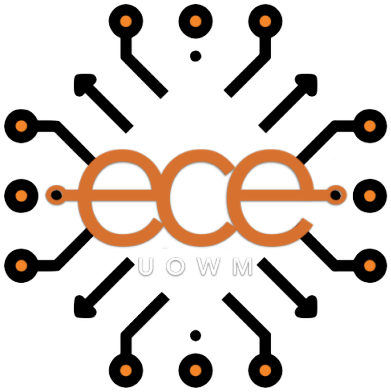


Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Αλγόριθμοι συνδυαστικής βελτιστοποίησης για το πρόβλημα της χωροθέτησης ηλεκτρικών φορτιστών

Μαρίνα Νούση (ΑΜ: 1873)

Επιβλέπων Καθηγητής: Νικόλαος Πλόσκας

Εργαστήριο Ευφυών Συστημάτων & Βελτιστοποίησης

26 Ιουνίου 2025

Περίληψη

Στην παρούσα διπλωματική εργασία μελετάται το πρόβλημα χωροθέτησης ηλεκτρικών φορτιστών, με επίκεντρο την παραλλαγή του ετερογενούς προβλήματος p-μέσου. Το πρόβλημα στοχεύει στον εντοπισμό των κατάλληλων θέσεων για την εγκατάσταση p σταθμών φόρτισης, με σκοπό την ελαχιστοποίηση της συνολικής απόστασης που πρέπει να διανύσουν οι χρήστες για να εξυπηρετηθούν, λαμβάνοντας υπόψη τα διαφορετικά χαρακτηριστικά κάθε εγκατάστασης και τις ιδιαιτερότητες της εξυπηρέτησης των χρηστών. Αρχικά πραγματοποιείται η διατύπωση και μοντελοποίηση του προβλήματος, το οποίο αναπαρίσταται ως ακέραιο γραμμικό μοντέλο και επιλύεται με τον εμπορικό λύτη Gurobi. Παράλληλα, υλοποιούνται ευρετικοί και μεθευρετικοί αλγόριθμοι, οι οποίοι χρησιμοποιούνται τόσο για παραγωγή αρχικών λύσεων όσο και για συγκριτική αξιολόγηση. Επιπλέον, αναπτύχθηκε αλγόριθμος διακλάδωσης και περιορισμού με τέσσερεις διαφορετικές τεχνικές. Οι πρώτες δύο αφορούν τον τρόπο εξερεύνησης κλαδιών και οι υπόλοιπες δύο τη στρατηγική επιλογής κόμβων προς διακλάδωση, με στόχο τη βελτίωση του χρόνου εκτέλεσης και της ποιότητας των λύσεων. Η πειραματική μελέτη περιλαμβάνει τη σύγκριση των παραπάνω τεχνικών με βάση την υπολογιστική απόδοση και την ακρίβεια των αποτελεσμάτων. Τα αποτελέσματα δείχνουν ότι πιο αποδοτική προσέγγιση είναι η Branch & Bound με BestFS και ευρετική Min Distance, με μέσο χρόνο εκτέλεσης 1.25 δευτερόλεπτα, υπερτερώντας των υπόλοιπων αλγορίθμων.

Λέξεις κλειδιά: χωροθέτηση ηλεκτρικών φορτιστών, πρόβλημα p μέσου, διακλάδωση και περιορισμός, ευρετικές μέθοδοι, μεθευρετικές μέθοδοι, γενετικός αλγόριθμος.

Abstract

This thesis investigates the problem of locating electric vehicle chargers, focusing on a variant of the heterogeneous p-median problem. The objective is to identify suitable locations for the installation of p charging stations, aiming to minimize the total distance users must travel to be served, while considering the distinct characteristics of each facility and the particularities of user service. Initially, the problem is formulated and modeled as an integer linear program, which is solved using the commercial solver Gurobi. Concurrently, heuristic and metaheuristic algorithms are implemented to generate initial solutions as well as for comparative evaluation. Additionally, a Branch and Bound algorithm was developed employing four different strategies: the first two concern the method of branch exploration and the latter two pertain to the node selection strategy for branching, with the goal of improving both execution time and solution quality. The experimental study includes a comparison of these techniques based on computational performance and solution accuracy. The results demonstrate that the most efficient approach is the Branch & Bound method with BestFS combined with the Min Distance heuristic, achieving an average execution time of 1.25 seconds and outperforming the other algorithms.

Keywords: charging station location problem, p-median problem, Branch and Bound, heuristic methods, metaheuristic methods, genetic algorithm.

Δήλωση Πνευματικών Δικαιωμάτων

Δήλωση Πνευματικών Δικαιωμάτων Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Αλγόριθμοι συνδυαστικής βελτιστοποίησης για το πρόβλημα της χωροθέτησης ηλεκτρικών φορτιστών" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Νικόλαου Πλόσκα αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

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

Υπογραφή Φοιτητή

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος αλγορίθμων

Κατάλογος Πινάκων

Κεφάλαιο 1

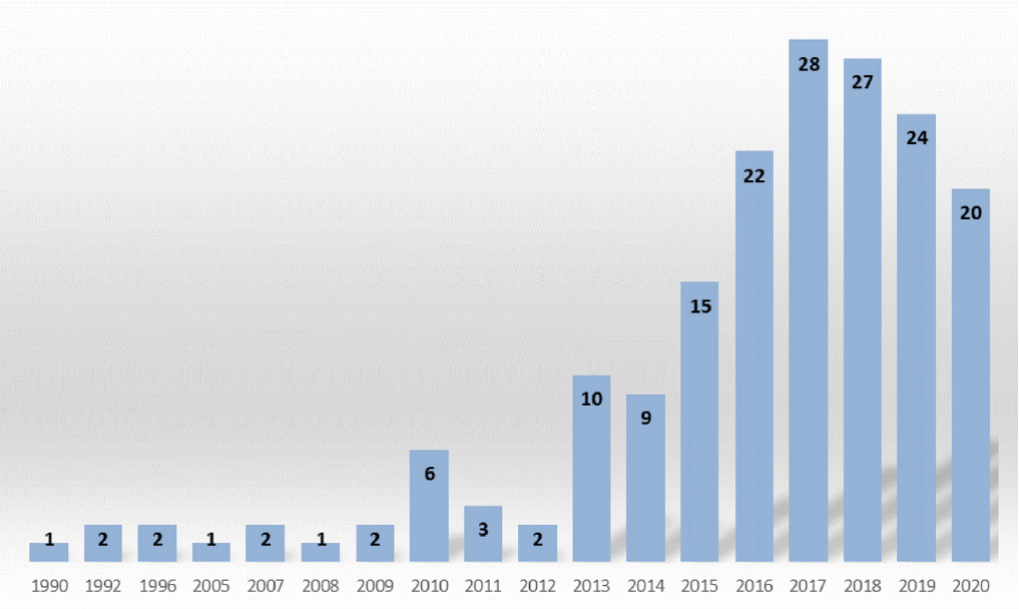
Εισαγωγή

1.1 Ορισμός του προβλήματος

Το πρόβλημα χωροθέτησης σταθμών φόρτισης (Charging Station Location Problems) είναι μια υποκατηγορία των προβλημάτων χωροθέτησης εγκαταστάσεων (Facility Location Problems) με βασική διαφορά τον περιορισμό του εύρους οδήγησης που έχουν τα ηλεκτρικά αυτοκίνητα [2]. Τα τελευταία χρόνια, η χρήση των ηλεκτρικών αυτοκινήτων αυξάνεται και έτσι οι έρευνες για το πρόβλημα χωροθέτησης σταθμών φόρτισης με σκοπό τον ανεφοδιασμό των ηλεκτρικών αυτοκινήτων προκαλούν ολοένα και περισσότερο ενδιαφέρον, όπως φαίνεται και στο Σχήμα 1.1.

Το πρόβλημα μπορεί να χωριστεί σε 4 βασικές κατηγορίες ανάλογα με την αντικειμενική του συνάρτηση. Αυτές είναι το πρόβλημα κάλυψης συνόλου, το πρόβλημα μέγιστης κάλυψης, το p κέντρου και p μέσου. Πιο συγκεκριμένα, το πρόβλημα κάλυψης συνόλου επικεντρώνεται στην εύρεση του ελάχιστου πλήθους σταθμών φόρτισης που χρειάζεται να χωροθετηθούν ώστε να εξυπηρετείται όλο το σύνολο των οδηγών. Ενώ το πρόβλημα μέγιστης κάλυψης στοχεύει στην κατάλληλη τοποθέτηση συγκεκριμένου πλήθους σταθμών φόρτισης με σκοπό τη μέγιστη κάλυψη των οδηγών. Επίσης, το πρόβλημα p κέντρου τοποθετεί συγκεκριμένο αριθμό σταθμών φόρτισης και επικεντρώνεται στο να καλυφθούν όλες οι απαιτήσεις ελαχιστοποιώντας τη μέγιστη απόσταση που πρέπει να διανύσουν οι οδηγοί για να φτάσουν στην πλησιέστερη εγκατάσταση. Τέλος, το πρόβλημα p μέσου το οποίο θα εξετάσουμε, αφορά τον εντοπισμό των καλύτερων θέσεων για να τοποθετηθούν οι p σταθμοί φόρτισης ώστε να ελαχιστοποιηθεί η συνολική απόσταση που χρειάζεται να διανύσουν οι οδηγοί για να φτάσουν στον πλησιέστερο σταθμό [3].

Σχήμα 1.1: Έρευνες για το πρόβλημα χωροθέτησης σταθμών φόρτισης ανά τα χρόνια [1].



1.2 Κίνητρα και στόχοι υλοποίησης

Οι περιβαλλοντικές αλλαγές που επιβαρρύνουν τον πλανήτη, σε συνδυασμό με τη σταδιακή εξάντληση των αποθεμάτων πετρελαίου, καθιστούν αναγκαία την υιοθέτηση περισσότερο βιώσιμων και φιλικών προς το περιβάλλον μέσων μετακίνησης. Η ηλεκτροκίνηση προσφέρει μια αποτελεσματική λύση προς αυτή την κατεύθυνση, καθώς συμβάλλει στη μείωση του περιβαλλοντικού αποτυπώματος των μετακινήσεών. Ωστόσο, ένα από τα σημαντικότερα εμπόδια στην ευρύτερη υιοθέτηση των ηλεκτρικών οχημάτων είναι η περιορισμένη διαθεσιμότητα υποδομών φόρτισης. Η έλλειψη αυτή προκαλεί αβεβαιότητα στους οδηγούς, οι οποίοι συχνά διστάζουν να προχωρήσουν στην αγορά, παρόλο που αναγνωρίζουν τα οφέλη της ηλεκτροκίνησης. Επομένως, είναι κρίσιμο να εξεταστεί διεξοδικά το πρόβλημα της χωροθέτησης φορτιστών, ώστε τα αποτελέσματα από τη συγκριτική αξιολόγηση σχετικών αλγορίθμων να αξιοποιηθούν αποτελεσματικά σε πραγματικές συνθήκες, με στόχο την αντιμετώπιση του προβλήματος εγκατάστασης σταθμών φόρτισης.

1.3 Διάρθρωση κειμένου

Η παρούσα εργασία χωρίζεται σε πέντε κεφάλαια. Το δεύτερο κεφάλαιο περιλαμβάνει τη διατύπωση και τη μοντελοποίηση του προβλήματος. Στη συνέχεια,

παρουσιάζονται οι μέθοδοι επίλυσης, με ανάλυση των ακριβών, προσεγγιστικών, ευρετικών και μεθευρετικών μεθόδων. Το τρίτο κεφάλαιο επικεντρώνεται στην υλοποίηση των αλγορίθμων και την εφαρμογή τους, συνοδευόμενη από ψευδοκώδικα για κάθε περίπτωση. Στο τέταρτο κεφάλαιο περιγράφεται η πειραματική διαδικασία και διεξάγεται η υπολογιστική μελέτη, με στόχο τη σύγκριση των αλγορίθμων. Τέλος, στο πέμπτο κεφάλαιο παρουσιάζονται τα συμπεράσματα της μελέτης.

Κεφάλαιο 2

Βιβλιογραφική ανασκόπηση

2.1 Διατύπωση του προβλήματος p μέσου

Το πρόβλημα p μέσου αφορά την τοποθέτηση p εγκαταστάσεων, με σκοπό την ελαχιστοποίηση της συνολικής απόστασης μεταξύ των κόμβων ζήτησης και της πλησιέστερης επιλεγμένης εγκατάστασης. Χρονολογείται απο το έργο του Hakimi (1964, 1965) [4] και είναι μια μόνο απο τις κατηγορίες των προβλημάτων χωροθέτησης. Ενδεικτικά υπάρχουν ακόμη τα προβλήματα τοποθέτησης εγκαταστάσεων με ή χωρίς χωρητικότητα, το p κέντρου, τα προβλήματα covering και anti-covering. Το πρόβλημα p μέσου που εξετάζουμε πρόκειται για ένα NP-hard πρόβλημα όταν αναπαρίσταται με τη μορφή γενικού γράφου, δηλαδή δεν λύνεται σε πολυωνυμικό χρόνο. Όμως, όπως έδειξαν οι Hakimi και Kariv [5] στην περίπτωση που αναπαρασταθεί σε μορφή δέντρου, το πρόβλημα λύνεται σε πολυωνυμικό χρόνο [6].

Μία εκδοχή του προβλήματος, αυτή που θα μελετήσουμε, είναι το ετερογενές p-median πρόβλημα στο οποίο οι p εγκαταστάσεις που επιλέγονται να τοποθετηθούν στο δίκτυο διαφέρουν μεταξύ τους ως προς τα χαρακτηριστικά τους και δεν είναι ομοιογενείς. Για παράδειγμα, μπορεί να διαφοροποιούνται ως προς το μέγεθος, το λειτουργικό κόστος ή τα επιτρεπτά όρια απόστασης από τους κόμβους ζήτησης. Η προσέγγιση αυτή αναδεικνύει τη σημασία της ετερογένειας στις εγκαταστάσεις, ώστε το μοντέλο να αντανακλά καλύτερα τις συνθήκες του πραγματικού κόσμου, όπου οι ανάγκες και οι προτιμήσεις των κόμβων ζήτησης ποικίλλουν [7]. Τα βασικά δεδομένα του προβλήματος περιλαμβάνουν: N τον αριθμό των υποψήφιων κόμβων εγκατάστασης, CL τον αριθμό των κόμβων ζήτησης και P το πλήθος των εγκαταστάσεων που πρόκειται να λειτουργήσουν.

2.2 Μοντελοποίηση

Το πρόβλημα χωροθέτησης σταθμών φόρτισης ηλεκτρικών οχημάτων έχει προσεγγιστεί με διάφορους τρόπους, ανάλογα με την οπτική γωνία που υιοθετείται [8]. Μια από τις βασικές προσεγγίσεις βασίζεται στην οπτική του διαχειριστή του δικτύου διανομής, ο οποίος είναι υπεύθυνος για την παροχή ηλεκτρικής ενέργειας. Στο πλαίσιο αυτό, η μοντελοποίηση μπορεί να εστιάζει στο ενεργό κόστος απώλειας ισχύος [9, 10, 11], στο κόστος απώλειας τάσης, καθώς και στο κόστος απώλειας άεργου ισχύος [11, 12]. Μια δεύτερη προσέγγιση αφορά τους ιδιοκτήτες των σταθμών φόρτισης, οι οποίοι επιδιώκουν να εντοπίσουν τοποθεσίες που μεγιστοποιούν τα έσοδά τους, ελαχιστοποιώντας παράλληλα το απαιτούμενο επενδυτικό κόστος [13, 14, 15]. Τέλος, υπάρχει και η προσέγγιση από την πλευρά των χρηστών, δηλαδή των οδηγών ηλεκτρικών οχημάτων. Σε αυτή την περίπτωση, τα μοντέλα δίνουν έμφαση στο κόστος πρόσβασης, στο κόστος μετακίνησης από την τρέχουσα θέση του οδηγού έως τον πλησιέστερο σταθμό φόρτισης [16], καθώς και στο κόστος που συνεπάγεται ο χρόνος αναμονής στον σταθμό [17].

Στόχος όλων των προσεγγίσεων αυτού του είδους προβλημάτων παραμένει ο εντοπισμός των p μονάδων που θα χωροθετηθούν, απο ένα σύνολο υποψήφιων κόμβων προς εξυπηρέτηση. Ωστόσο, η συνδυαστική χρήση διαφορετικών προσεγγίσεων μοντελοποίησης μπορεί να προσφέρει περισσότερο αξιόπιστα και ρεαλιστικά αποτελέσματα, καθώς επιτρέπει την πληρέστερη αποτύπωση της πολυπλοκότητας και των παραμέτρων που χαρακτηρίζουν τις συνθήκες του πραγματικού κόσμου. Ακολουθεί η διατύπωση για το p μέσου πρόβλημα, ως πρόβλημα ακέραιου γραμμικού προγραμματισμού.

Στο πρόβλημα υπάρχουν τα εξής σύνολα: $N = \{1, 2, ..., n\}$ που εκφράζει τους υποψήφιους κόμβους εξυπηρέτησης, $CL = \{1, 2, ..., cl\}$ για το σύνολο των κόμβων ζήτησης και το σύνολο $F = \{2, 3\}$ εκφράζει τους τύπους φορτιστών που μπορούν να τοποθετηθούν σε κάθε εγκατάσταση που ανοίγει και πρόκειται για φορτιστές Level 2 και 3 αντίστοιχα. Επίσης, ο αριθμός P εκφράζει το πλήθος των εγκαταστάσεων προς χωροθέτηση για κάθε πρόβλημα. Συνδυάζοντας το σύνολο N και CL, ορίζεται η παράμετρος c_{ki}, ένας πίνακας διαστάσεων $cl \times n$, ο οποίος εκφράζει την ελάχιστη απόσταση μεταξύ κάθε πελάτη k και κάθε υποψήφιας εγκατάστασης i. Ως

προς τις μεταβλητές μοντελοποίησης, υπάρχει μια ακέραια μεταβλητή x_{ij} η οποία εκφράζει το πλήθος των φορτιστών Level 2 και 3 που εγκαθίστανται στον κόμβο i και θεωρούμε πως το μέγιστο πλήθος φορτιστών σε έναν κόμβο είναι 5. Ακόμη, έχουμε μια δυαδική μεταβλήτη y_{ki} η οποία δείχνει αν ο πελάτης k εξυπηρετείται απο τον κόμβο i οπότε και έχει τιμή ίση με 1 ($y_{ki} = 1$) αλλιώς έχει τιμή 0 ($y_{ki} = 0$). Τέλος, η δυαδική μεταβλητή z_i δηλώνει αν έχει ανοίξει ($z_i = 1$) ή όχι ($z_i = 0$) μια εγκατάσταση φορτιστών στον κόμβο i.

