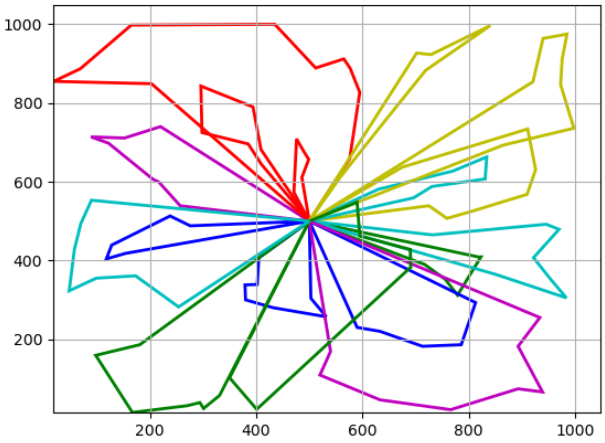


Figure 5.4: X-n115-k10 routes

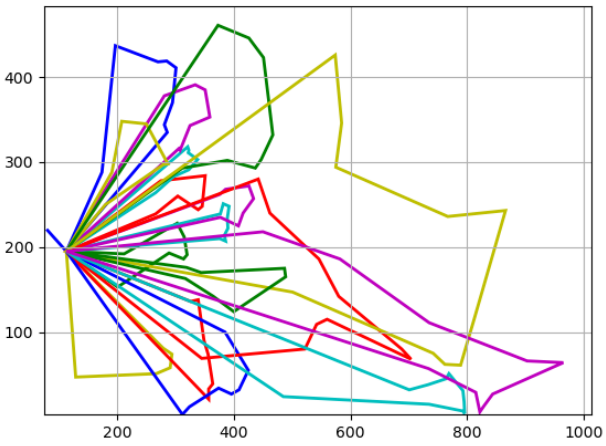


Figure 5.5: X-n134-k13 routes

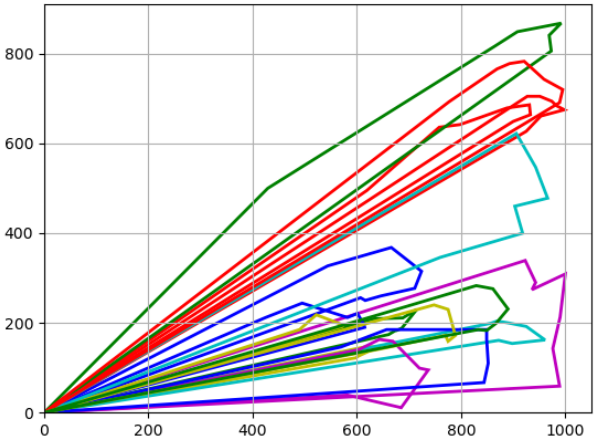


Figure 5.6: X-n106-k14 routes

5.2.3 Comparison with other initialization methods

In this subsection, we want to prove that our local search procedure manages to find good solutions despite the initialization. For this reason we used two other initialization methods.

The first initializes the routes based on the routes of the optimal solution without compartments. Specifically, we tried to use the optimal routes and split the into two or three routes when they weren't feasible. This technique should produce better results in theory as the routes used to initialize the process are near the optimal. If our algorithm's solution are near these, then it is an indication that it provides good quality solutions. In table 5.9 we present the costs found by our algorithm using our initialization, using the initialization from the optimal values and how much above(%) our costs are.

Instance	Our solution	Init. from opt.	Difference%
X-n101-k25	31254	31428	-0.6
X-n106-k14	30822	32117	-4.0
X-n110-k13	17952	17866	0.5
X-n115-k10	18576	Impossible	N/A
X-n125-k30	64370	65835	-2.2
X-n129-k18	38224	37899	0.9
X-n134-k13	14835	Impossible	N/A
X-n139-k10	18663	18537	0.7
X-n148-k46	53208	53277	-0.1
X-n153-k22	27263	Impossible	N/A
X-n157-k13	20221	20479	-1.3
X-n162-k11	20460	20796	-1.6
X-n172-k51	49530	50026	-1.0
X-n176-k26	61226	Impossible	N/A
X-n181-k23	22414	22319	0.4
X-n186-k15	35554	35877	-0.9
X-n195-k51	49985	49294	1.4
X-n200-k36	69012	70371	-1.9
X-n204-k19	25285	25370	-0.3
X-n209-k16	44791	45425	-1.4
X-n219-k73	42614	42117	1.2
X-n223-k34	50426	50017	0.8
X-n228-k23	35984	35639	1.0
X-n233-k16	27753	Impossible	N/A
X-n242-k48	99987	100174	-0.2
X-n247-k50	44624	45812	-2.6
X-n251-k28	48877	49493	-1.2
X-n266-k58	86050	86311	-0.3
X-n270-k35	42869	Impossible	N/A
X-n275-k28	22489	22479	0.0
X-n289-k60	109580	110519	-0.8
X-n294-k50	54881	55578	-1.3
X-n298-k31	43473	43777	-0.7

Table 5.9: Initialization from optimal vs greedy init

In table 5.9 'Impossible' indicates that the routes couldn't be initialized based on the optimal solution as too many splits were required. Our algorithm using our greedy initialization seems to be

finding almost identical costs with our algorithm using the initialization based on the optimal routes. In fact, our costs were 0.58% lower, probably due to the randomness of the algorithm.