Η αντικειμενική συνάρτηση του προβλήματος βελτιστοποίησης στοχεύει στην ελαχιστοποίηση της συνολικής απόστασης μεταξύ κόμβων ζήτησης και κόμβων εξυπηρέτησης και είναι:

$$min \sum_{k \in CL} \sum_{i=1}^{N} c_{ki} y_{ki}$$

με περιορισμούς:

$$\sum_{j \in F} x_{ij} \leq \theta_i, \forall i \in N \quad (1)$$

$$\sum_{j \in F} l_j x_{ij} \leq C_i, \forall i \in N \quad (2)$$

$$\sum_{j \in F} x_{ij} \geq z_i, \forall i \in N \quad (3)$$

$$\sum_{j \in F} x_{ij} \leq \theta_i z_i, \forall i \in N \quad (4)$$

$$\sum_{i=1}^{n} z_i = P \quad (5)$$

$$\sum_{i}^{n} y_{ki} = 1, \forall k \in CL \quad (6)$$

$$\sum_{i=1}^{n} y_{ki} \leq z_i, \forall k \in CL \quad (7)$$

$$x_{if_1} + x_{jf_2} \leq 1, \forall (i, j, f_1, f_2) \in C1 \quad (8)$$

$$\sum_{i=1}^{n} d_{ki} y_{ki} \leq D, \forall k \in CL \quad (9)$$

$$\sum_{i=1}^{n} a x_{i2} + b x_{i3} \geq \sum_{k \in CL} y_{ki}, \forall i \in N \quad (10)$$

$$\sum_{i=1}^{n} fz_i + gx_{i2} + hx_{i3} \leq budget \quad (11)$$

$$C1 = \{(i, j, f_1, f_2) \mid i, j \in N, f_1, f_2 \in F, D[i, j] \leq d1_{f_1 f_2}\} \quad (12)$$

$$x_{ij} \in [0, 5], \forall i \in N, \forall j \in F \quad (13)$$

$$y_{ki} \in \{0, 1\}, \forall k \in CL, \forall i \in N \quad (14)$$

$$z_i \in \{0, 1\}, \forall i \in N \quad (15)$$

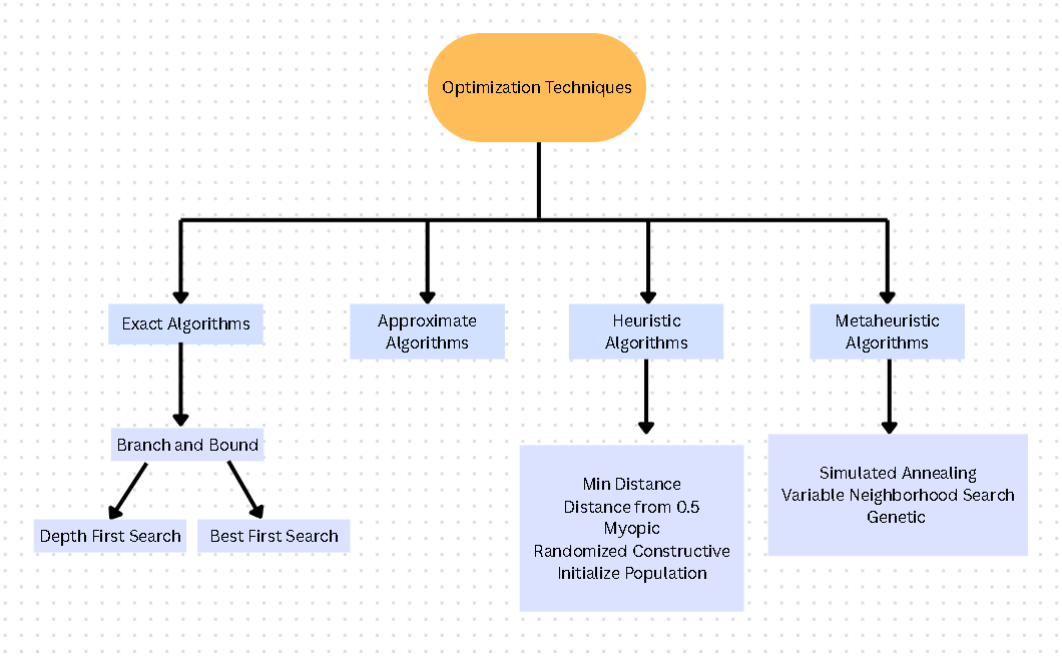
Ακολουθεί επεξήγηση για κάθε έναν περιορισμό. Ο πρώτος περιορισμός εξασφαλίζει ότι το άθροισμα των φορτιστών που τοποθετούνται σε έναν κόμβο δεν ξεπερνά το πλήθος των διαθέσιμων θέσεων φόρτισης (θ) που έχει ο εκάστοτε κόμβος. Ο δεύτερος περιορισμός ορίζει ότι η συνολική ισχύς των φορτιστών που θα τοποθετηθούν σε έναν κόμβο δεν πρέπει να ξεπερνά τη διαθέσιμη ισχύ της εγκατάστασης (C) στον κόμβο αυτό, κάνοντας την παραδοχή ότι οι φορτιστές Level 2 καταναλώνουν ισχύ 22kW, ενώ οι Level 3 καταναλώνουν 100kW. Ο τρίτος περιορισμός εξασφαλίζει ότι σε μια ανοιχτή εγκατάσταση πρέπει να υπάρχει τουλάχιστον ένας φορτιστής. Αντίστοιχα, ο τέταρτος περιορισμός ορίζει οτι αν ένας κόμβος εξυπηρέτησης δεν είναι ανοιχτός, τότε δεν πρέπει να υπάρχουν φορτιστές σε αυτόν. Απο τον πέμπτο περιορισμό επιτυγχάνεται ο αριθμός των ανοιχτών κόμβων εξυπηρέτησης να ισούται με τον επιθυμητό αριθμό κόμβων εξυπηρέτησης P. Επίσης, ο έκτος περιορισμός δηλώνει ότι κάθε κόμβος ζήτησης θα εξυπηρετείται απο έναν κόμβο εξυπηρέτησης. Ο έβδομος περιορισμός εξασφαλίζει ότι ένας κόμβος ζήτησης μπορεί να εξυπηρετηθεί απο έναν κόμβο εξυπηρέτησης μόνο αν σε αυτόν υπάρχει ανοιχτή εγκατάσταση. Ο όγδοος περιορισμός εγγυάται ότι δεν θα ανοίξουν δύο εγκαταστάσεις ταυτόχρονα σε κόμβους εξυπηρέτησης που έχουν απόσταση μικρότερη απο μια ορισμένη τιμή. Επιπλέον, με τον ένατο περιορισμό εξασφαλίζεται ότι η απόσταση μεταξύ κόμβου ζήτησης και του κόμβου εξυπηρέτησης δεν πρέπει να υπερβαίνει τα 5 km. Στο δέκατο περιορισμό εξασφαλίζουμε ότι οι κόμβοι ζήτησης που εξυπηρετούνται απο μια εγκατάσταση φορτιστών κατά τη διάρκεια της ημέρας δεν ξεπερνούν το πλήθος αυτών που μπορεί να εξυπηρετήσει η εγκατάσταση. Για τον υπολογισμό της μέγιστης δυναμικότητας εξυπηρέτησης κάθε εγκατάστασης, υιοθετείται η παραδοχή ότι ένα όχημα απαιτεί 2 ώρες φόρτισης σε φορτιστή τύπου 2, ενώ σε φορτιστή τύπου 3 αρκούν 30 λεπτά. Με βάση αυτή τη σύμβαση, καθορίζεται πόσα οχήματα μπορεί

να εξυπηρετήσει κάθε φορτιστής εντός του ημερήσιου χρονικού ορίου λειτουργίας. Ο ενδέκατος περιορισμός εξασφαλίζει ότι το άθροισμα του κόστους εγκατάστασης και τοποθέτησης των φορτιστών δεν ξεπερνά το διαθέσιμο οικονομικό ποσό σε κάθε περίπτωση (60,000 ευρώ). Συναντάμε το σύνολο C1 που περιέχει τους κόμβους εξυπηρέτησης των οποίων οι αποστάσεις ανα δύο είναι μικρότερες απο μια ορισμένη επιτρεπτή απόσταση d1. Οι τρείς τελευταίοι περιορισμοί δείχνουν πως η μεταβλητή x_{ij} είναι ακέραια, ενώ οι y_{ki} και z_i είναι δυαδικές. Στο μοντέλο μας έχουμε θεωρήσει ότι σε κάθε κόμβο εξυπηρέτησης μπορεί να υπάρχουν φορτιστές Level 2 ή 3 αντίστοιχα. Τέλος θεωρήσαμε ότι το κόστος για την εγκατάσταση σταθμού φόρτισης σε έναν κόμβο είναι 100 ευρώ, το κόστος κάθε φορτιστή Level 2 και 3 ανέρχεται στα 2000 και 6000 ευρώ αντίστοιχα. Επίσης, στα πλαίσια αυτής της μελέτης θεωρούμε τις σταθερές να έχουν αυτές τις τιμές: $a = 12$, $b = 48$, $f = 100$, $g = 2000$, $h = 6000$.

2.3 Μέθοδοι επίλυσης

Η επίλυση του προβλήματος χωροθέτησης σταθμών ηλεκτρικής φόρτισης αποτελεί μια σύνθετη πρόκληση, παρόμοια με εκείνες που προκύπτουν σε άλλα προβλήματα χωροθέτησης με εφαρμογές στον πραγματικό κόσμο. Η πολυπλοκότητα του προβλήματος απορρέει από παράγοντες όπως το μεγάλο πλήθος μεταβλητών και περιορισμών, η στοχαστική φύση ορισμένων παραμέτρων, καθώς και η πιθανή μη γραμμικότητα του μοντέλου. Αυτά τα χαρακτηριστικά καθιστούν την εύρεση της βέλτιστης λύσης ιδιαίτερα απαιτητική υπολογιστικά. Στη σύγχρονη έρευνα, η έμφαση δίνεται όλο και περισσότερο στην εύρεση εφικτών και ποιοτικών προσεγγιστικών λύσεων, χωρίς απαραίτητα να εξασφαλίζεται η αυστηρή μαθηματική σύγκλιση [18]. Προς αυτή την κατεύθυνση, έχουν αναπτυχθεί διάφορες μεθοδολογίες που αποσκοπούν τόσο στη βελτίωση της ποιότητας των λύσεων όσο και στη μείωση του υπολογιστικού κόστους. Στο Σχήμα 2.1 παρουσιάζεται ένα διάγραμμα με τις μεθόδους που υιοθετήθηκαν στην παρούσα μελέτη. Στη συνέχεια, παρουσιάζονται συγκεκριμένες ακριβείς, προσεγγιστικές, ευρετικές και μεθευρετικές τεχνικές που έχουν εφαρμοστεί σε σχετικές μελέτες της υπάρχουσας βιβλιογραφίας.

Σχήμα 2.1: Μέθοδοι επίλυσης για τη μελέτη στο πρόβλημα χωροθέτησης σταθμών φόρτισης.



2.3.1 Ακριβείς μέθοδοι

Η κλασσική προσέγγιση για την επίλυση προβλημάτων χωροθέτησης με ακριβείς μεθόδους είναι με τη χρήση Branch and Bound αλγορίθμου. Σε αυτόν μπορούν να ενσωματωθούν cutting planes [19] ή column generation [20] ώστε να αντιμετωπιστεί πιο αποδοτικά το πρόβλημα. Ο αλγόριθμος Branch and Price που χρησιμοποίησαν οι Yildiz et al. [21] εξετάζει τη διακλάδωση μόνο των μεταβλητών τοποθέτησης όπου η γενίκευση στηλών χρησιμοποιείται για να υπολογίσει ένα άνω όριο για κάθε κόμβο του δέντρου. Στον αλγόριθμο αυτό, εξετάζουν την ανοχή του οδηγού απο το να αποκλίνει απο την κύρια διαδρομή του με σκοπό να φτάσει σε σταθμό φόρτισης. Ακόμη, συναντάται συχνά στη βιβλιογραφία και η χρήση τεχνικών αποσύνθεσης όπως η αποσύνθεση Benders. Με αυτή τη μέθοδο το πρόβλημα χωρίζεται σε δύο μέρη, το κύριο πρόβλημα και το υποπρόβλημα, με το πρώτο να έχει τις κύριες μεταβλητές και το δεύτερο τις βοηθητικές. Διαχωρίζοντας το πρόβλημα κατά αυτόν τον τρόπο, παρατηρείται ότι η λύση γίνεται πιο αποδοτική [2] [22].

2.3.2 Προσεγγιστικές μέθοδοι

Προβλήματα χωροθέτησης σταθμών ηλεκτρικής φόρτισης που περιλαμβάνουν μη γραμμικά στοιχεία σε στοχαστικά μοντέλα ή δομές διπλού επιπέδου (bi-level) οδη-

γούν σε μη κυρτά προβλήματα. Σε τέτοιες περιπτώσεις απαιτούνται ειδικές μέθοδοι επίλυσης, με στόχο την εύρεση αξιόλογων προσεγγιστικών λύσεων. Οι πιο συνηθισμένες τεχνικές βασίζονται στην ανισότητα Bonferroni, στη μερική δειγματοληπτική προσέγγιση (Partial Sample Approximation – PSA) των τυχαίων μεταβλητών μέσω δειγματοληψίας Monte Carlo, καθώς και στη θεωρία ουρών [2].

2.3.3 Ευρετικές μέθοδοι

Πρόκειται για τις πιο διαδεδομένες μεθόδους επίλυσης προβλημάτων χωροθέτησης, οι οποίες βρίσκουν γρήγορα καλές λύσεις καθώς δεν εξερευνούν όλο το χώρο λύσεων. Παρόλο αυτά δεν εγγυώνται τη βέλτιστη λυση [23]. Οι άπληστοι ευρετικοί αλγόριθμοι, που επιλέγουν κάθε φορά την καλύτερη τοπική επιλογή, παρουσιάζουν καλή ποιότητα λύσεων σε σύντομο υπολογιστικό χρόνο [24, 25]. Οι Lim και Kuby [26] χρησιμοποίησαν έναν άπληστο ευρετικό αλγόριθμο ώστε να ξεπεράσουν το πρόβλημα με το μεγάλο συνδυασμό σταθμών φόρτισης σε πραγματική μελέτη για την περίπτωση της Φλόριντα. Ένας ακόμη γνωστός ευρετικός αλγόριθμος είναι αυτός της τοπικής αναζήτησης, όπου προσπαθεί να βελτιώσει τοπικά τη λύση απο την οποία ξεκίνησε.

2.3.4 Μεθευρετικές μέθοδοι

Οι μεθευρετικοί αλγόριθμοι χρησιμοποιούν συνδυασμό τεχνικών και στρατηγικών ώστε να αποφύγουν τα τοπικά βέλτιστα κατά την επίλυση του προβλήματος [27]. Οι γενετικοί αλγόριθμοι [28, 29, 30, 31] είναι αυτοί που έχουν χρησιμοποιηθεί περισσότερο σύμφωνα με τη βιβλιογραφία και εξετάζουν σημαντικά χαρακτηριστικά τους όπως η αναπαράσταση του χρωμοσώματος, η ρύθμιση του αρχικού πληθυσμού και η καταλληλότητα των τελεστών μετάλλαξης και διασταυρώσεων. Μερικές ακόμη απο τις μεθευρετικές μεθόδους που έχουν χρησιμοποιηθεί για την επίλυση αυτού του είδους προβλημάτων είναι ο αλγόριθμος Simulated Annealing [32, 33] όπου μιμείται τη διαδικασία της προσομοιωμένης ανόπτησης, όπου ένα υλικό θερμαίνεται και ψύχεται αργά, ώστε να ξεπεράσει προβλήματα τοπικών ελαχίστων. Ακόμη, ο αλγόριθμος Adaptive Large Neighborhood Search [34] αναζητά μεγάλες γειτονιές λύσεων και εφαρμόζει πολλαπλές στρατηγικές γειτονιάς με σκοπό να μειώσει το ''range anxiety'', δηλαδή την ανησυχία των οδηγών για τη διαδρομή που πρέπει να α-

κολουθήσουν ώστε να εντοπίσουν σταθμό φόρτισης. Συναντάμε επίσης αλγορίθμους που βασίζονται στη χαλάρωση Lagrangean [35], οι οποίοι εστιάζουν στη χαλάρωση των περιορισμών που αφορούν τη δυναμικότητα των σταθμών φόρτισης. Με την αξιοποίηση των πολλαπλασιαστών Lagrangean, το πρόβλημα καθίσταται πιο απλοποιημένο. Τέλος, ο αλγόριθμος Tabu Search [36] βελτιώνει σταδιακά τη λύση μέσω αναζήτησης σε περιοχές γειτονικών λύσεων, αποφεύγοντας την επιστροφή σε λύσεις που έχει ήδη επισκεφθεί. Στο συγκεκριμένο άρθρο, υπήρχαν συνθήκες αβεβαιότητας σχετικά με την εμβέλεια των οχημάτων οι οποίες αντιμετωπίστηκαν με τη χρήση του αλγορίθμου.

Κεφάλαιο 3

Υλοποίηση

3.1 Μοντέλο γραμμικού προγραμματισμού - περίπτωση του λύτη βελτιστοποίησης Gurobi

3.1.1 Γενικά

Η επίλυση γραμμικών προβλημάτων χαρακτηρίζεται απο αυξημένες απαιτήσεις σε μνήμη, υπολογιστική απόδοση και αριθμητική σταθερότητα οι οποίες καθιστούν ιδιαίτερα χρήσιμη τη χρήση εξειδικευμένων λυτών. Πρόκειται για ιδιαίτερα αποτελεσματικά εργαλεία που μπορούν να χειριστούν μεγάλα και πολύπλοκα προβλήματα με πολλαπλά κριτήρια [37]. Μεταξύ των πιο διαδεδωμένων λυτών είναι οι IBM, ILOG CPLEX, SCIP, CLP και Gurobi. Στην παρούσα εργασία, επιλέχθηκε ο λύτης Gurobi λόγω υψηλής αποδοτικότητας και ευρείας αποδοχής του στη βέλτιστη επίλυση προβλημάτων γραμμικού προγραμματισμού.

Οι λύτες γενικά διακρίνονται σε εμπορικά και ανοιχτού κώδικα λογισμικά, με τον Gurobi να ανήκει στην πρώτη κατηγορία [38]. Αποτελεί έναν απο τους πλέον σύγχρονους και ισχυρούς εμπορικούς λύτες βελτιστοποίησης. Όσον αφορά την εταιρεία Gurobi Optimization, ιδρύθηκε το 2008 με στόχο την ανάπτυξη ενός λύτη υψηλής απόδοσης για την επίλυση μαθηματικών προβλημάτων, όπως προβλήματα γραμμικού και μικτού ακέραιου προγραμματισμού, με έμφαση στην ταχύτητα και την αξιοπιστία. Το όνομα της εταιρείας προκύπτει από τα αρχικά των επιθέτων των τριών συνιδρυτών της: Zonghao Gu, Ed Rothberg και Bob Bixby. Ο Gurobi προσφέρει ευρεία υποστήριξη σε πολλές γλώσσες προγραμματισμού, συμπεριλαμβανομένων των C, C++, C#, Java, Microsoft .NET, Python, MATLAB και R. Στο πλάισιο

της παρούσας μελέτης αξιοποιήθηκε ο Gurobi Optimizer version 11.0.3 μέσω της γλώσσας Python, για την ανάπτυξη και επίλυση του μαθηματικού μοντέλου. Για τη διατύπωση του μοντέλου χρησιμοποιηθήκε το Pyomo, ένα ισχυρό εργαλείο ανοιχτού κώδικα για μοντελοποίηση προβλημάτων βελτιστοποίησης, το οποίο επιτρέπει την εύκολη ενσωμάτωση περιορισμών και μεταβλητών με μεγάλο βαθμό ευελιξίας [39].