The second initialization technique we used was to put the customers to vehicles in the order they are given in the instance file. This order is has no logical reason behind it, so this method can be seen as a random initialization. If we achieve to get good solutions with this initialization, we prove that our local search procedure works and can drastically improve bad solutions. In table 5.10 we present the initial and final costs found by using our greedy initialization and the random method we just described.

Instance	g_initial	g_final	g_improvement%	r_initial	r_final	r_improvement%
X-n101-k25	37058	31254	15.66	65871	31659	51.94
X-n106-k14	35545	30822	13.29	64437	31271	51.47
X-n110-k13	20724	17952	13.38	61680	18428	70.12
X-n115-k10	22277	18576	16.61	57531	18832	67.27
X-n125-k30	75890	64370	15.18	107065	64989	39.3
X-n129-k18	42104	38224	9.22	86329	37835	56.17
X-n134-k13	16497	14835	10.07	46404	15238	67.16
X-n139-k10	21783	18663	14.32	72417	18709	74.16
X-n148-k46	72774	53208	26.89	99486	52556	47.17
X-n153-k22	31998	27263	14.8	76919	27859	63.78
X-n157-k13	21907	20221	7.7	75807	20428	73.05
X-n162-k11	23413	20460	12.61	81382	21289	73.84
X-n172-k51	61822	49530	19.88	100922	50083	50.37
X-n176-k26	70637	61226	13.32	142416	62245	56.29
X-n181-k23	27604	22414	18.8	80813	22674	71.94
X-n186-k15	39034	35554	8.92	115075	37333	67.56
X-n195-k51	60753	49985	17.72	110204	49574	55.02
X-n200-k36	80841	69012	14.63	139476	67772	51.41
X-n204-k19	28008	25285	9.72	93153	26016	72.07
X-n209-k16	48986	44791	8.56	131291	45880	65.05
X-n219-k73	69678	42614	38.84	158519	43120	72.8
X-n223-k34	59900	50426	15.82	151770	50353	66.82
X-n228-k23	41989	35984	14.3	130267	36470	72.0
X-n233-k16	32849	27753	15.51	111757	29294	73.79
X-n242-k48	110582	99987	9.58	203837	100255	50.82
X-n247-k50	53731	44624	16.95	100932	45130	55.29
X-n251-k28	51554	48877	5.19	138196	50050	63.78
X-n266-k58	95848	86050	10.22	173234	87545	49.46
X-n270-k35	50832	42869	15.67	153204	43799	71.41
X-n275-k28	26613	22489	15.5	101400	22974	77.34
X-n289-k60	126824	109580	13.6	224623	110093	50.99
X-n294-k50	66288	54881	17.21	168445	55392	67.12
X-n298-k31	50301	43473	13.57	168982	45173	73.27

Table 5.10: Initialization from random vs greedy init

It is evident from the above table that our algorithm can greatly improve bad solutions and reach good solution areas. The routes our algorithm finds with the random initialization are worse than with our initialization as expected, but they are really close to them.

5.3 Evaluation on real instances

In this section we test our algorithm on a set of real instances provided to us from a fuel distributor operating in Greece. In their real-life scenarios, they also have more constraints like time-windows, labeled vehicles and more loading constraints on how the compartments are filled. They also allow multiple trips per vehicle. We ignored those and solved the instances using only the constraints relevant to our problem. Getting past the multi-trip attribute was hard, as our algorithm failed to provide an initial solution to some instances. Nevertheless, we succeeded to get results in 5 instances, which we are going to present.

We compare our results with those provided by an algorithm developed by a previous research team for the company. This algorithm used Google's OR-Tools to assign the customers to vehicles. Because OR-Tools don't support compartment constraints, they tried simulating them by using some weaker constraints. This leads most of the routes generated by this algorithm to being infeasible. To overcome this, they used a matching procedure at the end where they tried to much the routes produced with their corresponding vehicles. In the matching, routes and vehicles are represented as nodes in a bipartite graph. An edge (r, v) in this graph symbolize that route r is feasible on vehicle v. In the matching, the goal is to select the maximum number of edges, such that every edge selected has no common vertices with another. This helped, but in almost all cases there were still infeasible routes left.

Because, their algorithm after the matching only calculates the feasible routes, we expect them to find lower costs in general. In table 5.11 we present the costs of our solutions compared to the matching algorithm and how bigger are the costs we find in percentage. We also mention the percentage of the feasible routes their algorithm finds. We also present in a column the costs they find before the matching, which take into account all routes. Because some of them are infeasible we also expect them to have a lower cost than ours.