3.1.2 Τρόπος υλοποίησης

Για την υλοποίηση του μοντέλου ακέραιου γραμμικού προγραμματισμού με τη χρήση του λύτη Gurobi χρησιμοποιήθηκαν τρείς μεταβλητές. Η πρώτη ακέραια μεταβλητή x_{ij} δηλώνει πόσοι φορτιστές τύπου j εγκαθίστανται στον κόμβο i και έχει εύρος τιμών απο 0 εως 5. Οι δύο επόμενες μεταβλητές είναι οι δυαδικές y_{ki} και z_i που ορίζουν αν ένας πελάτης k εξυπηρετείται απο τον κόμβο i και αντίστοιχα αν η εγκατάσταση στον κόμβο i είναι ανοιχτή. Η υλοποίηση περιλαμβάνει 11 περιορισμούς που αναλύθηκαν στη μοντελοποίηση στο κεφάλαιο 2.2 και εξασφαλίζουν την ορθότητα και τη λειτουργικότητα της λύσης. Αυτό επιτυγχάνεται με την κάλυψη κρίσιμων πτυχών, όπως η πλήρης εξυπηρέτηση των πελατών, η ανάθεση εγκαταστάσεων σε κόμβους που διαθέτουν επαρκή αριθμό θέσεων και την απαιτούμενη χωρητικότητα ισχύος, η ικανοποίηση οικονομικών περιορισμών και περιορισμών απόστασης οδήγησης, η τήρηση ελάχιστης απόστασης μεταξύ εγκαταστάσεων, καθώς και άλλων σχετικών περιορισμών. Η βελτιστοποίηση βασίστηκε στην ελαχιστοποίηση της συνολικής απόστασης που διανύουν οι πελάτες για να εξυπηρετηθούν από τον πλησιέστερο σταθμό φόρτισης, η οποία αντιστοιχεί στο πρόβλημα p-median. Το μοντέλο περιλαμβάνει διακριτούς τύπους φορτιστών, όρια ισχύος στις εγκαταστάσεις και περιορισμούς απόστασης καθιστώντας το πρόβλημα πιο σύνθετο σε σχέση με το τυπικό p median.

Ακολουθούν οι εντολές που προσδιορίζουν ότι το μοντέλο είναι για ελαχιστοποίηση και η εντολή της βελτιστοποίησης για να υπολογίσει την καλύτερη λύση. Η επίλυση με τον Gurobi έγινε μέσω της $gurobi\_persistent$ διεπαφής του Pyomo και ορίστηκε ανώτατο όριο εκτέλεσης η μία ώρα.

Παρακάτω υπάρχουν στιγμιότυπα εκτέλεσης απο τη λύση του λύτη Gurobi για το πρώτο πρόβλημα (pmed1) της κλάσης 25_10_5. Στην περίπτωση αυτή, θεωρούνται $N = 15$ πιθανοί κόμβοι για εγκατάσταση φορτιστών, 10 πελάτες που πρέπει να

εξυπηρετηθούν, ενώ ο στόχος είναι να επιλεγούν 5 θέσεις εγκατάστασης, δηλαδή να ανοίξουν 5 σταθμοί φόρτισης. Στο Σχήμα 3.1 δίνονται οι τεχνικές πληροφορίες και περιγράφεται το μαθηματικό μοντέλο. Αποτελείται από 523 περιορισμούς και 196 μεταβλητές, εκ των οποίων οι 195 είναι ακέραιες και μεταξύ αυτών, 165 είναι δυαδικές μεταβλητές. Ακόμη, υπάρχουν 1262 μη μηδενικοί συντελεστές στον πίνακα περιορισμών. Η διαδικασία προεπεξεργασίας απλοποίησε σημαντικά το πρόβλημα, μειώνοντας τελικά σε 177 περιορισμούς και 164 μεταβλητές και καταλήγει σε ευρετική λύση με αντικειμενική τιμή 20.

Σχήμα 3.1: Παράδειγμα προβλήματος pmed1 με λύτη Gurobi part1.

```
PS C:\Users\hp-i3> python "C:\Users\hp-i3\python2\model.py" ./25_10_5/  --problems 0.txt



***************Solving problem 0.txt***************


Set parameter Username
Academic license - for non-commercial use only - expires 2025-10-21
Set parameter TimeLimit to value 3600
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0 (19045.2))

CPU model: AMD A6-9225 RADEON R4, 5 COMPUTE CORES 2C+3G, instruction set [SSE2|AVX|AVX2]
Thread count: 2 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 523 rows, 196 columns and 1262 nonzeros
Model fingerprint: 0x0e016df5
Variable types: 1 continuous, 195 integer (165 binary)
Coefficient statistics:
  Matrix range     [1e+00, 6e+03]
  Objective range  [1e+00, 8e+00]
  Bounds range     [1e+00, 5e+00]
  RHS range        [1e+00, 6e+04]
Found heuristic solution: objective 38.0000000
Presolve removed 346 rows and 32 columns
Presolve time: 0.01s
Presolved: 177 rows, 164 columns, 496 nonzeros
Variable types: 0 continuous, 164 integer (164 binary)
Found heuristic solution: objective 20.0000000
```

Στο Σχήμα 3.2 φαίνεται το αποτέλεσμα απο το Root Relaxation το οποίο βρήκε τελική βέλτιστη λύση με τιμή 12 σε 0.08 δευτερόλεπτα, η οποία είναι η βέλτιστη λύση καθώς το κενό (gap) μηδενίστηκε.

Σχήμα 3.2: Παράδειγμα προβλήματος pmed1 με λύτη Gurobi part2.

```
Root relaxation: objective 1.200000e+01, 31 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

H    0     0                    12.0000000    0.00000   100%     -    0s
     0     0          -    0        12.00000   12.00000  0.00%     -    0s

Explored 1 nodes (39 simplex iterations) in 0.08 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 3: 12 20 38
```

Οι τιμές των μεταβλητών της λύσης παρουσιάζονται στο Σχήμα 3.3. Πρόκειται

για τις τρείς μεταβλητές του προβλήματος x_{ij}, y_{ki} και z_i. Ο πρώτος δείκτης στη μεταβλητή x_{ij} δηλώνει σε ποια θέση άνοιξε σταθμός φόρτισης και ο δεύτερος δείκτης αναφέρεται στον τύπο του φορτιστή (το 0 αντιστοιχεί σε φορτιστή τύπου 2 και το 1 σε φορτιστή τύπου 3). Στη μεταβλητή y_{ki}, ο πρώτος δείκτης αντιστοιχεί στον πελάτη και ο δεύτερος στη θέση εγκατάστασης. Η τιμή $y[k,i] = 1$ υποδηλώνει ότι ο πελάτης k εξυπηρετείται απο τον σταθμό που έχει ανοίξει στη θέση i. Τέλος, με τη μεταβλητή $z_i = 1$ δηλώνεται κάθε μία απο τις θέσεις i που άνοιξαν σταθμοί φόρτισης. Αναφέρεται ότι το πρόβλημα λύθηκε βέλτιστα και δίνεται η αντικειμενική τιμή 12.

Σχήμα 3.3: Παράδειγμα προβλήματος pmed1 με λύτη Gurobi part3.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 1.200000000000e+01, best bound 1.200000000000e+01, gap 0.0000%
Print values for all variables
x variables (φορτιστές σε εγκαταστάσεις):
x[0,0] = 1.0
x[1,0] = 1.0
x[6,0] = 1.0
x[11,0] = 1.0
x[14,0] = 1.0


y variables (ανάθεση πελατών σε εγκαταστάσεις):
y[0,1] = 1.0
y[1,1] = 1.0
y[2,0] = 1.0
y[3,0] = 1.0
y[4,1] = 1.0
y[5,0] = 1.0
y[6,6] = 1.0
y[7,11] = 1.0
y[8,14] = 1.0
y[9,11] = 1.0


z variables (εγκαταστάσεις που ανοίγουν):
z[0] = 1.0
z[1] = 1.0
z[6] = 1.0
z[11] = 1.0
z[14] = 1.0
Model solved to optimality
Objective value = 12.0
```

Οι τελευταίες πληροφορίες που μας δίνονται και φαίνονται στο Σχήμα 3.4 αφορούν τα αποτελέσματα και είναι η μέση τιμή αντικειμενικής όπου εδώ ισούται με 12, ο μέσος χρόνος επίλυσης 0.102 δευτερόλεπτα και αναφέρεται ότι λύθηκε μόνο το πρόβλημα το οποίο περιλαμβανόταν στο αρχείο 0.txt και κρίθηκε ως εφικτό.

Σχήμα 3.4: Παράδειγμα προβλήματος pmed1 με λύτη Gurobi part4.

```
============================== RESULTS ================================


Average objective value for all feasible problems: 12.0.
Average run time for all feasible problems: 0.102 s.


Infeasible problems:
Feasible problems: 0.txt
Aborted problems:

=======================================================================
```

3.1.3 Ψευδοκώδικας της μοντελοποίησης του γραμμικού προβλήματος με χρήση του λύτη Gurobi

Στον αλγόριθμο 1 δίνεται ο ψευδοκώδικας της μοντελοποίησης του προβλήματος με χρήση του λύτη Gurobi.

3.2 Μέθοδος Branch and Bound

3.2.1 Γενικά

Ο Branch and Bound είναι ένας αλγόριθμος βελτιστοποίησης που στοχεύει στη συστηματική αναζήτηση της βέλτιστης λύσης ενός προβλήματος, μειώνοντας σημαντικά τον υπολογιστικό φόρτο. Στο χώρο του μαθηματικού προγραμματισμού οι τέσσερεις βασικοί τομείς εφαρμογής της μεθόδου Branch and Bound είναι: ο ακέραιος προγραμματισμός, ο μη γραμμικός προγραμματισμός, το πρόβλημα του πλανόδιου πωλητή και το πρόβλημα τετραγωνικής ανάθεσης [40]. Ο συγκεκριμένος αλγόριθμος ακολουθεί τη στρατηγική ”διαίρει και βασίλευε”, επιχειρώντας να διασπάσει το αρχικό πρόβλημα σε μικρότερα και πιο διαχειρίσιμα υποπροβλήματα, τα οποία επιλύονται πρώτα ώστε να διευκολυνθεί η συνολική επίλυση. Επίσης, χρησιμοποιεί εκτιμήσεις ορίων με σκοπό να αποκλείσει τα υποπροβλήματα που δεν μπορούν να οδηγήσουν σε καλύτερη λύση απο την υπάρχουσα, αποφεύγοντας έτσι την εξαντλητική αναζήτηση. Η μέθοδος αυτή εφαρμόζεται με επιτυχία σε σύνθετα και πολύπλοκα προβλήματα επιλογής τοποθεσίας [41].

3.2.2 Τρόπος υλοποίησης

Για τη μέθοδο Branch and Bound, υλοποιήθηκαν τέσσερεις τεχνικές. Οι δύο πρώτες αφορούν τους ευρετικούς αλγόριθμους Min Distance και Distance from 0.5 και

Input: Input filename with instance data
Output: Optimization results, objective value
function `runmodel(`*filename*`)`;
$model = ConcreteModel()$;
$x[i,j]$: integer variables – number of chargers of type j at location i;
$y[k,i]$: binary variables – customer k is assigned to facility i;
$z[i]$: binary variables – whether a facility is installed at node i;
for $i = 0$ **to** $N-1$ **do**
  $\sum_j x[i,j] \leq place[i]$;
  $\sum_j c\_level[j] \cdot x[i,j] \leq capacity[i]$;
  $\sum_j x[i,j] \geq z[i]$;
  $\sum_j x[i,j] \leq place[i] \cdot z[i]$;
end
$\sum_i z[i] = P$;
for $k = 0$ **to** $CL-1$ **do**
  $\sum_i y[k,i] = 1$;
  for $i = 0$ **to** $N-1$ **do**
    $y[k,i] \leq z[i]$;
  end
end
for $i,j = 0$ **to** $N-1$ ***with*** $i \neq j$ **do**
  if $fpEucDistances[i][j] \leq binaryConstraint[p][l]$ **then**
    $z[i] + z[j] \leq 1$;
  end
end
for $k = 0$ **to** $CL-1$ **do**
  for $i = 0$ **to** $N-1$ **do**
    $clEucDistances[k][i] \cdot y[k,i] \leq D$;
  end
end
for $i = 0$ **to** $N-1$ **do**
  $a \cdot x[i,0] + b \cdot x[i,1] \geq \sum_k y[k,i]$;
end
$\sum_i (f \cdot z[i] + g \cdot x[i,0] + h \cdot x[i,1]) \leq budget$;
$minimize\ \sum_{k,i} clSpDistances[k][i] \cdot y[k,i]$;
$solver = SolverFactory('gurobi\_persistent')$;
$solver.set\_instance(model)$;
$solver.options['TimeLimit'] = TIME\_LIMIT$;
$results = solver.solve(tee = True)$;
return $results$, $model.obj()$

Αλγόριθμος 1: Μοντελοποίηση προβλήματος με λύτη Gurobi.

αναπτύσσονται με αναζήτηση πρώτα κατά βάθος (Depth First Search), ενώ οι άλλες δύο μέθοδοι αφορούν τους ίδιους ευρετικούς αλλά αναπτύσσονται με τη μέθοδο αναζήτησης πρώτα στο καλύτερο (Best First Search). Οι μέθοδοι αυτοί διαφέρουν

στον τρόπο που επιλέγονται κόμβοι-φύλλα για να εξερευνηθούν.

Για την υλοποίηση του αλγορίθμου Branch and Bound με αναζήτηση κατά βάθος, χρησιμοποιείται η δομή δεδομένων στοίβα, η οποία λειτουργεί με τη λογική "Last In First Out" (LIFO), δηλαδή εξάγει πρώτο το στοιχείο που εισήχθη τελευταίο.

Η διαδικασία ξεκινά με τη δημιουργία του ριζικού κόμβου και την επίλυση του αντίστοιχου χαλαρωμένου προβλήματος, χωρίς περιορισμούς ακεραιότητας (Root Relaxation). Αν σε αυτό το στάδιο προκύψει ακέραια λύση, το πρόβλημα θεωρείται λυμένο και επιστρέφονται οι τιμές των μεταβλητών και η τιμή της αντικειμενικής συνάρτησης.

Αν η λύση είναι μη ακέραια, αποθηκεύονται οι τιμές των μεταβλητών και συνεχίζεται η αναζήτηση. Ο έλεγχος ακεραιότητας πραγματοποιείται αποκλειστικά για τις δυαδικές μεταβλητές z_i, οι οποίες υποδηλώνουν την εγκατάσταση ή μη σταθμού φόρτισης σε συγκεκριμένη θέση.

Η επιλογή της μεταβλητής για διακλάδωση βασίζεται σε ευρετικές μεθόδους που θα αναλυθούν στη συνέχεια. Μόλις επιλεγεί μεταβλητή προς διακλάδωση, δημιουργούνται δύο θυγατρικοί κόμβοι: ο αριστερός κόμβος, όπου η μεταβλητή λαμβάνει την τιμή 0 και ο δεξιός κόμβος, όπου η τιμή είναι 1. Οι κόμβοι αυτοί προστίθενται στη στοίβα με σειρά: πρώτα ο αριστερός και έπειτα ο δεξιός.

Δεδομένου ότι το πρόβλημα είναι ελαχιστοποίησης, το κάτω όριο (lower bound) σε κάθε κόμβο ισούται με την τιμή της αντικειμενικής συνάρτησης που προκύπτει από την επίλυση. Η διαδικασία συνεχίζεται επαναληπτικά, έως ότου βρεθεί ακέραια βέλτιστη λύση ή εξαντληθούν όλοι οι κόμβοι στη στοίβα.

Σε κάθε βήμα, ανακτάται ο κόμβος από την κορυφή της στοίβας. Επιλέγεται να εξετάζεται πρώτος ο δεξιός κόμβος (αντιστοιχεί σε ανοιχτή εγκατάσταση), καθώς αυτό επιτρέπει την εστίαση της αναζήτησης σε κόμβους με πιθανώς καλύτερες λύσεις. Για να διασφαλιστεί αυτή η σειρά επεξεργασίας, ο δεξιός κόμβος προστίθεται τελευταίος στη στοίβα, ώστε να ανακτηθεί πρώτος κατά την επόμενη επανάληψη, ακολουθώντας τη λογική της στοίβας τύπου LIFO.

Για λόγους αποδοτικότητας, γίνεται χρήση βάσεων (basis) από προηγούμενους κόμβους για την ταχεία εκκίνηση (warm-start) του μοντέλου. Ενσωματώνονται οι νέοι περιορισμοί που απορρέουν από τη διακλάδωση της μη ακέραιας μεταβλητής και το τροποποιημένο πρόβλημα επιλύεται εκ νέου.

Αν το υποπρόβλημα είναι μη εφικτό, ο κόμβος αποκόπτεται και δεν δημιουργούνται περαιτέρω θυγατρικοί κόμβοι. Αν η λύση που προκύψει είναι και πάλι μη ακέραια, η διαδικασία διακλάδωσης επαναλαμβάνεται. Αντίθετα, αν προκύψει ακέραια λύση, ελέγχεται κατά πόσο βελτιώνει το άνω όριο (upper bound). Εφόσον ισχύει ότι το άνω όριο ισούται με το κάτω, έχει επιτευχθεί βέλτιστη λύση και η αναζήτηση σταματά.

Αν η νέα ακέραια λύση δεν βελτιώνει το τρέχον άνω όριο, τότε ο κόμβος δεν επεκτείνεται περαιτέρω. Στην περίπτωση που η λύση είναι ακέραια και βελτιώνει το άνω όριο, αυτή αποθηκεύεται και η αναζήτηση συνεχίζεται με νέα διακλάδωση.

Το άνω όριο ενημερώνεται με την καλύτερη ακέραια λύση που έχει βρεθεί έως εκείνη τη στιγμή. Όταν ολοκληρωθεί η διαδικασία και εντοπιστεί η βέλτιστη λύση, ανακτώνται οι ακέραιες τιμές των μεταβλητών z_i και εισάγονται ως σταθερές στο μοντέλο του Gurobi.

Στη συνέχεια, καλείται ο Gurobi για να επιλύσει το πρόβλημα εκ νέου, με δεδομένες τις τιμές των z_i, παρέχοντας βέλτιστες ακέραιες τιμές για τις x_{ij} και y_{ki}. Τέλος, υπολογίζεται και καταγράφεται ο συνολικός χρόνος εκτέλεσης τόσο του custom Branch and Bound αλγορίθμου όσο και του Gurobi.

Ο αλγόριθμος Branch and Bound με στρατηγική επιλογής του καλύτερου επόμενου κόμβου (Best First Search) ακολουθεί παρόμοια διαδικασία με την εκδοχή αναζήτησης κατά βάθος, παρουσιάζοντας ωστόσο ορισμένες βασικές διαφοροποιήσεις.

Αντί για στοίβα, χρησιμοποιείται μια δομή λίστας για την αποθήκευση των κόμβων προς εξερεύνηση. Η χρήση λίστας επιτρέπει την επιλογή, σε κάθε βήμα, του κόμβου που θεωρείται περισσότερο υποσχόμενος με βάση συγκεκριμένα κριτήρια.

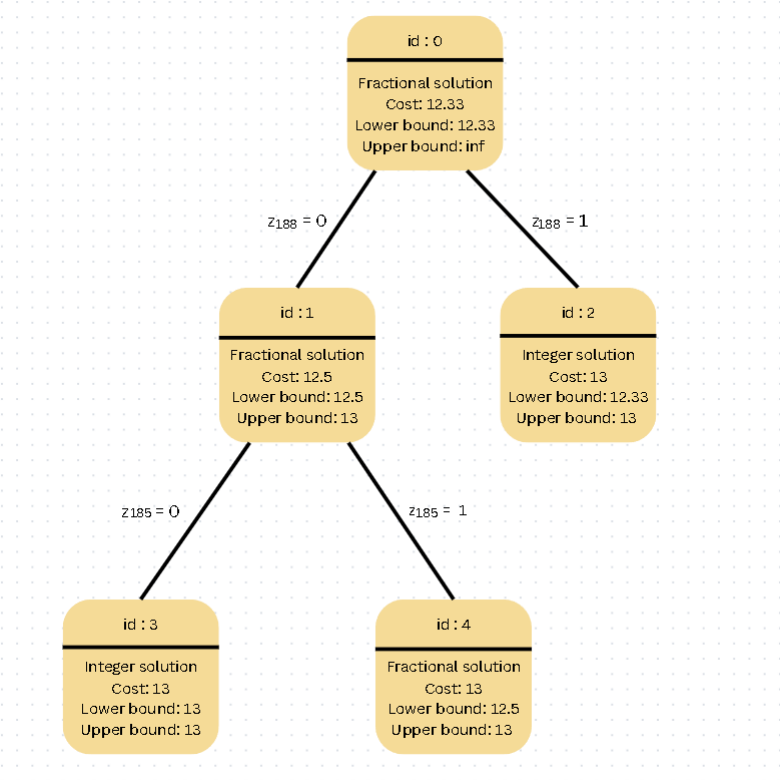
Συγκεκριμένα, επιλέγεται κάθε φορά για επέκταση ο κόμβος με το χαμηλότερο εκτιμώμενο κάτω όριο (lower bound) ανάμεσα σε όλους τους διαθέσιμους κόμβους της λίστας. Σε περίπτωση ισοτιμίας μεταξύ κόμβων ως προς το κάτω όριο, προτιμάται εκείνος που αποτελεί δεξιό παιδί στη διαδικασία διακλάδωσης, δηλαδή για τον οποίο ισχύει $z_i = 1$.

Για την υλοποίηση του αλγορίθμου Branch and Bound δημιουργήθηκε μια κλάση Node, η οποία λειτουργεί ως δομή δεδομένων για την αναπαράσταση κάθε κόμβου στο δέντρο αναζήτησης. Η δομή αυτή περιλαμβάνει τα χαρακτηριστικά του κά-

θε κόμβου, όπως το άνω και κάτω όριο (upper bound, lower bound), το βάθος του κόμβου στο δέντρο (depth), τις βάσεις μεταβλητών και περιορισμών (vbasis, cbasis), τη μεταβλητή που χρησιμοποιείται για διακλάδωση (branching var) και μια ετικέτα (label) που λειτουργεί ως βοηθητικό χαρακτηριστικό για να ξεχωρίζουμε τους κόμβους σε δεξί και αριστερό παιδί. Η δομή Node εξυπηρετεί τη δημιουργία κόμβων-παιδιών στο δέντρο αναζήτησης επιτρέποντας την αξιοποίηση της λύση του γονικού κόμβου ως αρχική τιμή (warm start), ώστε να επιταχυνθεί η επίλυση. Παράλληλα, για λόγους αποδοτικότητας ο Gurobi αποφεύγει τον επαναπροσδιορισμό του μοντέλου απο την αρχή και επαναχρησιμοποιεί το ίδιο, τροποποιώντας μόνο τους περιορισμούς που αλλάζουν απο κόμβο σε κόμβο.

Στο Σχήμα 3.5 ακολουθεί παράδειγμα επίλυσης του προβλήματος pmed10 με τον αλγόριθμο Branch and Bound που ακολουθεί τη στρατηγική πρώτα στο καλύτερο με τη χρήση της ευρετικής συνάρτησης Min Distance.

Σχήμα 3.5: Παράδειγμα Branch and Bound BestFS - Min Distance heuristic.



Η διαδικασία αναζήτησης με τη μέθοδο Branch and Bound ξεκινά από το ριζικό κόμβο με id = 0 και βάθος 0. Σε αυτό το σημείο, το άνω όριο (upper bound) αρχικοποιείται στο $+\infty$, καθώς δεν έχει εντοπιστεί ακόμη κάποια εφικτή ακέραια λύση, ενώ το κάτω όριο (lower bound) λαμβάνει την τιμή της αντικειμενικής συνάρτησης

της βέλτιστης λύσης του αντίστοιχου χαλαρωμένου προβλήματος. Στην περίπτωση αυτή, το χαλαρωμένο πρόβλημα επιστρέφει μη ακέραια λύση με αντικειμενική τιμή 12.33, συνεπώς το αρχικό κάτω όριο τίθεται ως $LB = 12.33$.

Εφόσον η λύση του ριζικού κόμβου δεν είναι ακέραια, πραγματοποιείται διακλάδωση. Με χρήση της ευρετικής Min Distance, επιλέγεται ως μεταβλητή διακλάδωσης η z_{188} και δημιουργούνται δύο κόμβοι-παιδιά σε βάθος 1: ο κόμβος με id = 1, όπου τίθεται $z_{188} = 0$ (δηλαδή η εγκατάσταση είναι κλειστή) και ο κόμβος με id = 2, όπου $z_{188} = 1$ (δηλαδή η εγκατάσταση είναι ανοιχτή).

Η στρατηγική Best First Search επιλέγει για διερεύνηση τον κόμβο της λίστας με το μικρότερο lower bound. Εφόσον και οι δύο νέοι κόμβοι έχουν $LB = 12.33$, επιλέγεται από σύμβαση ο δεξιός κόμβος, δηλαδή αυτός με id = 2. Το αντίστοιχο πρόβλημα επιλύεται και αυτή τη φορά επιστρέφει ακέραια λύση με αντικειμενική τιμή 13. Δεδομένου ότι είναι εφικτή ακέραια λύση και είναι καλύτερη από την αρχική τιμή του UB, το άνω όριο ενημερώνεται ως $UB = 13$.

Στη συνέχεια, ο μόνος κόμβος που απομένει στη λίστα είναι αυτός με id = 1. Η λύση του κόμβου αυτού είναι μη ακέραια με αντικειμενική τιμή 12.5, επομένως το LB για αυτόν τον κόμβο γίνεται 12.5. Εφόσον το 12.5 είναι μεγαλύτερο από το προηγούμενο 12.33, ανανεώνεται το global lower bound σε $LB = 12.5$.