Instance	Tabu search	Time	OR-Tools	Difference%	Matching	Difference%	Feasible%	Time
1	10115085	14m59s	9803947	3.2	9036121	11.9	90.9	17m40s
2	10399898	23m15s	9984231	4.2	9053257	14.9	92.2	15m3s
3	7550051	13m41s	7336853	2.9	7002968	7.8	93.8	16m35s
4	5214640	15m36s	5989852	-12.9	5245078	0.0	90.8	17m22s
5	5120051	1m20s	5095335	0.0	5095335	0.0	100.0	1m40s

Table 5.11: Performance on real instances

Despite our algorithm, finding feasible routes in all cases we are never more than 4.2% worse than the OR-Tools solution, which has weaker constraints and allows infeasible routes. There is only one case where the OR-Tools/Matching algorithm achieves in providing feasible routes, which is a special case with smaller dimension and only rural customers. We can also see that in 4 of the 5 occasions our algorithm was faster in finding a solution. As for the evaluation of our loading heuristic, we found that in no instance CPLEX was needed more than 0.3% of the times.

The most noteworthy case of the above is definitely instance 4, where we are 12.9% better, in spite of their feasibility rate being only 91%. Even if we remove those routes altogether, our algorithm still has a slightly lower cost than the other. In the next pages we are going to examine the routes found by each algorithm and explain the strengths of our algorithm.

Instance 1

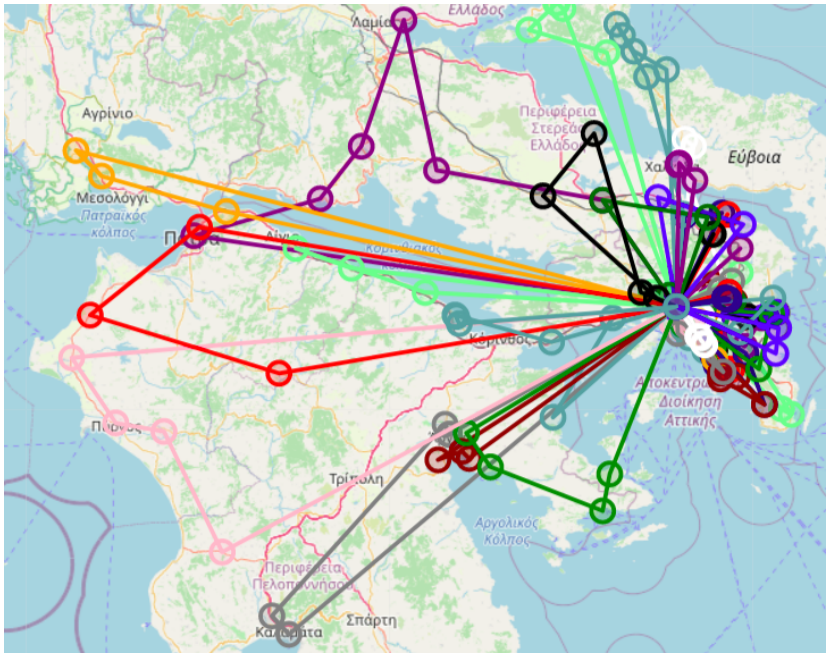
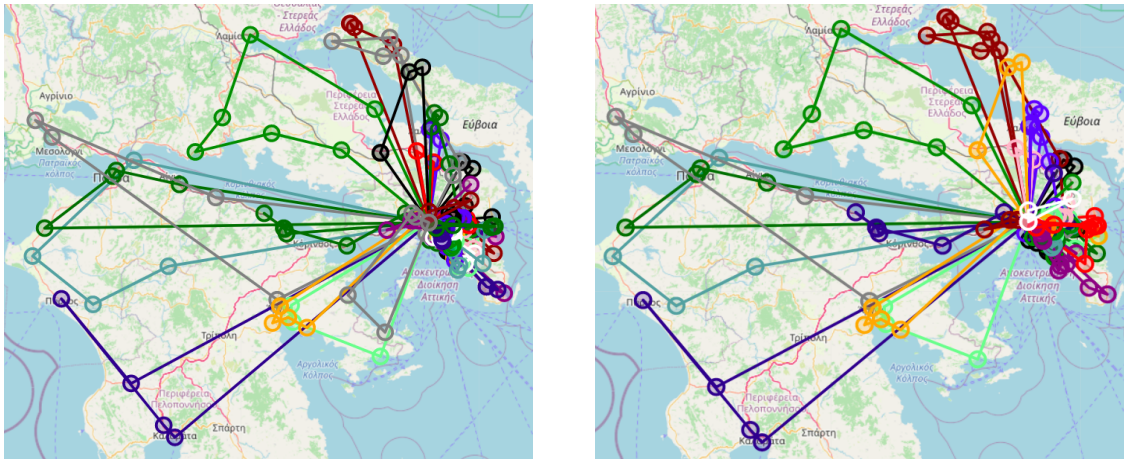


Figure 5.7: Routes found by our algorithm on instance 1



(a) Before matching (b) After matching

Figure 5.8: Routes found by OR-Tools on instance 1

The first thing we notice from the above maps is that in every route produced by our algorithm customers are routed in perfect order, which is expected since we optimize them by solving the TSP. On the contrary, the other algorithm often yields routes that visit customers in a bad order, with the most notable example of that being the blue route that goes northwest and then southeast again. Their algorithm also produces 9.1% infeasible routes with one of them being rural (the one near Epidaurus). This causes a chain reaction with a large impact in the total cost as our algorithm is forced to go multiple times in East Peloponnese, while their algorithm delivers those customers with fewer routes, because of that infeasible one.