Καθώς η λύση δεν είναι ακέραια, απαιτείται νέα διακλάδωση. Με χρήση της ίδιας ευρετικής επιλέγεται η μεταβλητή z_{185} για διαχωρισμό. Προκύπτουν δύο νέοι κόμβοι σε βάθος 2: ο κόμβος με id = 3, όπου $z_{185} = 0$ και ο κόμβος με id = 4, όπου $z_{185} = 1$. Και οι δύο κόμβοι έχουν αντικειμενική τιμή 12.5, δηλαδή $LB = 12.5$ για καθέναν. Επιλέγεται και πάλι ο δεξιός κόμβος (id = 4) για επέκταση.

Το πρόβλημα του κόμβου 4 επιλύεται και επιστρέφει μη ακέραια λύση με αντικειμενική τιμή 13. Επειδή όμως το $LB = 13$, δηλαδή ίσο με το υπάρχον $UB = 13$, δεν υπάρχει περιθώριο βελτίωσης. Συνεπώς, το υποδέντρο αυτό κλαδεύεται με την ένδειξη "Cut by bound", ακόμα και αν η λύση είναι μη ακέραια.

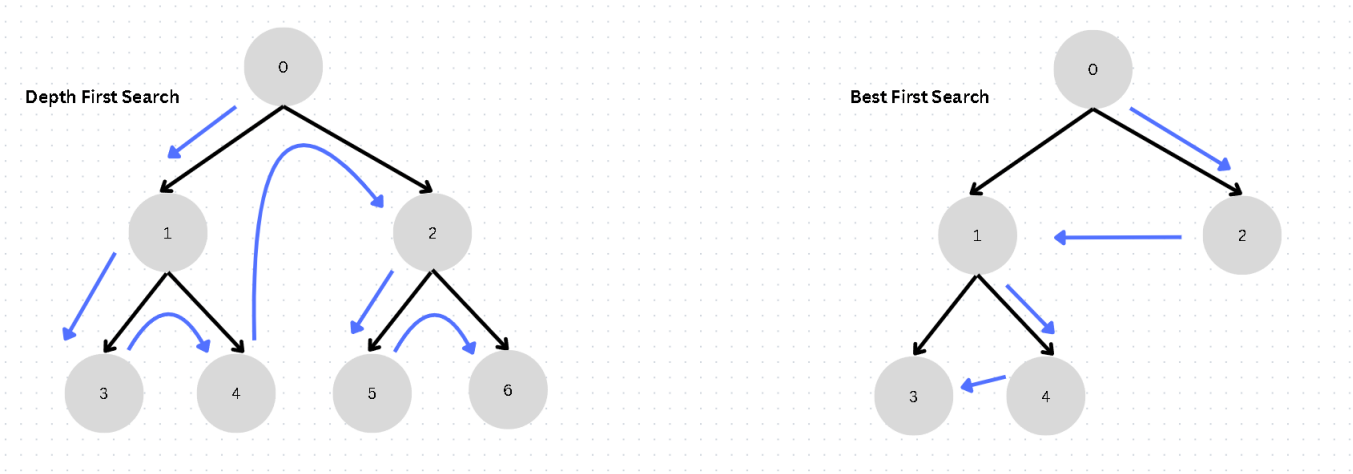
Η αναζήτηση συνεχίζεται με τον κόμβο με id = 3. Η λύση του είναι ακέραια, με αντικειμενική τιμή 13. Ωστόσο, επειδή είναι ισοδύναμη με την ήδη γνωστή καλύτερη λύση, δεν προκαλεί ενημέρωση του άνω ορίου και ο κόμβος απορρίπτεται ως μη βελτιωτικός.

Με αυτόν τον τρόπο, η διαδικασία Branch and Bound ολοκληρώνεται. Η καλύ-

τερη ακέραια λύση εντοπίζεται στον κόμβο με id = 2, με αντικειμενική τιμή 13 και αυτή θεωρείται η βέλτιστη λύση του προβλήματος.

Στο Σχήμα 3.6 παρουσιάζεται η σειρά εξερεύνησης των κόμβων με τις στρατηγικές αναζήτησης κατά βάθος (DFS) και πρώτα στο καλύτερο (BestFS). Το δέντρο που αντιστοιχεί στη μέθοδο BestFS βασίζεται στο παράδειγμα που αναλύθηκε προηγουμένως και στο οποίο εφαρμόστηκε η συγκεκριμένη τεχνική.

Σχήμα 3.6: Παράδειγμα Branch and Bound DFS - BestFS τεχνική.



3.2.3 Ψευδοκώδικας του Branch and Bound

Στον αλγόριθμο 2 δίνεται ο ψευδοκώδικας του αλγορίθμου Branch and Bound με BestFS στρατηγική και χρήση της Min Distance ευρετικής.

3.3 Ευρετικοί Αλγόριθμοι

3.3.1 Γενικά

Η χρήση ευρετικών μεθόδων σε αλγορίθμους Branch and Bound είναι κρίσιμη για τη βελτίωση της αποδοτικότητας και της ταχύτητας εύρεσης λύσεων [42]. Συγκεκριμένα, οι ευρετικές μπορούν να χρησιμοποιηθούν για την παροχή αρχικών λύσεων (warm-start), οι οποίες επιτρέπουν στον αλγόριθμο να ξεκινήσει απο ένα καλό σημείο στο χώρο αναζήτησης, μειώνοντας έτσι το πλήθος των κόμβων που θα χρειαστεί να εξεταστούν. Επιπλέον, η επιλογή της κατάλληλης μεταβλητής προς διακλάδωση με βάση ευρετικές αντί της τυχαίας επιλογής μπορεί να οδηγήσει σε

```
Output: Solutions list, best solution index, number of solutions
function Branch&Bound_Gurobi_BestFS_MinDistance(model, ub, lb,
 integer_var, best_bound_per_depth, nodes_per_depth, vbasis, cbasis, depth)
Initialize solutions ← [], solutions_found ← 0, best_sol_idx ← 0;
if isMax then
    best_sol_obj ← −∞;
end
else
    best_sol_obj ← +∞;
end
Create root node root;
node_list ← [root];
while node_list is not empty do
    current_node ← node with minimum lb (prefer "Right" in tie);
    Remove current_node from node_list;
    if has bases then
        Use VBasis, CBasis from parent in model;
    end
    Update LB, UB variables in model;
    model.optimize();
    if solution is infeasible then
        Skip node, update nodes_per_depth;
        continue;
    end
    Get x_candidate and x_obj;
    if all integer variables are integer then
        Store solution;
        Update best_sol_obj if necessary;
        if solution is optimal then
            return solutions, best_sol_idx, solutions_found;
        end
        continue;
    end
    if node can be pruned due to bounds then
        Update nodes_per_depth;
        continue;
    end
    Find selected_var_idx to branch (with minimum distance);
    Create left child with ub[selected_var_idx] ← ⌊x⌋;
    Create right child with lb[selected_var_idx] ← ⌈x⌉;
    Add children to node_list;
end
return solutions, best_sol_idx, solutions_found;
```

Αλγόριθμος 2: Αλγόριθμος Branch and Bound.

ταχύτερη σύγκλιση προς τη βέλτιστη λύση, καθώς επηρεάζει τη δομή του δέντρου αναζήτησης και την αποτελεσματικότητα των περικοπών που γίνονται σε μη υπο-

σχόμενες περιοχές [43]. Όσον αφορά το μυωπικό αλγόριθμο, πρόκειται για έναν άπληστο αλγόριθμο που διαλέγει κάθε φορά την επιλογή που φαίνεται καλύτερη για εκείνη τη στιγμή χωρίς να εξετάζει μελλοντικές συνέπειες [44]. Στη μελέτη μας, το χρησιμοποιούμε για να δώσει μια πρώτη εφικτή λύση στο πρόβλημα, ακόμη κι αν δεν είναι η βέλτιστη, ώστε να τη χρησιμοποιήσουμε ως warm-start στους μεθευρετικούς αλγόριθμους. Επιπλέον, υλοποιούμε δύο ευρετικές μεθόδους που βασίζονται στην τυχαιότητα, με στόχο την εύρεση εφικτών λύσεων για το πρόβλημά μας, οι οποίες ενεργοποιούνται εφόσον ο μυωπικός αλγόριθμος δεν καταφέρει να εντοπίσει αρχική εφικτή λύση. Οι στοχαστικοί ευρετικοί αλγόριθμοι έχουν το πλεονέκτημα ότι εξερευνούν ευρύτερα το χώρο λύσεων, αποφεύγοντας την παγίδευση σε τοπικά βέλτιστα. Παράλληλα, η υλοποίησή τους είναι σχετικά απλή και προσφέρει γρήγορα αποδεκτές λύσεις [45]. Συνεπώς, οι ευρετικές μέθοδοι είναι ιδιαίτερα σημαντικές σε προβλήματα μεγάλης κλίμακας, όπου ο υπολογιστικός φόρτος μπορεί να είναι υψηλός [46].

3.3.2 Ευρετικός αλγόριθμος Min Distance

Τρόπος υλοποίησης

Η στρατηγική αυτή καθορίζει τον τρόπο επιλογής κάθε φορά μιας από τις μη ακέραιες μεταβλητές z_i στον αλγόριθμο Branch and Bound, με σκοπό τη δημιουργία διακλαδώσεων στο δέντρο αναζήτησης. Η ευρετική μέθοδος που εφαρμόζεται υπολογίζει τη συνολική απόσταση κάθε πιθανής εγκατάστασης από όλους τους κόμβους ζήτησης. Για κάθε μεταβλητή z_i, που αντιστοιχεί σε πιθανή θέση σταθμού φόρτισης, εκτελείται μια σειρά ελέγχων, προκειμένου να επιλεγεί για διακλάδωση η μη ακέραια μεταβλητή με τη μικρότερη συνολική απόσταση από τους κόμβους ζήτησης. Μέσω αυτής της στρατηγικής, επιδιώκουμε τη διακλάδωση σε μεταβλητές που φαίνεται να είναι πιο κρίσιμες για την επίλυση του προβλήματος, συμβάλλοντας έτσι σε ταχύτερη εύρεση της βέλτιστης λύσης. Οι μεταβλητές που επιλέγονται με αυτό το κριτήριο πιθανόν να επηρεάζουν μεγαλύτερο αριθμό κόμβων ζήτησης και να δημιουργούν πιο αυστηρά όρια στο δέντρο αναζήτησης, επιτρέποντας την έγκαιρη αποκοπή μεγάλου μέρους του χώρου λύσεων.

Ψευδοκώδικας Min Distance

Στον Αλγόριθμο 3 δίνεται ο ψευδοκώδικας του ευρετικού αλγορίθμου Min Distance.

Output: *selected_var_idx*: index of selected branching variable
function `calculate_min_distance`(x_candidate, integer_var, cl_sp_distances, start_idx, end_idx);
$min\_total\_distance \leftarrow \infty$;
$selected\_var\_idx \leftarrow$ `None`;
$vars\_have\_integer\_vals \leftarrow$ `True`;
for $idx \leftarrow start\_idx$ **to** $end\_idx - 1$ **do**
  if $integer\_var[idx]$ *is True* ***and*** $x\_candidate[idx]$ *is not nearly integer* **then**
    $vars\_have\_integer\_vals \leftarrow$ `False`;
    $total\_distance \leftarrow 0$;
    for $cl \leftarrow 0$ **to** $|cl\_sp\_distances| - 1$ **do**
      $total\_distance \leftarrow total\_distance + cl\_sp\_distances[cl,\ idx - start\_idx]$;
      `// Sum distance from current variable to all clients`
    end
    if $total\_distance < min\_total\_distance$ **then**
      $min\_total\_distance \leftarrow total\_distance$;
      $selected\_var\_idx \leftarrow idx$;
      `// Update if better (closer) variable found`
end
return *selected_var_idx*

Αλγόριθμος 3: Ευρετικός αλγόριθμος Min Distance.

3.3.3 Ευρετικός αλγόριθμος Distance from 0.5

Τρόπος Υλοποίησης

Ο ευρετικός αλγόριθμος Distance from 0.5 αποτελεί μια ακόμη στρατηγική που εφαρμόσαμε για την επιλογή της μεταβλητής προς διακλάδωση κατά την αναζήτηση της βέλτιστης λύσης σε κάθε πρόβλημα. Για κάθε μεταβλητή z_i, η οποία έχει οριστεί ως ακέραια αλλά δεν έχει λάβει ακόμη ακέραια τιμή στη λύση, υπολογίζεται η απόστασή της από την τιμή 0.5. Στη συνέχεια, επιλέγουμε για διακλάδωση τη μεταβλητή με τη μικρότερη απόσταση από το 0.5, εστιάζοντας έτσι σε μεταβλητές που είναι κλασματικές και αβέβαιες ως προς το αν θα γίνουν 0 ή 1. Με αυτή την προσέγγιση αυξάνουμε τις πιθανότητες να καθοριστεί γρηγορότερα η τελική τιμή αυτών των κρίσιμων μεταβλητών, οι οποίες οδηγούν σε καθοριστικές διακλαδώσεις.

Ψευδοκώδικας Distance from 0.5

Στον Αλγόριθμο 4 δίνεται ο ψευδοκώδικας του ευρετικού αλγορίθμου Distance from 0.5.

Output: Selected index *selected_var_idx*
function `calculate_distance_from_0.5`(x_candidate, integer_var, start_idx, end_idx);
$min\_distance \leftarrow \infty$;
$selected\_var\_idx \leftarrow$ `None`;
$vars\_have\_integer\_vals \leftarrow$ `True`;
for $idx \leftarrow start\_idx$ **to** $end\_idx - 1$ **do**
  if $integer\_var[idx]$ *is True* ***and*** $x\_candidate[idx]$ *is not nearly integer* **then**
    $vars\_have\_integer\_vals \leftarrow$ `False`;
    $distance \leftarrow |x\_candidate[idx] - 0.5|$;
    `// Compute how far variable is from 0.5`
    if $distance < min\_distance$ **then**
      $min\_distance \leftarrow distance$;
      $selected\_var\_idx \leftarrow idx$;
      `// Update the most fractional variable`
end
return *selected_var_idx*

Αλγόριθμος 4: Ευρετικός αλγόριθμος Distance from 0.5.

3.3.4 Μυωπικός ευρετικός αλγόριθμος

Τρόπος υλοποίησης

Ο μυωπικός αλγόριθμος είναι ένας άπληστος αλγόριθμος που επιδιώκει να βρεί μια σχετικά καλή λύση γρήγορα ακόμη κι αν δεν είναι η βέλτιστη. Σκοπός μας είναι παρέχουμε αυτή τη λύση στους μεθευρετικούς αλγορίθμους ώστε να προσπαθήσουν να τη βελτιώσουν. Η διαδικασία του αλγόριθμου αποσκοπεί στον εντοπισμό P εγκαταστάσεων που ελαχιστοποιούν τη συνολική απόσταση μεταξύ σταθμών φόρτισης και χρηστών. Η επιλογή των εγκαταστάσεων γίνεται διαδοχικά, μία κάθε φορά, έως ότου καθοριστούν πλήρως και οι P θέσεις. Σε κάθε βήμα, επιλέγεται η εγκατάσταση που προσφέρει τη μεγαλύτερη μείωση στο συνολικό κόστος, χωρίς να λαμβάνονται υπόψη οι συνέπειες των επόμενων επιλογών, χαρακτηριστικό που καθιστά τον αλγόριθμο «μυωπικό». Για κάθε υποψήφια θέση εξετάζονται όλοι οι εφικτοί συνδυασμοί τύπων φορτιστών, που δεν υπερβαίνουν το διαθέσιμο αριθμό θέσεων της εγκατάστασης. Στη συνέχεια, οι πελάτες ανατίθενται στις πλησιέστερες

ανοιχτές εγκαταστάσεις, με την προϋπόθεση ότι αυτές βρίσκονται εντός ακτίνας 5 χιλιομέτρων. Πριν η θέση που εξετάζεται κατοχυρωθεί ως μία από τις P, πραγματοποιείται έλεγχος εφικτότητας βάσει των περιορισμών του προβλήματος. Αν δεν πληρούνται οι απαιτήσεις, εξετάζεται η επόμενη υποψήφια θέση. Μόλις ολοκληρωθεί η επιλογή των P εγκαταστάσεων, η τελική λύση υπόκειται σε συνολικό έλεγχο εφικτότητας. Αν ικανοποιούνται όλοι οι περιορισμοί, η λύση επιστρέφεται μαζί με το αντίστοιχο κόστος. Διαφορετικά, επιστρέφεται τιμή κόστους ίση με άπειρο.

Ψευδοκώδικας για τον Μυωπικό Αλγόριθμο

Στον Αλγόριθμο 5 δίνεται ο ψευδοκώδικας του ευρετικού, μυωπικού αλγορίθμου.