Instance 2

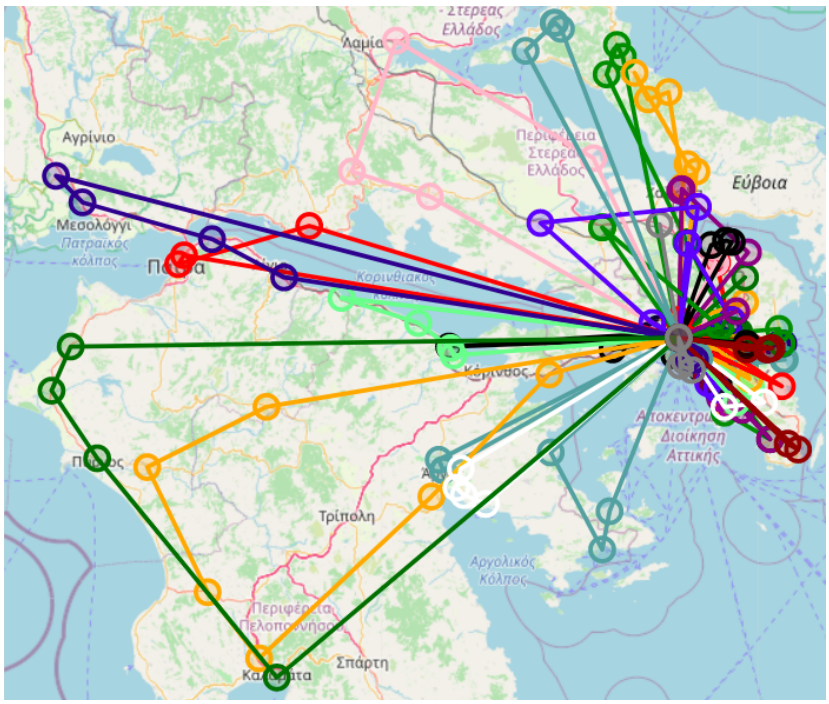
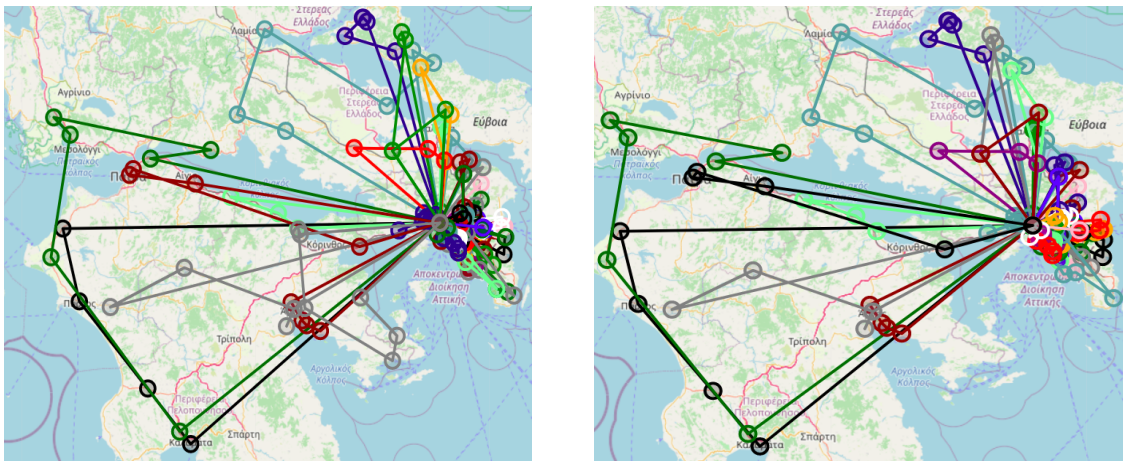


Figure 5.9: Routes found by our algorithm on instance 2



(a) Before matching

(b) After matching

Figure 5.10: Routes found by OR-Tools on instance 2

In this instance, again the OR-Tools provide a high percentage of infeasible routes with 7.8%. A large rural route that goes to Epidaurus and Kiato is infeasible which greatly reduces the total cost after the matching. Apart from that our algorithm probably should have assigned the customers of the green and orange routes in the Peloponnese in a slightly different way avoiding the edge between Kalamata and Pyrgos. However the same problem also exists in the OR-Tools solution, but in a smaller extent because of the existence of the infeasible route.