```
Result: final_solution
function myopic_heuristic(P, N, CL, binaryConstraint,
    fp_euc_distances, cl_euc_distances, cl_sp_distances,
    F, D, place, capacity, c_level);
  tempsol ← zeros(P);
  used_values ← [ ];
  for p ← 0 to P − 1 do
      best_cost ← ∞;
      for j ← 0 to N − 1 do
          if j ∈ used_values then continue;
          tempsol[p] ← j;
          if ¬ check_distance(tempsol, . . . ) then continue;
          for l2 ← 0 to place[j] do
              for l3 ← 0 to place[j] − l2 do
                  if l2 = 0 ∧ l3 = 0 then continue;
                  temp_x ← x;
                  temp_z ← z;
                  temp_y ← zeros(CL, N);
                  temp_x[j, 0] ← l2;
                  temp_x[j, 1] ← l3;
                  temp_z[j] ← 1;
              end
          end
      end
  end
```

Αλγόριθμος 5: Μυωπικός Αλγόριθμος (Μέρος 1).

```
for k ← 0 to CL − 1 do
    best_facility ← −1, best_dist ← ∞;
    for i ← 0 to N − 1 do
        if temp_z[i] = 1 ∧ cl_sp_dist[k, i] ≤ D ∧ cl_sp_dist[k, i] < best_dist
         then
            best_dist ← cl_sp_dist[k, i];
            best_facility ← i;
        end
    end
    if best_facility ≠ −1 then temp_y[k, best_facility] ← 1;
end
temp_solution ← concat(temp_x, temp_y, temp_z);
if ¬ partial_check_feasibility(temp_solution, . . . ) then continue;
cost ← getAssignmentCost(temp_solution, . . . );
if cost < best_cost then
    best_cost ← cost;
    best_x ← temp_x, best_y ← temp_y, best_z ← temp_z, best_j ← j;
end
if best_j ≠ None then
    tempsol[p] ← best_j;
    x ← best_x, y ← best_y, z ← best_z;
    used_values.append(best_j);
end
final_solution ← concat(x, y, z);
if check_feasibility(final_solution, . . . ) then
    final_cost ← getAssignmentCost(final_solution, . . . );
    return (1, final_solution, final_cost);
end
else
    return (0, final_solution, ∞);
end
```

Αλγόριθμος 5: Μυωπικός Αλγόριθμος (Μέρος 2).

3.3.5 Ευρετικός αλγόριθμος Randomized Constructive

Τρόπος υλοποίησης

Ο ευρετικός αυτός αλγόριθμος χρησιμοποιείται όταν ο μυωπικός αλγόριθμος αποτύχει να βρει εφικτή λύση, η οποία θα λειτουργούσε ως αρχική λύση για τους μεθευρετικούς αλγορίθμους. Σε αυτή την περίπτωση, ο Randomized Constructive προσπαθεί να δημιουργήσει μια πλήρη και εφικτή λύση, επιλέγοντας τυχαία P εγκαταστάσεις και εξερευνώντας έναν πολύ μεγάλο χώρο λύσεων μέσω πολλαπλών προσπαθειών, στην περίπτωση μας 1000 επαναλήψεις. Για κάθε επιλεγμένη εγκα-

τάσταση υπολογίζεται ο μέγιστος αριθμός φορτιστών που μπορεί να τοποθετηθεί, λαμβάνοντας υπόψη τόσο τον αριθμό διαθέσιμων θέσεων όσο και τη διαθέσιμη ισχύ της εγκατάστασης. Στη συνέχεια, γίνεται τυχαία κατανομή των φορτιστών μεταξύ των διαφορετικών τύπων. Ακολούθως, οι πελάτες ανατίθενται στην κοντινότερη ενεργή εγκατάσταση που βρίσκεται εντός απόστασης 5 χιλιομέτρων. Ο έλεγχος εφικτότητας πραγματοποιείται μόνο αφού ολοκληρωθεί η υποψήφια λύση, επομένως δεν απορρίπτονται πρόωρα ενδιάμεσες λύσεις. Η στοχαστική φύση της μεθόδου αξιοποιεί την ποικιλομορφία για να αυξήσει τις πιθανότητες εύρεσης έγκυρης λύσης. Σε περίπτωση που δεν προκύψει καμία εφικτή λύση μετά από όλες τις επαναλήψεις, επιστρέφεται μια μηδενική λύση με άπειρο κόστος, προκειμένου να αποτραπούν σφάλματα στα επόμενα βήματα.

Ψευδοκώδικας Randomized Constructive

Στον Αλγόριθμο 6 δίνεται ο ψευδοκώδικας του ευρετικού αλγορίθμου Randomized Constructive.

3.3.6 Ευρετικός αλγόριθμος Initialize Population

Τρόπος υλοποίησης

Για την αρχικοποίηση του πληθυσμού λύσεων στο γενετικό αλγόριθμο, υλοποιήθηκε η συνάρτηση Initialize Population, η οποία στοχεύει στη δημιουργία εφικτών ή προσεγγιστικά αποδεκτών λύσεων μέσω επαναλαμβανόμενης στοχαστικής διαδικασίας. Σε κάθε προσπάθεια, επιλέγεται τυχαία ένα υποσύνολο P εγκαταστάσεων από το σύνολο των N υποψήφιων σημείων. Οι επιλεγμένες εγκαταστάσεις ενεργοποιούνται και επιχειρείται η ανάθεση των CL πελατών σε αυτές με βάση τον περιορισμό της μέγιστης επιτρεπόμενης απόστασης των 5 χιλιομέτρων. Η διαδικασία αυτή υλοποιείται με επιλογή της κοντινότερης αποδεκτής εγκατάστασης για κάθε πελάτη. Ταυτόχρονα, καταγράφεται το πλήθος των πελατών που έχουν ανατεθεί σε κάθε ενεργή εγκατάσταση, καθώς αυτή η πληροφορία είναι κρίσιμη για τον υπολογισμό των απαιτήσεων εξοπλισμού και ειδικότερα για τον καθορισμό της κατάλληλης σύνθεσης φορτιστών που απαιτείται για την εξυπηρέτηση της ζήτησης.

Στη συνέχεια, για κάθε εγκατάσταση που επιλέχθηκε, εξετάζονται όλοι οι δυνατοί συνδυασμοί φορτιστών τύπου Level 2 και Level 3, με στόχο την κάλυψη του πλή-

θους των πελατών που της έχουν ανατεθεί, υπό τους περιορισμούς χωρητικότητας και ισχύος. Αν δεν εντοπιστεί κάποιος αποδεκτός συνδυασμός που να εξυπηρετεί τη ζήτηση, η υποψήφια λύση απορρίπτεται. Εφόσον βρεθεί συνδυασμός που ικανοποιεί τους περιορισμούς, επιλέγεται αυτός με το χαμηλότερο κόστος εγκατάστασης. Στο τέλος κάθε προσπάθειας, η συνολική λύση ελέγχεται για καθολική εφικτότητα και εφόσον αυτή επιβεβαιωθεί, επιστρέφεται. Σε διαφορετική περίπτωση, υπολογίζεται ποινή (penalty), η οποία βασίζεται στην έκταση των παραβιάσεων των περιορισμών και προστίθεται στο βασικό κόστος ανάθεσης. Σε περίπτωση που καμία εφικτή λύση δεν εντοπιστεί εντός των επιτρεπόμενων προσπαθειών, επιστρέφεται η καλύτερη μη εφικτή λύση βάσει του ποινικοποιημένου κόστους. Η χρήση ποινών επιτρέπει την αποδοτική αξιολόγηση και σύγκριση μη πλήρως εφικτών λύσεων, ενισχύοντας τη διερεύνηση του χώρου λύσεων, ακόμη και όταν δεν είναι δυνατή η άμεση εύρεση έγκυρης λύσης.

Αυτή η στρατηγική είναι ιδιαίτερα σημαντική στο πλαίσιο ενός Γενετικού Αλγορίθμου, καθώς επιτρέπει την ύπαρξη και αξιοποίηση μη απολύτως εφικτών λύσεων στον αρχικό πληθυσμό, δίνοντας τη δυνατότητα στον αλγόριθμο να εξερευνήσει ευρύτερα το χώρο αναζήτησης. Οι λύσεις αυτές, αν και αρχικά παραβιάζουν κάποιους περιορισμούς, μπορούν να εξελιχθούν μέσω των γενετικών τελεστών (διασταύρωση, μετάλλαξη) σε πλήρως εφικτές, συμβάλλοντας στη βελτίωση της ποιότητας και της ποικιλομορφίας του πληθυσμού και ενισχύοντας την εξερεύνηση του χώρου λύσεων.

Ψευδοκώδικας

Στον Αλγόριθμο 7 δίνεται ο ψευδοκώδικας του ευρετικού αλγορίθμου Initialize Population.

3.4 Μεθευρετικοί Αλγόριθμοι

3.4.1 Γενικά

Ο συνδυασμός μεθευρετικών μεθόδων με ακριβείς αλγορίθμους, όπως ο Branch and Bound, έχει αποδειχθεί ιδιαίτερα αποτελεσματικός, ιδίως όταν οι μεθευρετικές μέθοδοι χρησιμοποιούνται για την παραγωγή αρχικών εφικτών λύσεων (warm starts), οι οποίες συμβάλλουν στην καθοδήγηση ή τον περιορισμό του χώρου ανα-

ζήτησης του ακριβούς αλγορίθμου. Οι μεθευρετικοί αλγόριθμοι, όπως οι γενετικοί αλγόριθμοι, ο Simulated Annealing και ο Variable Neighborhood Search, μπορούν να προσφέρουν γρήγορα εφικτές λύσεις υψηλής ποιότητας με σχετικά χαμηλό υπολογιστικό κόστος [47]. Αυτές οι λύσεις, όταν ενσωματωθούν ως αρχικά άνω όρια (upper bounds) σε αλγορίθμους όπως ο Branch and Bound, οδηγούν σε σημαντική μείωση του δέντρου αναζήτησης και κατ' επέκταση του συνολικού χρόνου επίλυσης καθώς παρέχουν αυστηρότερα όρια [48]. Επιπλέον, σε εφαρμογές πραγματικού κόσμου, έχει φανεί πως τέτοιου είδους υβριδικές προσεγγίσεις αυξάνουν την αξιοπιστία και την επεκτασιμότητα των αλγορίθμων βελτιστοποίησης, καθιστώντας τες κατάλληλες και για μεγάλης κλίμακας προβλήματα [49]. Επιπλέον, οι γενετικοί αλγόριθμοι αποτελούν μια ισχυρή μεθοδολογία για την επίλυση του προβλήματος τοποθέτησης εγκαταστάσεων (Facility Location Problem), κυρίως λόγω της ικανότητάς τους να αναζητούν καλές λύσεις σε μεγάλα και σύνθετα διακριτά πεδία, όπου οι παραδοσιακές μέθοδοι δυσκολεύονται [50]. Επιτρέπουν την ευέλικτη ενσωμάτωση περιορισμών και έχουν εφαρμοστεί επιτυχώς σε διάφορες παραλλαγές του προβλήματος, όπως το capacitated p-median problem, προσφέροντας ποιοτικές λύσεις σε σύντομο χρόνο [51]. Αξιοσημείωτο είναι επίσης πως χάρη στη δυνατότητά τους να αποφύγουν τοπικά βέλτιστα και να εξερευνούν μεγάλο μέρος του χώρου λύσεων, είναι κατάλληλοι για προβλήματα facility location με πολλαπλούς στόχους, όπως ελαχιστοποίηση της απόστασης και του κόστους εγκατάστασης [52]. Τέλος, η χρήση υβριδικών γενετικών αλγορίθμων ενισχύει ακόμη περισσότερο την αποτελεσματικότητα τους, συνδυάζοντας την παγκόσμια αναζήτηση των GAs με τοπικές βελτιστοποιήσεις [53], καθιστώντας τους πολύτιμο εργαλείο για τη λήψη αποφάσεων σε προβλήματα πραγματικής ζωής.

3.4.2 Simulated Annealing

Τρόπος υλοποίησης

Ο αλγόριθμος Simulated Annealing που παρουσιάζεται είναι ένας μεθευρετικός αλγόριθμος βελτιστοποίησης, ο οποίος χρησιμοποιείται για την εύρεση καλών λύσεων σε προβλήματα υψηλής πολυπλοκότητας. Ξεκινά με μια αρχική λύση που, στην περίπτωσή μας, δίνεται από το μυωπικό αλγόριθμο ή από το Randomized Constructive και θεωρείται ως η καλύτερη μέχρι εκείνη τη στιγμή. Σε κάθε επα-

νάληψη, δημιουργείται μια νέα υποψήφια λύση μεταβάλλοντας ελαφρώς την τρέχουσα. Η δημιουργία αυτής της γειτονικής λύσης υλοποιείται μέσω της συνάρτησης get neighbor, η οποία επιλέγει τυχαία μια ενεργή εγκατάσταση και μια ανενεργή και μεταφέρει την πρώτη στη θέση της δεύτερης. Οι φορτιστές που υπήρχαν στην προηγούμενη θέση προσαρμόζονται ώστε να τηρούν τη χωρητικότητα της νέας εγκατάστασης, ενώ οι πελάτες επανατοποθετούνται ώστε να εξυπηρετούνται από την πλησιέστερη ενεργή εγκατάσταση εντός ακτίνας 5 χιλιομέτρων. Η νέα υποψήφια λύση αξιολογείται ως προς το συνολικό κόστος εξυπηρέτηση και ελέγχεται ότι πληροί όλους τους περιορισμούς εφικτότητας του προβλήματος. Η θερμοκρασία μειώνεται σταδιακά σε κάθε επανάληψη και αν η νέα λύση έχει χαμηλότερο κόστος από την καλύτερη μέχρι στιγμής, γίνεται αποδεκτή και ενημερώνει την τιμή της καλύτερης λύσης. Διαφορετικά, αν και είναι χειρότερη, μπορεί να γίνει αποδεκτή με πιθανότητα που εξαρτάται από τη θερμοκρασία. Όσο η θερμοκρασία μειώνεται, μειώνεται και η πιθανότητα αποδοχής χειρότερων λύσεων, επιτρέποντας έτσι σταδιακή σύγκλιση προς βελτιστοποιημένες λύσεις.

Ψευδοκώδικας Simulated Annealing

Στον Αλγόριθμο 8 δίνεται ο ψευδοκώδικας του μεθευρετικού αλγορίθμου Simulated Annealing.

3.4.3 Variable Neighborhood Search

Τρόπος υλοποίησης

Ο αλγόριθμος Variable Neighborhood Search είναι ένας μεθευρετικός αλγόριθμος βελτιστοποίησης που αξιοποιεί την ιδέα της δυναμικής αλλαγής ''γειτονιάς'' κατά την αναζήτηση λύσεων. Στόχος του είναι η βελτίωση μιας αρχικής λύσης ελαχιστοποιώντας το συνολικό κόστος εξυπηρέτησης με βάση τους περιορισμούς του προβλήματος που αφορούν αποστάσεις, χωρητικότητα εγκαταστάσεων κα. Η υλοποίηση του αλγορίθμου περιλαμβάνει τέσσερα βασικά στάδια: πρώτα γίνεται η αρχικοποίηση, όπου δημιουργείται μια αρχική λύση, ακολουθεί το ''shaking'', όπου επιλέγεται τυχαία μια νέα λύση εντός συγκεκριμένης γειτονιάς, ακολουθεί τοπική αναζήτηση, όπου η λύση βελτιώνεται μέσω έξυπνων τροποποιήσεων εγκαταστάσεων και ανακατανομής πελατών και τέλος γίνεται έλεγχος για αποδοχή ή απόρριψη της

νέας λύσης, ανάλογα με το αν επιτυγχάνει καλύτερο κόστος. Αν υπάρξει βελτίωση, η διαδικασία συνεχίζεται από την ίδια γειτονιά για να εξετάσουμε περαιτέρω βελτιώσεις διαφορετικά, εξερευνώνται σταδιακά οι επόμενες, πιο μακρινές γειτονιές. Με αυτό τον τρόπο, ο VNS συνδυάζει την εξερεύνηση και την εκμετάλλευση του χώρου λύσεων, αποφεύγοντας τοπικά ελάχιστα και αυξάνοντας την πιθανότητα εύρεσης μιας πολύ καλής ή και βέλτιστης λύσης.

Στην παρούσα υλοποίηση, οι επιμέρους διαδικασίες (όπως το shaking, η εύρεση γειτονικών λύσεων, η τοπική αναζήτηση και οι έλεγχοι εφικτότητας και απόστασης) έχουν αναπτυχθεί ως ξεχωριστές συναρτήσεις, προκειμένου να διασφαλιστεί η καλύτερη οργάνωση του κώδικα. Η συνάρτηση shaking εφαρμόζει τυχαίες αλλαγές στις θέσεις των εγκαταστάσεων μιας δεδομένης λύσης εντός των επιλεγμένων γειτονιών και κατά συνέπεια προσαρμόζει το πλήθος των φορτιστών και επαναπροσδιορίζει τις αναθέσεις των πελατών, ελέγχοντας ότι τηρείται η εφικτότητα των περιορισμών. Επιπλέον, η συνάρτηση create neighborhoods δημιουργεί τυχαία σύνολα εγκαταστάσεων που θα εξεταστούν σε κάθε επανάληψη του VNS με σκοπό να ελεγχθούν διαφορετικές τοπικές περιοχές λύσεων. Η συνάρτηση find best value αξιολογεί εναλλακτικές καταστάσεις (ενεργοποίηση ή απενεργοποίηση) για μια συγκεκριμένη εγκατάσταση, επιχειρώντας τροποποιήσεις που μπορεί να μειώσουν το συνολικό κόστος εξυπηρέτησης. Κατά τη διαδικασία, ελέγχεται αν η νέα κατάσταση καταλήγει σε εφικτή λύση, υπό την προϋπόθεση ότι ικανοποιούνται όλοι οι περιορισμοί του προβλήματος. Τελικά, επιλέγεται η βέλτιστη τιμή (ανοιχτή ή κλειστή) για τη συγκεκριμένη θέση, που βελτιστοποιεί το κόστος χωρίς να παραβιάζει την εφικτότητα. Η συνάρτηση local search υλοποιεί τοπική αναζήτηση. Για κάθε στοιχείο της γειτονιάς, επιχειρεί να βρει μια καλύτερη εκδοχή της λύσης τροποποιώντας τη συγκεκριμένη θέση (μέσω της find best value). Σε περίπτωση που η νέα λύση είναι εφικτή και παρουσιάζει μικρότερο κόστος εξυπηρέτησης, αντικαθιστά την τρέχουσα με τη νέα λύση. Η διαδικασία επαναλαμβάνεται για όλα τα στοιχεία της γειτονιάς και επιστρέφει τη βελτιωμένη λύση μαζί με το κόστος της. Τέλος, η συνάρτηση check distance εξετάζει αν ικανοποιούνται οι περιορισμοί απόστασης μεταξύ των εγκαταστάσεων και αντίστοιχα η check feasibility ελέγχει αν τηρούνται όλοι οι υπόλοιποι περιορισμοί του προβλήματος.

Ψευδοκώδικας Variable Neighborhood Search

Στον Αλγόριθμο 9 δίνεται ο ψευδοκώδικας του μεθευρετικού αλγορίθμου Variable Neighborhood Search.

3.4.4 Γενετικός αλγόριθμος GAFacilityOpt

Τρόπος υλοποίησης

Η υλοποίηση του γενετικού αλγορίθμου GAFacilityOpt βασίζεται σε μία προσαρμοσμένη έκδοση του πακέτου PyGAD, με σκοπό την ελαχιστοποίηση της συνολικής απόστασης ανάθεσης πελατών σε εγκαταστάσεις υπό την ύπαρξη πολλαπλών περιορισμών που αφορούν προϋπολογισμό, χωρητικότητα, ισχύ, απόσταση και κάλυψη. Η αναπαράσταση κάθε λύσης γίνεται με ένα χρωμόσωμα που περιλαμβάνει τις τρεις ομάδες μεταβλητών του προβλήματος: μεταβλητές x_{ij} που υποδεικνύουν τον αριθμό φορτιστών τύπου j σε εγκατάσταση i, τις μεταβλητές y_{ki} που δηλώνουν την ανάθεση του πελάτη k στην εγκατάσταση i και τις δυαδικές μεταβλητές z_i που δηλώνουν αν η εγκατάσταση i είναι ενεργή ή όχι.

Η συνάρτηση καταλληλότητας (fitness function) αποτελεί το βασικό μηχανισμό αξιολόγησης κάθε υποψήφιας λύσης. Συγκεκριμένα, υπολογίζει το αρνητικό της συνολικής απόστασης μετακίνησης των πελατών προς τις εγκαταστάσεις, μειώνοντας έτσι το στόχο σε ένα πρόβλημα μέγιστης καταλληλότητας. Επιπλέον, εφαρμόζεται ένα σύστημα ποινών που επιβάλει τιμωρητικό κόστος όταν παραβιάζονται οι περιορισμοί του μοντέλου. Αν το άθροισμα των ποινών είναι θετικό, δηλαδή αν υπάρχουν παραβιάσεις, προστίθενται στο συνολικό κόστος και επηρεάζουν αρνητικά τη συνολική τιμή της καταλληλότητας. Αν μια λύση έχει μεγάλο αριθμό ή σοβαρές παραβιάσεις, τότε το fitness της υποβαθμίζεται σημαντικά, καθιστώντας τη μη ανταγωνιστική στον πληθυσμό. Ο αρχικός πληθυσμός δημιουργείται με τις μεθόδους της Μυωπικής ευρετικής, η οποία βασίζεται σε τοπικές αποφάσεις ώστε να ικανοποιεί βασικούς περιορισμούς κόστους και κάλυψης και με τη Initialize Population, η οποία κατασκευάζει λύσεις μέσω τυχαίας αλλά ελεγχόμενης επιλογής ενεργών εγκαταστάσεων και φορτιστών. Η επιλογή (selection) πραγματοποιείται μέσω rank selection, κατά την οποία οι λύσεις ταξινομούνται με βάση την τιμή της συνάρτησης καταλληλότητας και οι καλύτερες έχουν μεγαλύτερη πιθανότητα να επιλεγούν

ως γονείς. Η διασταύρωση (crossover) υλοποιείται με single-point crossover, όπου επιλέγεται ένα τυχαίο σημείο στο χρωμόσωμα και ανταλλάσσονται τα αντίστοιχα τμήματα μεταξύ δύο γονεϊκών λύσεων, οδηγώντας στη δημιουργία απογόνων που ενσωματώνουν στοιχεία και από τους δύο γονείς. Η μετάλλαξη (mutation) εφαρμόζεται στο 10% των γονιδίων κάθε χρωμοσώματος, εισάγοντας μικρές, ελεγχόμενες και τυχαίες μεταβολές, όπως αλλαγές στον αριθμό φορτιστών ή μεταθέσεις ανάθεσης πελατών, με σκοπό τη διεύρυνση της αναζήτησης και την αποτροπή παγίδευσης σε τοπικά βέλτιστα.

Η διαδικασία επαναλαμβάνεται για έως και 1000 γενιές ή μέχρι την ικανοποίηση του προκαθορισμένου κριτηρίου σύγκλισης $saturate\_200$, σύμφωνα με το οποίο η βελτίωση της καλύτερης λύσης πρέπει να παραμένει αμετάβλητη για 200 διαδοχικές γενιές. Στο τελικό στάδιο, η προτεινόμενη λύση συγκρίνεται με τη βέλτιστη λύση που προκύπτει από τον ακριβή λύτη Gurobi. Η ποιότητα της προσέγγισης αξιολογείται μέσω του σχετικού GAP, το οποίο μετρά την απόκλιση της προσεγγιστικής λύσης από το βέλτιστο αποτέλεσμα.

Ψευδοκώδικας

Στον Αλγόριθμο 10 δίνεται ο ψευδοκώδικας του γενετικού αλγορίθμου GAFacilityOpt.

Output: Feasible solution or solution with cost ∞
function `generate_random_feasible_solution`(P, N, CL, F, D, place, capacity, c_level, binaryConstraint, clEucDistances, fpEucDistances) ;
$attempts \leftarrow 0$;
$max\_attempts \leftarrow 1000$;
$last\_solution \leftarrow$ None;
while $attempts < max\_attempts$ **do**
  $z \leftarrow$ zero vector of size N;
  $selected \leftarrow$ random selection of P facilities from $\{0, \ldots, N-1\}$;
  foreach $i \in selected$ **do**
    $z[i] \leftarrow 1$;
  end
  $x \leftarrow$ zero matrix of size $N \times |F|$;
  foreach $i \in selected$ **do**
    $max\_total \leftarrow place[i]$;
    $total \leftarrow$ random integer in $[1, max\_total]$;
    $first \leftarrow$ random integer in $[0, total]$;
    $x[i][0] \leftarrow first$;
    $x[i][1] \leftarrow total - first$;
  end
  $y \leftarrow$ zero matrix of size $CL \times N$;
  $feasible \leftarrow$ True;
  foreach $client \in \{0, \ldots, CL-1\}$ **do**
    $min\_dist \leftarrow \infty$, $assigned \leftarrow$ None;
    foreach $facility \in selected$ **do**
      if $clEucDistances[client][facility] \leq D \wedge clEucDistances[client][facility] < min\_dist$ **then**
        $min\_dist \leftarrow clEucDistances[client][facility]$;
        $assigned \leftarrow facility$;
      end
    end
    if $assigned \neq None$ **then**
      $y[client][assigned] \leftarrow 1$;
    else
      $feasible \leftarrow$ False; **break**;
    end
  end
  if $\neg feasible$ **then**
    $attempts \leftarrow attempts + 1$; **continue**;
  end
  $solution \leftarrow flatten(x) + flatten(y) + z$;
  $last\_solution \leftarrow solution$;
  $open\_facilities \leftarrow \{i \mid z[i] = 1\}$;
  if $check\_feasibility(solution) \wedge check\_distance(open\_facilities)$ **then**
    return $solution$, $getAssignmentCost(solution)$;
  end
  $attempts \leftarrow attempts + 1$;
end
return $last\_solution$ *or* $[0, 0, \ldots, 0], \infty$;

Αλγόριθμος 6: Αλγόριθμος Randomized Construction.

Output: Feasible solution or best attempt with cost ∞
function
`initialize_population_with_random_solution`$(P, N, CL, F, D, place,$
$capacity, c\_level, binaryConstraint, clEucDistances,$
$fpEucDistances, budget\_limit)$
$attempts \leftarrow 0$, $max\_attempts \leftarrow 100$;
$best\_attempt \leftarrow$ None, $best\_cost \leftarrow \infty$;
while $attempts < max\_attempts$ **do**
  $z \leftarrow$ zero vector of size N;
  $selected \leftarrow$ random set of P facilities from $\{0, \ldots, N-1\}$;
  foreach $i \in selected$ **do**
    $z[i] \leftarrow 1$;
  end
  $y \leftarrow$ zero matrix $CL \times N$;
  $assigned\_clients\_per\_node \leftarrow$ zero vector of size N;
  $feasible \leftarrow$ True;
  foreach $client \in \{0, \ldots, CL-1\}$ **do**
    $min\_dist \leftarrow \infty$, $assigned \leftarrow$ None;
    foreach $facility \in selected$ **do**
      if $clEucDistances[client][facility] \leq$
       $D \wedge clEucDistances[client][facility] < min\_dist$ **then**
        $min\_dist \leftarrow clEucDistances[client][facility]$,
         $assigned \leftarrow facility$;
      end
    end
    if $assigned \neq None$ **then**
      $y[client][assigned] \leftarrow 1$, $assigned\_clients\_per\_node[assigned] \mathrel{+}= 1$;
    else
      $feasible \leftarrow$ False; **break**;
    end
  end
  if $\neg feasible$ **then**
    $attempts \leftarrow attempts + 1$; **continue**;
  end
  $x \leftarrow$ zero matrix $N \times |F|$;
end

Αλγόριθμος 7: Αλγόριθμος Initialize Population (Μέρος 1).

```
foreach i ∈ selected do
    clients ← assigned_clients_per_node[i];
    if clients = 0 then
        continue
    end
    best_l2, best_l3 ← None, min_cost ← ∞, found ← False;
    for l2 ← 0 to place[i] do
        for l3 ← 0 to place[i] − l2 do
            if l2 = 0 ∧ l3 = 0 then
                continue
            end
            total ← l2 + l3;
            if total > place[i] then
                continue
            end
            served ← 12l2 + 48l3;
            if served < clients then
                continue
            end
            power ← 22l2 + 100l3;
            if power > capacity[i] then
                continue
            end
            cost ← 2000l2 + 6000l3;
            if cost < min_cost then
                best_l2, best_l3 ← l2, l3, min_cost ← cost, found ← True;
            end
        end
    end
    if ¬found then
        feasible ← False; break;
    end
    x[i][0] ← best_l2, x[i][1] ← best_l3;
end
if ¬feasible then
    attempts ← attempts + 1; continue;
end
solution ← flatten(x) + flatten(y) + z;
```

Αλγόριθμος 7: Αλγόριθμος Initialize Population (Μέρος 2).

$(feasible, penalty) \leftarrow check\_feasibility(solution, \ldots, \text{return\_penalty} = True)$;
$assignment\_cost \leftarrow getAssignmentCost(solution)$;
if $feasible$ **then**
  return $solution$, $assignment\_cost$;
end
$penalty\_cost \leftarrow assignment\_cost + penalty \cdot 10^6$;
if $penalty\_cost < best\_cost$ **then**
  $best\_cost \leftarrow penalty\_cost$, $best\_attempt \leftarrow solution$;
end
$attempts \leftarrow attempts + 1$;
return $best\_attempt$, ∞

Αλγόριθμος 7: Αλγόριθμος Initialize Population (Μέρος 3).

Output: Best solution `best_sol`, Best cost `best_sol_cost`, Score history `scores`
function `simulated_annealing_metaheuristic`(iterations, temperature, solution, solution_cost, P, N, CL, binaryConstraint, fp_euc_distances, cl_euc_distances, cl_sp_distances) ;
`best_sol` $\leftarrow$ `solution`, `best_sol_cost` $\leftarrow$ `solution_cost`;
`current` $\leftarrow$ `solution`, `current_cost` $\leftarrow$ `solution_cost`;
`scores` $\leftarrow$ [`best_sol_cost`];
for $i \leftarrow 0$ **to** *iterations* **do**
  $t \leftarrow$ `temperature` / $(i + 1)$;
  (`candidate`, `candidate_cost`) $\leftarrow$ `get_neighbor(current)`;
  $z \leftarrow$ last segment of `candidate`;
  `active_facilities` $\leftarrow$ indices i where $z_i = 1$;
  if *candidate is feasible* ***and*** *satisfies distance constraints* **then**
    if *candidate_cost* $<$ *best_sol_cost* ***or*** *random()* $<$ $\exp\left(\frac{current\_cost - candidate\_cost}{t}\right)$ **then**
      `current` $\leftarrow$ `candidate`, `current_cost` $\leftarrow$ `candidate_cost`;
      if *candidate_cost* $<$ *best_sol_cost* **then**
        `best_sol` $\leftarrow$ `candidate`, `best_sol_cost` $\leftarrow$ `candidate_cost`;
        `scores.append(best_sol_cost)`;
      end
    end
  end
  if $i \bmod 100 = 0$ **then**
    Print current progress (iteration, temperature, best cost);
  end
end
return `best_sol`, `best_sol_cost`, `scores`

Αλγόριθμος 8: Αλγόριθμος Simulated Annealing.

Result: Best feasible solution and its cost
function `VNS_algorithm`(k_{max}, $max\_iterations$, $neighborhood\_size$, $solution$, $solution\_cost$, P, N, CL, `binaryConstraint`, `fp_euc_distances`, `cl_euc_distances`);
$iterations \leftarrow 0$;
$current\_sol \leftarrow solution$;
$current\_sol\_cost \leftarrow solution\_cost$;
$x, y, z \leftarrow$ extract parts from $current\_sol$;
if *sum(z) == 0* **then**
    $current\_sol, current\_sol\_cost \leftarrow$ generate random feasible solution;
end
$neighborhoods \leftarrow$ create_neighborhoods($k_{max}, P, neighborhood\_size$);
if $neighborhoods$ *is None* **then**
    return None, ∞;
end
while $iterations < max\_iterations$ **do**
    $k \leftarrow 0$;
    while $k < k_{max}$ **do**
        $(new\_sol, new\_sol\_cost) \leftarrow$ shaking($current\_sol$, $neighborhoods[k]$);
        if *new solution is not feasible* **then**
            $k \leftarrow k + 1$;
            continue;
        end
        $(new\_sol, new\_sol\_cost) \leftarrow$ local_search_vns($neighborhoods[k]$, new_sol);
        if $new\_sol\_cost < current\_sol\_cost$ **then**
            $current\_sol \leftarrow new\_sol$;
            $current\_sol\_cost \leftarrow new\_sol\_cost$;
            $k \leftarrow 0$;
        end
        else
            $k \leftarrow k + 1$;
        end
    end
    if $iterations \mod 10 == 0$ **then**
        print current best solution cost;
    end
    $iterations \leftarrow iterations + 1$;
end
return $current\_sol$, $current\_sol\_cost$;

Αλγόριθμος 9: Αλγόριθμος Variable Neighborhood Search.

Result: Best solution, fitness value, and timing statistics
function `GAFacilityOpt`(filename)
$P, N, CL, clients, points, place, capacity, unaryConstraint, binaryConstraint,$
$fp\_sp\_distances, fp\_euc\_distances, cl\_euc\_distances, cl\_sp\_distances \leftarrow$
$read\_data(filename)$;
$F \leftarrow [2, 3]$, $c\_level \leftarrow [22, 100]$, $D \leftarrow 5$, $budget\_limit \leftarrow 60000$;
$num\_charger\_types \leftarrow length(F)$;
$num\_genes \leftarrow (N \cdot num\_charger\_types) + (CL \cdot N) + N$;
$gene\_space \leftarrow create\_gene\_space()$; `// Define gene boundaries and types`
Solve with Gurobi for optimal cost;
Print "Solving with Gurobi...";
$start\_time \leftarrow current\_time()$;
$st, tc, gurobi\_obj, solving\_time \leftarrow run\_model(filename)$;
$gurobi\_time \leftarrow current\_time() - start\_time$;
Print f"$Gurobi\ objective : gurobi\_obj : .2f in gurobi\_time : .2f sec$";
Initialize population;
$start\_initialization \leftarrow current\_time()$;
$initial\_population \leftarrow []$, $max\_attempts \leftarrow 100$;
for $i \leftarrow 1$ **to** $max\_attempts$ **do**
    $heur\_status, initial\_solution, initial\_cost \leftarrow myopic\_heuristic(...)$;
    if $heur\_status == False$ **then**
        $initial\_solution \leftarrow initialize\_population\_with\_random\_solution(...)$;
    end
    if *initial_solution not in initial_population* **then**
        $initial\_population.append(initial\_solution)$;
    end
end
$end\_initialization \leftarrow current\_time()$;
$init\_time \leftarrow end\_initialization - start\_initialization$;

Αλγόριθμος 10: Γενετικός αλγόριθμος GAFacilityOpt (Μέρος 1).

Create and run genetic algorithm instance;
$ga\_instance \leftarrow pygad.GA($
$num\_generations = 1000,$
$num\_parents\_mating = \min(100, |initial\_population|),$
$sol\_per\_pop = |initial\_population|,$
$num\_genes = num\_genes,$
$fitness\_func = fitness\_func(...),$
$parent\_selection\_type = \text{"}rank\text{"},$
$keep\_parents = \min(20, |initial\_population|),$
$crossover\_type = \text{"}single\_point\text{"},$
$mutation\_type = \text{"}random\text{"},$
$mutation\_percent\_genes = 10,$
$initial\_population = initial\_population,$
$stop\_criteria = [\text{"}saturate\_200\text{"}],$
$parallel\_processing = [\text{"}thread\text{"}, 4]$
);
$start\_ga \leftarrow current\_time();$
$ga\_instance.run();$
$end\_ga \leftarrow current\_time();$
$ga\_time \leftarrow end\_ga - start\_ga;$
$total\_time \leftarrow init\_time + ga\_time;$
$solution, solution\_fitness, \_ \leftarrow ga\_instance.best\_solution();$
$x, y, z \leftarrow decode\_solution(solution, N, F, CL);$
Print basic results (fitness, open facilities, total distance);
Print variables x[i][j], y[k][i], z[i];
Calculate GAP;
if $solution\_fitness == -\infty$ **then**
    $gap \leftarrow \text{"}-\text{"};$
    Print "No feasible solution found by GA.";
end
else
    $best\_cost \leftarrow -solution\_fitness;$
    $gap \leftarrow \frac{(best\_cost - gurobi\_obj)}{gurobi\_obj} \cdot 100;$
    Print GA best cost, Gurobi optimal, GAP;
end
Save results to file;
$output\_file \leftarrow \text{"}results\_GA\_\text{"} +$ basename(dirname(filename)) $+ \text{"}.txt\text{"};$
Open file and write CSV row with timing, fitness, and GAP;

Αλγόριθμος 10: Γενετικός αλγόριθμος GAFacilityOpt (Μέρος 2).

Κεφάλαιο 4

Υπολογιστική μελέτη

4.1 Διαδικασία πειράματος

Για την υλοποίηση του πειράματος αναπτύχθηκε μια γεννήτρια προβλημάτων, η οποία παράγει τυχαία σενάρια σε ένα δυσδιάστατο πλέγμα. Η γεννήτρια λαμβάνει ως είσοδο το μέγεθος του πλέγματος, το συνολικό αριθμό σημείων (K), τον αριθμό πελατών (CL), τον αριθμό σταθμών φόρτισης προς εγκατάσταση (P) και τον αριθμό προβλημάτων που θα παραχθούν (PR). Αρχικά, τοποθετούνται τυχαία σημεία στο πλέγμα τα οποία χαρακτηρίζονται ως υποψήφιες εγκαταστάσεις. Απο αυτά τα σημεία, επιλέγονται τυχαία CL που αφορούν τους πελάτες, εξασφαλίζοντας οτι δεν υπάρχουν επικαλύψεις. Για κάθε ζεύγος σημείων υπολογίζονται οι ευκλείδιες και οι Manhattan αποστάσεις, οι οποίες χρησιμοποιούνται για τη δημιουργία περιορισμών απόστασης μεταξύ εγκαταστάσεων και μεταξύ εγκαταστάσεων και πελατών με βάση παραμέτρους αυστηρότητας. Επιπλέον, για κάθε υποψήφια εγκατάσταση δημιουργούνται τυχαίες τιμές που αντιστοιχούν στον αριθμό θέσεων φόρτισης και στη διαθέσιμη ισχύ ανά εγκατάσταση. Όλα τα δεδομένα αποθηκεύονται σε αρχεία κειμένου, οργανωμένα σε φακέλους ανάλογα με τις παραμέτρους του προβλήματος. Ειδικότερα, στη πρώτη σειρά του αρχείου αναγράφονται τέσσερεις αριθμοί. Ο πρώτος αριθμός αντιστοιχεί στο συνολικό αριθμό σημείων, ο δεύτερος στον αριθμό των πελατών, ο τρίτος στο πλήθος των υποψήφιων θέσεων προς εγκατάσταση (N = K - CL) και ο τέταρτος στον αριθμό των εγκαταστάσεων που χρειάζεται να ανοίξουν.

Για το πείραμα αυτό δημιουργήθηκαν 12 κλάσεις προβλημάτων, με 10 προβλήματα η κάθε μια, άρα συνολικά έχουμε 120 προβλήματα (pmed1,pmed2 εώς pmed120) που αφορούν τα προβλήματα p μέσου περιορισμένης χωρητικότητας. Στην παρού-

σα διπλωματική εργασία η γλώσσα που χρησιμοποιήθηκε για την υλοποίηση των αλγορίθμων είναι η Python στο περιβάλλον Visual Studio Code έκδοση 1.100.0. Οι πειραματικές δοκιμές πραγματοποιήθηκαν σε σύστημα με επεξεργαστή AMD A6-9225 dual-core στα 2.60GHz, 8GB DDR4 RAM, λειτουργικό σύστημα Windows 10 64-bit. Στη διαδικασία εκτέλεσης των αλγορίθμων, έτρεξαν αρχικά όλα τα προβλήματα για έναν αλγόριθμο και μετά ακολούθησε η ίδια διαδικασία και για τους υπόλοιπους αλγόριθμους. Για την εκτέλεση κάποιων αλγορίθμων εφαρμόστηκε ένας περιορισμός χρόνου. Πιο συγκεκριμένα, αν ο χρόνος εκτέλεσης ξεπερνούσε τη μια ώρα, η διαδικασία διακοπτόταν και εμφανιζόταν η λύση που είχε βρεθεί μέχρι τότε.

4.2 Σύγκριση αλγορίθμων DFS και BestFS

4.2.1 Min Distance heuristic

Στον Πίνακα 4.1 παρουσιάζονται τα αποτελέσματα των μετρήσεων για τις μεθόδους DFS και BestFS στον Branch & Bound αλγόριθμο, χρησιμοποιώντας την ευρετική μέθοδο Min Distance. Οι τιμές που καταγράφονται αφορούν το γεωμετρικό χρόνο εκτέλεσης (geomean) για κάθε μια απο τις κλάσεις των προβλημάτων. Όπως φαίνεται στον πίνακα, οι χρόνοι εκτέλεσης για τον αλγόριθμο DFS κυμαίνονται από 0.23 δευτερόλεπτα για μικρές κλάσεις έως 374.69 δευτερόλεπτα για τις μεγαλύτερες κλάσεις, ενώ για τον BestFS οι αντίστοιχοι χρόνοι κυμαίνονται από 0.22 έως 300.07 δευτερόλεπτα. Παρατηρούμε ότι σε 9 απο τις 12 κλάσεις προβλημάτων ο BestFS παρουσιάζει μικρότερο χρόνο εκτέλεσης σε σχέση με τον DFS, καθώς επικεντρώνεται σε πιο υποσχόμενους κόμβους στο χώρο αναζήτησης. Οι δύο αλγόριθμοι έχουν παρόμοια απόδοση σε μικρότερα προβλήματα και πιο συγκεκριμένα φαίνεται οτι ο DFS αλγόριθμος είναι πιο αποδοτικός στις κλάσεις 36_15_10, 36_11_10 και 100_80_10. Ωστόσο, στις μεγαλύτες κλάσεις που είναι και πιο απαιτητικές ο BestFS υπερέχει σημαντικά στους γεωμετρικούς χρόνους εκτέλεσης. Πιο συγκεκριμένα, παρατηρούμε οτι στην κλάση 100_60_15 η εκτέλεση διαρκεί 2.83 δευτερόλεπτα και στην 225_40_10, 52.33 δευτερόλεπτα, δηλαδή είναι δύο και σχεδόν τρείς φορές ταχύτερη σε σχέση με τον DFS, που απαιτεί 4.06 και 140.10 δευτερόλεπτα αντίστοιχα. Στην τελευταία και μεγαλύτερη κλάση 324_80_20 ο αλγόριθμος BestFS είναι σχεδόν κατά 75 δευτερόλεπτα ταχύτερος σε σχέση με τον DFS. Επίσης,

ο DFS φαίνεται να έχει μεγαλύτερη διακύμανση στις μετρήσεις συγκρίνοντας τους γεωμετρικούς του χρόνους. Δηλαδή, οι χρόνοι εκτέλεσης μεταβάλλονται σημαντικά μεταξύ των διαφορετικών προβλημάτων μιας κλάσης. Αντιθέτως, ο BestFS παρουσιάζει μεγαλύτερη σταθερότητα στις τιμές του καθώς στις περισσότερες κλάσεις δεν έχει μεγάλες διακυμάνσεις ανάλογα με το πρόβλημα. Συνολικά, ο γεωμετρικός μέσος χρόνος εκτέλεσης για τον DFS είναι 1.51, ενώ για τον BestFS είναι χαμηλότερος, στα 1.25 δευτερόλεπτα.

Πίνακας 4.1: Αποτελέσματα μετρήσεων DFS και BestFS με Min Distance heuristic.

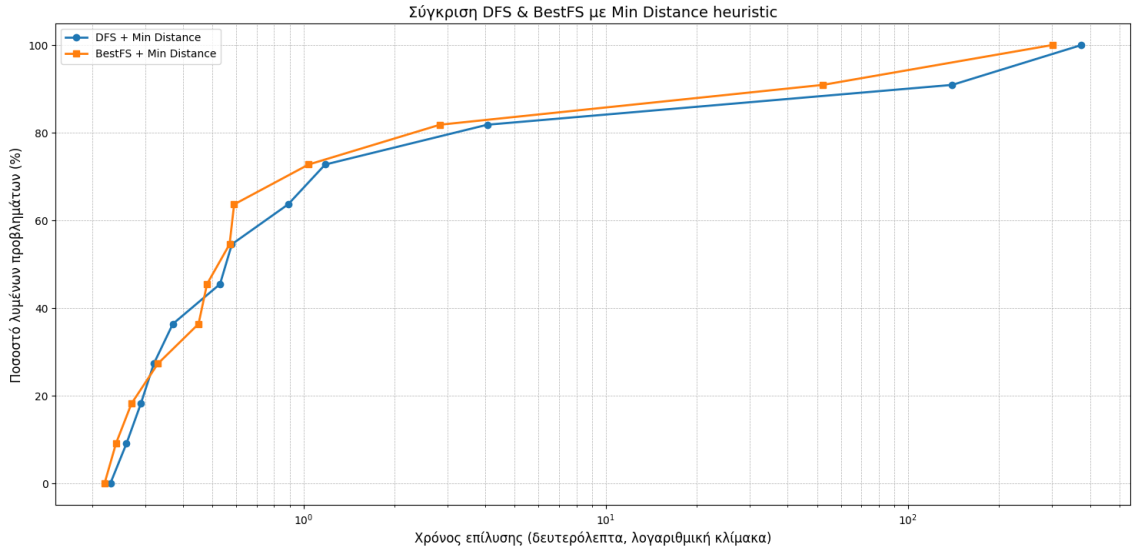
Κλάσεις	**Χρόνος DFS με Min Distance heuristic**	**Χρόνος BestFS με Min Distance heuristic**
25_10_5	0.29	0.27
25_15_5	0.26	0.22
36_15_10	0.23	0.24
36_11_10	0.32	0.33
64_25_10	0.58	0.57
64_39_10	0.89	0.59
100_20_10	0.53	0.48