Instance 3

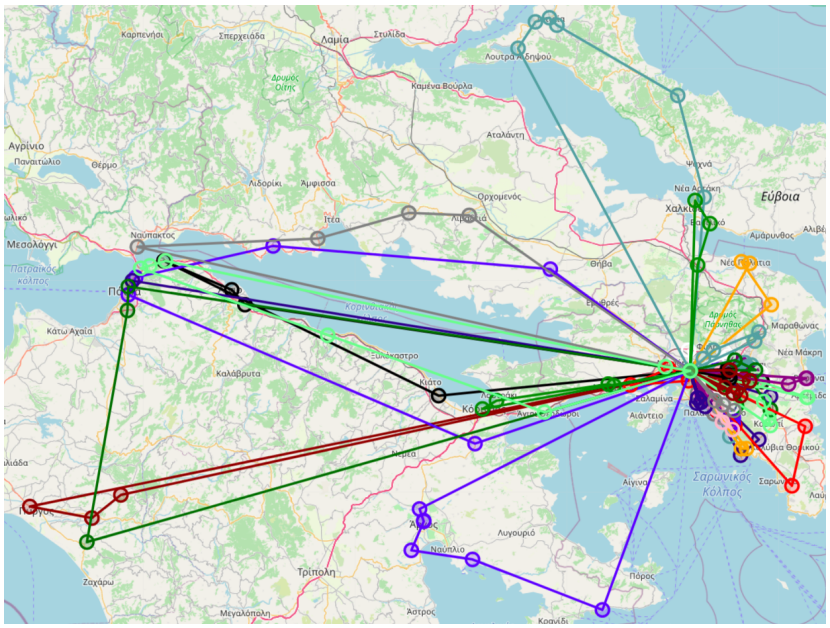
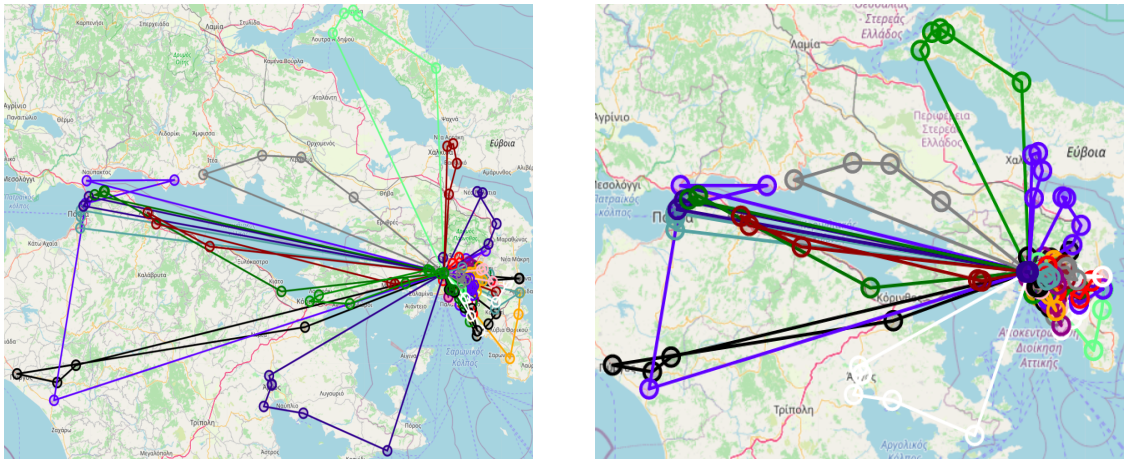


Figure 5.11: Routes found by our algorithm on instance 3



(a) Before matching (b) After matching

Figure 5.12: Routes found by OR-Tools on instance 3

In this instance 6.8% of the routes are infeasible after the matching. From the rural routes we can see that the one going to Corinth is remove after the matching. The routes both tools produce are very similar, but we have found worse routes in the Central Greece area, as we send two vehicles to deliver to customers there instead of one found by the OR-tools. This causes, our algorithm to have a 3% worse cost than the OR-Tools before the matching.

Instance 4

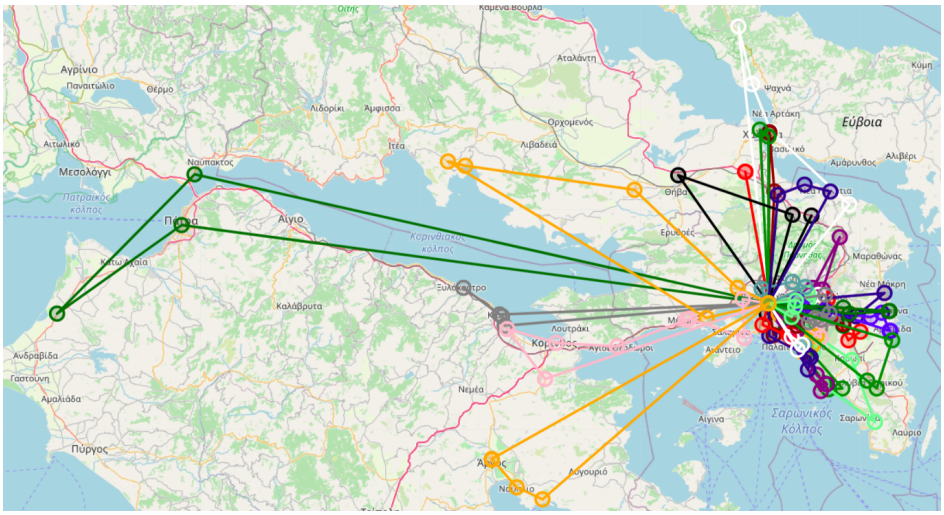


Figure 5.13: Routes found by Tabu Search on instance 4

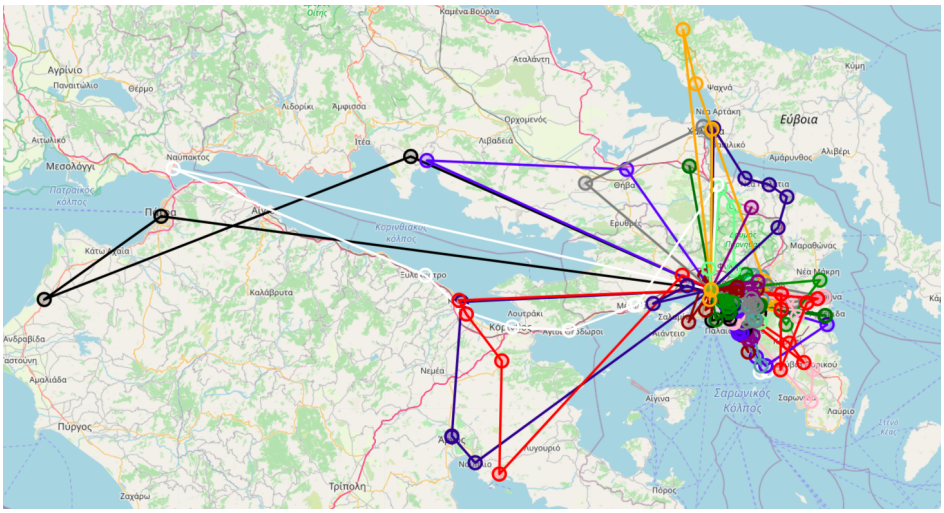
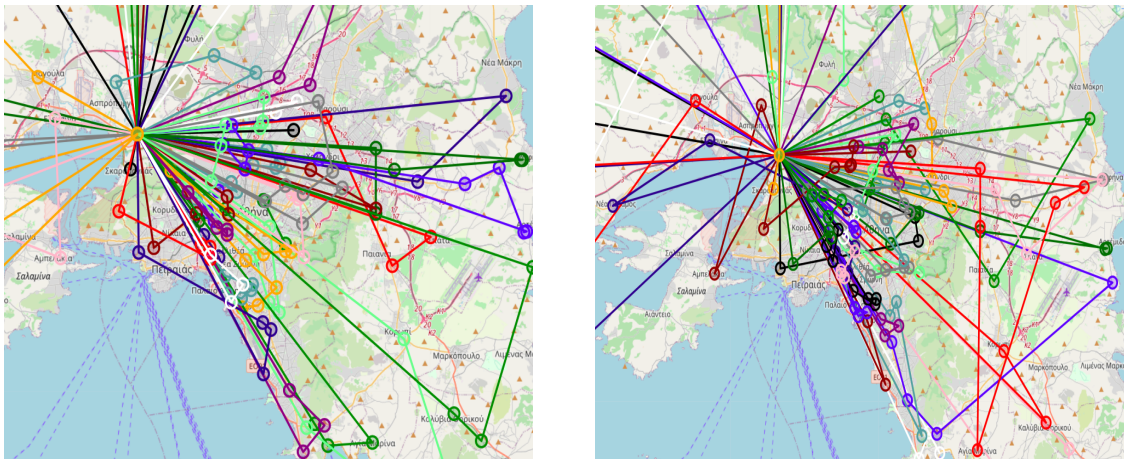


Figure 5.14: Routes found by OR-Tools on instance 4

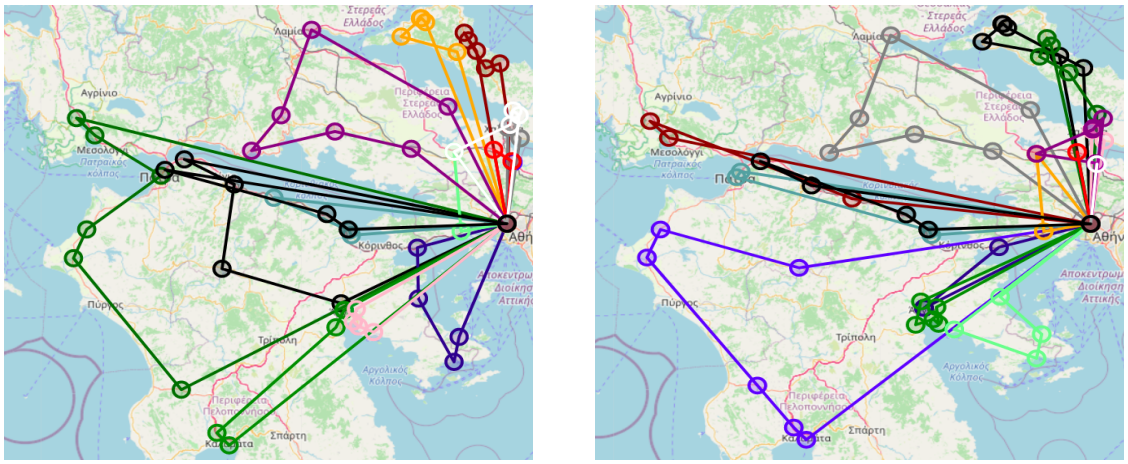
From figures 5.13 and 5.14 it is evident that the tabu search algorithm has routed the rural customers more efficiently as it avoids going to the far west of the map twice, which results in a much smaller distance overall. This is impressive if we also take into account the fact that 9.2% of the OR-Tools routes are infeasible. We also provide the routes for the Athens area, where we also seem to have partitioned the nodes better as the OR-tools provide some really bad routes like the red and blue that visit East Attica.



(a) Tabu search (b) OR-Tools

Figure 5.15: Routes found on instance 4 near Athens

Instance 5



(a) Tabu search (b) OR-Tools

Figure 5.16: Routes found on instance 5

This is the only instance where all of the OR-Tools' routes have been matched successfully to a vehicle. The routes produced by the two algorithms are very similar and don't require further analysis.

Chapter 6

Conclusion and future directions

In this thesis we have studied the Vehicle Routing Problem and its application. We proposed an algorithm that is targeted to a variation where all vehicles are compartmentalized and each compartment can hold up to one product, but our algorithm can also be used for the general VRP, especially for small instances.

As there is no public dataset for this particular problem we have used a set aimed to the general VRP with some modifications, alongside a few instances provided to as by a fuel distributing company. Our results for these instances give us good reason to believe that our algorithm find solutions with a cost close to optimal. We also tested our algorithm on a set of instances for the classic Vehicle Routing Problem, where we achieve to find solutions that are within 2% of the known optimal solutions.

This work can be continued in two main directions. Firstly, some of the internal characteristics of our algorithm could be re-examined, in order to make its performance even better. An example of this would be to modify the objective function so to accept infeasible solutions with penalized costs. This could lead to an exploration of unexplored regions of the solution space and possibly be a bridge between two local optima. Other things that could be modified are the structure of the tabu list or the reduction of the neighborhood size with an equivalent increase in the number of maximum iterations and maximum restarts.

Secondly, the algorithm could be extended to support more features that are commonly found in real-life scenarios. A usual constraint met in practice is the enforcement of time-windows on the delivery of the orders and the utilization of the vehicles. Another feature that is typically combined with the time-windows is the ability of vehicles to make more than one trip. Other common constraints include labeled vehicles that can only visit specific customers or can visit some with a higher cost and additional loading constraints related to the filling of the compartments.

Βιβλιογραφία

[Aart03] Emile Aarts, Emile HL Aarts and Jan Karel Lenstra, *Local search in combinatorial optimization*, Princeton University Press, 2003.

[Auge95] Philippe Augerat, *Approche polyèdrale du problème de tournées de véhicules*, Ph.D. thesis, Institut National Polytechnique de Grenoble-INPG, 1995.

[Bell62] Richard Bellman, "Dynamic programming treatment of the travelling salesman problem", *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 61–63, 1962.

[Bena16] Abdelaziz Benantar, Rachid Ouafi and Jaouad Boukachour, "A petrol station replenishment problem: new variant and formulation", *Logistics Research*, vol. 9, no. 1, p. 6, 2016.

[Brae16] Kris Braekers, Katrien Ramaekers and Inneke Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review", *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[Bran11] José Brandão, "A tabu search algorithm for the heterogeneous fixed fleet vehicle routing problem", *Computers & Operations Research*, vol. 38, no. 1, pp. 140–151, 2011.

[Brow81] Gerald G Brown and Glenn W Graves, "Real-time dispatch of petroleum tank trucks", *Management science*, vol. 27, no. 1, pp. 19–32, 1981.

[Cern85] Vladimír Černỳ, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm", *Journal of optimization theory and applications*, vol. 45, no. 1, pp. 41–51, 1985.

[Clar64] Geoff Clarke and John W Wright, "Scheduling of vehicles from a central depot to a number of delivery points", *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[Cord00] Jean-Francois Cordeau and Québec) Groupe d'études et de recherche en analyse des décisions (Montréal, *The VRP with time windows*, Groupe d'études et de recherche en analyse des décisions Montréal, 2000.

[Corn08] Fabien Cornillier, Fayez F Boctor, Gilbert Laporte and Jacques Renaud, "An exact algorithm for the petrol station replenishment problem", *Journal of the Operational Research Society*, vol. 59, no. 5, pp. 607–615, 2008.

[Corn09] Fabien Cornillier, Gilbert Laporte, Fayez F Boctor and Jacques Renaud, "The petrol station replenishment problem with time windows", *Computers & Operations Research*, vol. 36, no. 3, pp. 919–935, 2009.

[Dant59] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem", *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[Eksi09] Burak Eksioglu, Arif Volkan Vural and Arnold Reisman, "The vehicle routing problem: A taxonomic review", *Computers & Industrial Engineering*, vol. 57, no. 4, pp. 1472–1483, 2009.

[ElFa08] Abdellah El Fallahi, Christian Prins and Roberto Wolfler Calvo, "A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem", *Computers & Operations Research*, vol. 35, no. 5, pp. 1725–1741, 2008.

[Fran08] Peter M Francis, Karen R Smilowitz and Michal Tzur, "The period vehicle routing problem and its extensions", in *The vehicle routing problem: latest advances and new challenges*, pp. 73–102, Springer, 2008.

[Fuka06] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi De Aragão, Marcelo Reis, Eduardo Uchoa and Renato F Werneck, "Robust branch-and-cut-and-price for the capacitated vehicle routing problem", *Mathematical programming*, vol. 106, no. 3, pp. 491–511, 2006.

[Gend92] Michel Gendreau, Alain Hertz and Gilbert Laporte, "New insertion and postoptimization procedures for the traveling salesman problem", *Operations Research*, vol. 40, no. 6, pp. 1086–1094, 1992.

[Gend98] Michel Gendreau, Alain Hertz, Gilbert Laporte and Mihnea Stan, "A generalized insertion heuristic for the traveling salesman problem with time windows", *Operations Research*, vol. 46, no. 3, pp. 330–335, 1998.

[Gend08] Michel Gendreau, Jean-Yves Potvin, Olli Bräumlaysy, Geir Hasle and Arne Løkketangen, "Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography", in *The vehicle routing problem: latest advances and new challenges*, pp. 143–169, Springer, 2008.

[Glov98] Fred Glover and Manuel Laguna, "Tabu search", in *Handbook of combinatorial optimization*, pp. 2093–2229, Springer, 1998.

[Gold08] Bruce L Golden, Subramanian Raghavan and Edward A Wasil, *The vehicle routing problem: latest advances and new challenges*, vol. 43, Springer Science & Business Media, 2008.

[Held62] Michael Held and Richard M Karp, "A dynamic programming approach to sequencing problems", *Journal of the Society for Industrial and Applied mathematics*, vol. 10, no. 1, pp. 196–210, 1962.

[Kirk83] Scott Kirkpatrick, C Daniel Gelatt and Mario P Vecchi, "Optimization by simulated annealing", *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[Lin65] Shen Lin, "Computer solutions of the traveling salesman problem", *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[Osma93] Ibrahim Hassan Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", *Annals of operations research*, vol. 41, no. 4, pp. 421–451, 1993.

[Parr08] Sophie N Parragh, Karl F Doerner and Richard F Hartl, "A survey on pickup and delivery problems", *Journal für Betriebswirtschaft*, vol. 58, no. 2, pp. 81–117, 2008.

[Perr] Laurent Perron and Vincent Furnon, "OR-Tools".

[Pirl96] Marc Pirlot, "General local search methods", *European journal of operational research*, vol. 92, no. 3, pp. 493–511, 1996.

[Subr12] Anand Subramanian, Puca Huachi Vaz Penna, Eduardo Uchoa and Luiz Satoru Ochi, "A hybrid algorithm for the Heterogeneous Fleet Vehicle Routing Problem", *European Journal of Operational Research*, vol. 221, no. 2, pp. 285–295, 2012.

[Subr13] Anand Subramanian, Eduardo Uchoa and Luiz Satoru Ochi, "A hybrid algorithm for a class of vehicle routing problems", *Computers & Operations Research*, vol. 40, no. 10, pp. 2519–2531, 2013.

[Tail97] Éric Taillard, Philippe Badeau, Michel Gendreau, François Guertin and Jean-Yves Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows", *Transportation science*, vol. 31, no. 2, pp. 170–186, 1997.

[Toth02a] Paolo Toth and Daniele Vigo, *The vehicle routing problem*, SIAM, 2002.

[Toth02b] Paolo Toth and Daniele Vigo, "VRP with backhauls", in *The vehicle routing problem*, pp. 195–224, SIAM, 2002.

[Toth14] Paolo Toth and Daniele Vigo, *Vehicle routing: problems, methods, and applications*, SIAM, 2014.

[Ucho17] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal and Anand Subramanian, "New benchmark instances for the capacitated vehicle routing problem", *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017.