100_80_10	0.37	0.45
100_40_15	1.18	1.04
100_60_15	4.06	2.83
225_40_15	140.10	52.33
324_80_20	374.69	300.07
Γεωμετρικός Μέσος	1.51	1.25

Στο Σχήμα 4.1 παρουσιάζεται το διάγραμμα που απεικονίζει το ποσοστό λυμένων προβλημάτων ανά το χρόνο για τους αλγορίθμους DFS και BestFS που αναπτύσσονται με την Min Distance ευρετική μέθοδο. Όπως φαίνεται και απο το διάγραμμα, ο BestFS αλγόριθμος επιλύει μεγαλύτερο ποσοστό προβλημάτων σε μικρότερους χρόνους σε σχέση με τον DFS, παρουσιάζοντας πιο σταθερή συμπεριφορά. Αντιθέτως, ο DFS αν και είναι εξίσου αποδοτικός στις μικρότερες κλάσεις, παρουσιάζει μειωμένη απόδοση στις μεγαλύτερες έχοντας σημαντικά υψηλότερους χρόνους.

4.2.2 Distance from 0.5 heuristic

Στον Πίνακα 4.2 παρουσιάζονται τα αποτελέσματα των μετρήσεων για τις μεθόδους DFS και BestFS, χρησιμοποιώντας την ευρετική μέθοδο Distance from 0.5. Οι τιμές που καταγράφονται αφορούν το γεωμετρικό χρόνο εκτέλεσης (geomean) για κάθε μία από τις κλάσεις των προβλημάτων. Στον πίνακα, οι χρόνοι εκτέλεσης για τον DFS κυμαίνονται από 0.20 έως 466.02 δευτερόλεπτα, ενώ για τον BestFS κυμαίνονται από 0.23 έως 335.36 δευτερόλεπτα. Όπως παρατηρούμε, σε 9 απο

Σχήμα 4.1: Ποσοστό λυμένων προβλημάτων vs Χρόνος Min Distance Heuristic.



τις 12 κλάσεις ο BestFS αλγόριθμος έχει καλύτερο χρόνο εκτέλεσης σε σχέση με τον DFS. Στις υπόλοιπες κλάσεις και πιο συγκεκριμένα στις 36_15_10, 64_25_10 και 64_39_10 ο BestFS φαίνεται να έχει μεγαλύτερο γεωμετρικό χρόνο εκτέλεσης απο ότι ο DFS. Παρόλο που υπάρχουν μικρές διαφοροποιήσεις, οι χρόνοι είναι πολύ κοντά για τις μικρές κλάσεις. Η διαφορά είναι εντονότερη στις μεγαλύτερες κλάσεις, όπου οι χρόνοι εκτέλεσης είναι αυξημένοι και για τους δύο αλγορίθμους, με τον BestFS να έχει ελαφρώς καλύτερο χρόνο. Στην κλάση 225_40_15, ο DFS απαιτεί 137.31 δευτερόλεπτα, ενώ ο BestFS χρειάζεται 91.73 δευτερόλεπτα, μια διαφορά περίπου 45 δευτερολέπτων. Στην κλάση 324_80_20, η διαφορά είναι επίσης σημαντική, με τον DFS να χρειάζεται 466.02 δευτερόλεπτα και τον BestFS να εκτελείται σε 335.36 δευτερόλεπτα, μια διαφορά που ανέρχεται στα 130.66 δευτερόλεπτα υπερ του Best First Search. Συνολικά, ο γεωμετρικός μέσος χρόνος εκτέλεσης για τον DFS είναι 1.70, ενώ για τον BestFS είναι ελαφρώς χαμηλότερος, στα 1.60 δευτερόλεπτα.

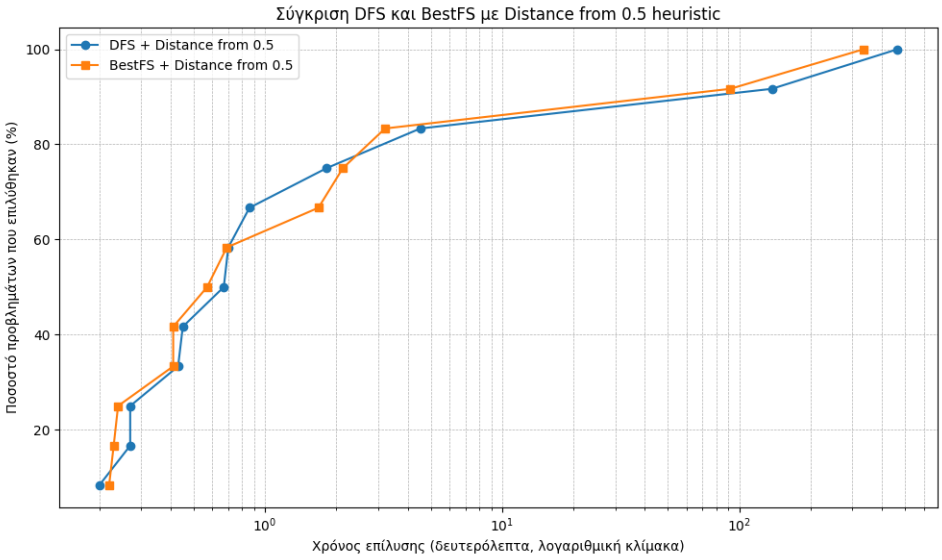
Στο Σχήμα 4.2 παρουσιάζεται το διάγραμμα που απεικονίζει το ποσοστό λυμένων προβλημάτων ανά το χρόνο για τους αλγορίθμους DFS και BestFS που αναπτύσσονται με την Distance from 0.5 ευρετική μέθοδο. Όπως φαίνεται και απο το διάγραμμα, για προβλήματα που ανήκουν σε μικρότερες κλάσεις και οι δύο αλγόριθμοι έχουν παρόμοια απόδοση λύνοντας σχεδόν ίδιο ποσοστό προβλημάτων σε κοντινούς χρόνους. Όσο όμως το μέγεθος των προβλημάτων αυξάνεται, ο BestFS έχει καλύτερη απόδοση λύνοντας μεγαλύτερο ποσοστό προβλημάτων σε συντομότερο

Πίνακας 4.2: Αποτελέσματα μετρήσεων DFS και BestFS με Distance from 0.5 heuristic.

Κλάσεις	Χρόνος DFS με Distance from 0.5 heuristic	Χρόνος BestFS με Distance from 0.5 heuristic
25_10_5	0.27	0.23
25_15_5	0.27	0.24
36_15_10	0.20	0.22
36_11_10	0.43	0.41
64_25_10	0.67	0.69
64_39_10	0.86	2.13
100_20_10	0.70	0.57
100_80_10	0.45	0.41
100_40_15	1.82	1.68
100_60_15	4.52	3.21
225_40_15	137.31	91.73
324_80_20	466.02	335.36
Γεωμετρικός Μέσος	1.70	1.60

χρόνο απο οτι ο DFS.

Σχήμα 4.2: Ποσοστό λυμένων προβλημάτων vs Χρόνος Distance from 0.5 Heuristic.



4.3 Σύγκριση αλγορίθμων με Min Distance heuristic και Distance from 0.5 heuristic.

Παρακάτω θα συγκρίνουμε τις δύο διαφορετικές ευρετικές μεθόδους που χρησιμοποιήθηκαν για την επιλογή μεταβλητής προς διακλάδωση στο Branch and Bound αλγόριθμο. Η ευρετική μέθοδος Min Distance επιλέγει προς διακλάδωση τη μεταβλητή που έχει τη μικρότερη συνολική απόσταση απο τους κόμβους ζήτησης, ενώ στην ευρετική μέθοδο Distance from 0.5 επιλέγεται προς διακλάδωση η μεταβλητή που έχει μικρότερη απόσταση απο τον αριθμό 0.5. Τα αποτελέσματα απο τον Πίνακα 4.3 δείχνουν ότι η πρώτη ευρετική μέθοδος είναι καλύτερη απο τη δεύτερη,

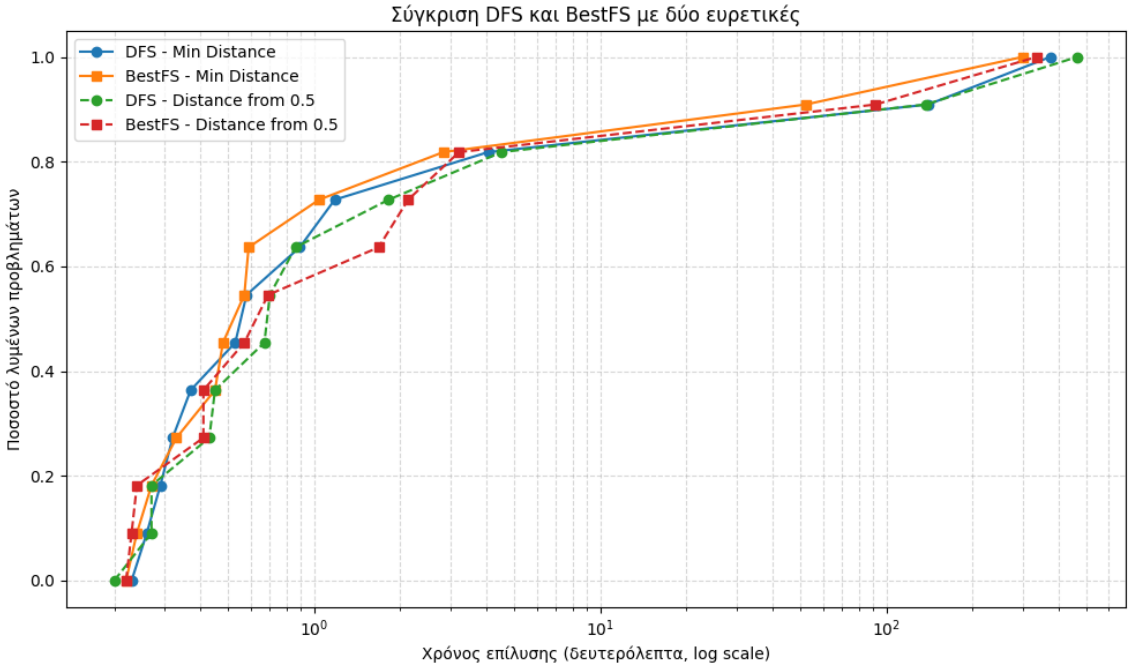
καθώς έχει σε 9 απο τις 12 κλάσεις μικρότερους γεωμετρικούς χρόνους τόσο στην αναζήτηση κατά πλάτος όσο και στην αναζήτηση πρώτα στο καλύτερο. Για τις μικρές κλάσεις, όπως οι 25_10_5 και 25_15_5, οι χρόνοι εκτέλεσης είναι σχετικά κοντά και η ευρετική μέθοδος Distance from 0.5 δεν παρουσιάζει σημαντική διαφοροποίηση σε σχέση με την ευρετική μέθοδο Min Distance. Σε κλάσεις όπως η 36_15_10 και 64_39_10, η Distance from 0.5 φαίνεται να προσφέρει μικρές βελτιώσεις για τον DFS σε σχέση με την Min Distance , ενώ για τον BestFS ο χρόνος φαίνεται καλύτερος μόνο για την πρώτη κλάση. Στις μεγαλύτερες κλάσεις, όπως η 225_40_15 και η 324_80_20 η ευρετική μέθοδος Min Distance έχει σημαντικά καλύτερους χρόνους τόσο για τον DFS αλγόριθμο όσο και για τον BestFS. Συνολικά, η Min Distance ευρετική μέθοδος έχει γεωμετρικούς μέσους 1.51 και 1.25 δευτερόλεπτα για την αναζήτηση με DFS και BestFS αντίστοιχα, ενώ η Distance from 0.5 έχει 1.70 και 1.60 δευτερόλεπτα αντίστοιχα. Επομένως, η Min Distance είναι πιο αποδοτική καθώς η επιλογή μεταβλητής προς διακλάδωση γίνεται με βάση μια πιο ολοκληρωμένη εκτίμηση του προβλήματος.

Πίνακας 4.3: Αποτελέσματα DFS και BestFS με Min Distance heuristic και Distance from 0.5 heuristic.

Κλάσεις	DFS Min Distance heuristic	DFS Distance from 0.5 heuristic	BestFS Min Distance heuristic	BestFS Distance from 0.5 heuristic
25_10_5	0.29	0.27	0.27	0.23
25_15_5	0.26	0.27	0.22	0.24
36_15_10	0.23	0.20	0.24	0.22
36_11_10	0.32	0.43	0.33	0.41
64_25_10	0.58	0.67	0.57	0.69
64_39_10	0.89	0.86	0.59	2.13
100_20_10	0.53	0.70	0.48	0.57
100_80_10	0.37	0.45	0.45	0.41
100_40_15	1.18	1.82	1.04	1.68
100_60_15	4.06	4.52	2.83	3.21
225_40_15	140.10	137.31	52.33	91.73
324_80_20	374.69	466.02	300.07	335.36
Γεωμετρικός Μέσος	1.51	1.70	1.25	1.60

Στο Σχήμα 4.3 παρουσιάζεται το διάγραμμα που απεικονίζει το ποσοστό λυμένων προβλημάτων ανά το χρόνο για τους αλγορίθμους DFS και BestFS που αναπτύσσονται με τις ευρετικές μεθόδους Min Distance και Distance from 0.5. Παρατηρούμε ότι οι BestFS αλγόριθμοι υπερτερούν απο τους DFS, ανεξαρτήτως ευρετικής. Όσον αφορα τις ερευτικές μεθόδους, η Min Distance φαίνεται να επιλύει μεγαλύτερο ποσοστό προβλημάτων σε μικρότερο χρόνο απο οτι η Distance from 0.5 τόσο για τον DFS όσο και για τον BestFS αλγόριθμο. Επομένως, ο αλγόριθμος BestFS με ευρετική μέθοδο την Min Distance έχει την καλύτερη συνολική απόδοση λύνοντας περισσότερα προβλήματα ταχύτερα.

Σχήμα 4.3: Ποσοστό λυμένων προβλημάτων vs Χρόνος Min Distance και Distance from 0.5 Heuristic



4.4 Σύγκριση Simulated Annealing και Variable Neighborhood Search αλγορίθμων με Min Distance heuristic BestFS.

Ακολουθεί η σύγκριση μεταξύ της βασικής εκδοχής του Branch and Bound με BestFS και Min Distance heuristic και των παραλλαγών της, που ενσωματώνουν τις λύσεις των μεθευρετικών Simulated Annealing και Variable Neighborhood Search ως warm-start. Στον Πίνακα 4.4 παρουσιάζονται οι γεωμετρικοί χρόνοι εκτέλεσης των αλγορίθμων και εντός των παρενθέσεων δηλώνεται ο αριθμός των προβλημάτων, απο τα 10 που έχει κάθε κλάση, στα οποία κάθε παραλλαγή ήταν ταχύτερη απο τον αρχικό Branch and Bound που υλοποιήθηκε χωρίς warm start. Όπως φαίνεται απο τα αποτελέσματα στον πίνακα, οι χρονικές διαφορές είναι μικρές και σε κάποιες περιπτώσεις υπάρχει επιβάρυνση του χρόνου η οποία επιβεβαιώνεται και απο τους γεωμετρικούς μέσους για τον Simmulated Annealing (1.40) και τον Variable Neighborhood Search (0.74), έναντι του BestFS (1.25). Από την ανάλυση των αποτελεσμάτων προκύπτει στην επίλυση με άνω όριο απο τον Simulated Annealing ότι τα 52 από τα 120 συνολικά προβλήματα (ποσοστό 43.30%), επιλύονται πιο γρήγορα όταν χρησιμοποιείται το άνω όριο. Όσον αφορά την επίλυση με άνω όριο απο τον VNS, σε 93 από τα 120 προβλήματα (ποσοστό 77.50%), η ενσωμάτωση του άνω

ορίου συνέβαλε σε ταχύτερη επίλυση, υποδηλώνοντας μια περιορισμένη συμβολή στη μείωση του χρόνου σε ορισμένες περιπτώσεις. Αξίζει να αναφερθεί ότι σε ορισμένες περιπτώσεις (τέσσερεις κλάσεις), η απουσία λύσης από τους μεθευρετικούς οδήγησε σε εκτέλεση του Branch and Bound χωρίς warm start, κάτι που επηρεάζει τις συγκρίσεις χρόνου. Ως εκ τούτου, το αυξημένο ποσοστό ταχύτερων επιλύσεων με άνω όριο δεν θα πρέπει να ερμηνευθεί ως απόλυτη υπεροχή της συγκεκριμένης παραλλαγής, καθώς η διαφορά ενδέχεται να οφείλεται και σε εξωτερικούς υπολογιστικούς παράγοντες. Επομένως, η χρήση των μεθευρετικών Simulated Annealing και Variable Neighborhood Search για την παραγωγή άνω ορίου στο BestFS έχουν αμελητέες βελτιώσεις στο χρόνο και αναγνωρίζοντας την πολυπλοκότητα που εισάγεται στο πρόβλημα, δεν κρίνονται αποδοτικοί για τη βελτίωση του χρόνου εκτέλεσης στο πρόβλημα μας.

Πίνακας 4.4: Αποτελέσματα μετρήσεων BestFS, Min Distance heuristic με SA και VNS.

Κλάσεις	**Χρόνος BestFS με Min Distance heuristic**	**Χρόνος SA BestFS**	**Χρόνος VNS BestFS**
25_10_5	0.27	0.17 (8)	0.09 (9)
25_15_5	0.22	0.13 (9)	0.08 (10)
36_15_10	0.24	0.15 (9)	0.08 (9)
36_11_10	0.33	0.26 (7)	0.11 (10)
64_25_10	0.57	0.49 (5)	0.30 (9)
64_39_10	0.59	0.49 (6)	0.23 (10)
100_20_10	0.48	1.77 (0)	0.34 (7)
100_80_10	0.45	0.48 (3)	0.37 (5)
100_40_15	1.04	2.64 (1)	1.36 (2)
100_60_15	2.83	6.65 (2)	2.88 (3)
225_40_15	52.33	56.62 (2)	43.67 (10)
324_80_20	300.07	346.01 (0)	291.33 (9)
Γεωμετρικός Μέσος	1.25	1.40	0.74

4.5 Σύγκριση Gurobi και γενετικού αλγορίθμου GAFacilityOpt

Στον Πίνακα 4.5 παρουσιάζεται η σύγκριση των δύο μεθόδων επίλυσης: του λύτη Gurobi και του γενετικού αλγορίθμου GAFacilityOpt, με βάση το γεωμετρικό χρόνο επίλυσης των προβλημάτων κάθε κλάσης. Όπως προκύπτει από τα αποτελέσματα, ο λύτης Gurobi εμφανίζει μεγάλη υπεροχή στους χρόνους επίλυσης, οι οποίοι κυμαίνονται από 0.08 έως 56.66 δευτερόλεπτα. Αντίθετα, ο GAFacilityOpt απαιτεί σημαντικά μεγαλύτερους χρόνους, που φτάνουν από δεκάδες μέχρι και χιλιάδες δευτερόλεπτα. Ενδεικτικά, καταγράφονται χρόνοι 9242.75 και 30698.10 δευτερόλεπτα για τις κλά-

σεις 225_40_15 και 324_80_20 αντίστοιχα. Η διαφορά αυτή παραμένει σταθερή και αυξάνεται καθώς μεγαλώνει το μέγεθος των προβλημάτων, υποδεικνύοντας ότι ο Gurobi είναι σαφώς πιο αποδοτικός σε υπολογιστικό χρόνο. Αξιοσημείωτη είναι επίσης η διαφορά στους γεωμετρικούς μέσους χρόνους εκτέλεσης: για τον Gurobi είναι μόλις 0.32 δευτερόλεπτα, ενώ για το GAFacilityOpt ανέρχεται στα 437.74 δευτερόλεπτα, δείχνοντας μια διαφορά τάξης μεγέθους υπέρ του Gurobi. Συνεπώς, η υπολογιστική σύγκριση καταδεικνύει ξεκάθαρα την υπεροχή του Gurobi σε όρους ταχύτητας, καθιστώντας τον πιο κατάλληλο για προβλήματα όπου ο χρόνος επίλυσης αποτελεί κρίσιμο παράγοντα.

Πίνακας 4.5: Αποτελέσματα μετρήσεων λύτη Gurobi και GAFacilityOpt.

Κλάσεις	**Χρόνος Gurobi**	**Χρόνος GAFacilityOpt**
25_10_5	0.09	26.43
25_15_5	0.08	13.01
36_15_10	0.09	113.08
36_11_10	0.08	119.39
64_25_10	0.13	313.71
64_39_10	0.11	259.61
100_20_10	0.19	943.41
100_80_10	0.14	366.5
100_40_15	0.22	1394.75
100_60_15	0.17	957.04
225_40_15	21.93	9242.75
324_80_20	56.66	30698.10
Γεωμετρικός Μέσος	0.32	437.74

Στον Πίνακα 4.6 παρουσιάζεται το μέσο σχετικό σφάλμα ($GAP\%$) των λύσεων που προέκυψαν από το γενετικό αλγόριθμο GAFacilityOpt σε σύγκριση με τις βέλτιστες λύσεις του λύτη Gurobi για κάθε κλάση προβλημάτων. Το σχετικό GAP αποτυπώνει την ποσοστιαία απόκλιση της αντικειμενικής τιμής του γενετικού αλγορίθμου από τη βέλτιστη που βρίσκει ο Gurobi, προσφέροντας έτσι ένα μέτρο της ακρίβειας των προσεγγιστικών λύσεων. Οι τιμές δείχνουν μεταβλητότητα μεταξύ των κλάσεων, με ορισμένες περιπτώσεις να έχουν χαμηλό GAP, όπως οι κλάσεις 36_15_10 με 2.22% και 64_39_10 με 3.41%, ενώ υπάρχουν κλάσεις που φτάνει σε υψηλότερα επίπεδα, όπως στη 225_40_15 με 60.96% και στη 324_80_20 με 48.40%. Αξίζει να σημειωθεί ότι ο υπολογισμός του μέσου GAP ανά κλάση βασίστηκε μόνο στα προβλήματα για τα οποία ο αλγόριθμος κατάφερε να εντοπίσει εφικτή λύση. Όσα απέτυχαν να δώσουν τέτοια λύση εξαιρέθηκαν, καθώς δεν ήταν δυνατός ο υ-

πολογισμός της απόκλισης. Ο γεωμετρικός μέσος του σχετικού σφάλματος για όλες τις κλάσεις διαμορφώνεται στο 11.83%, τιμή που υποδηλώνει μια σχετικά καλή συνολική ακρίβεια των λύσεων του αλγορίθμου. Συνολικά, τα αποτελέσματα υποδεικνύουν ότι ο γενετικός αλγόριθμος μπορεί να παράγει ικανοποιητικές λύσεις σε αρκετές περιπτώσεις, ωστόσο η απόδοσή του διαφοροποιείται αισθητά ανάλογα με τα χαρακτηριστικά του κάθε προβλήματος.

Πίνακας 4.6: Σύγκριση Μέσου GAP (%) μεταξύ Gurobi και GAFacilityOpt.

Κλάσεις	**Average GAP %**
25_10_5	6.45
25_15_5	3.68
36_15_10	2.22
36_11_10	22.72
64_25_10	18.11
64_39_10	3.41
100_20_10	30.94
100_80_10	4.47
100_40_15	24.74
100_60_15	10.16
225_40_15	60.96
324_80_20	48.40
Γεωμετρικός Μέσος	11.83

Κεφάλαιο 5

Συμπεράσματα

Στην παρούσα εργασία μελετήθηκε το πρόβλημα χωροθέτησης σταθμών φόρτισης ηλεκτρικών οχημάτων, το οποίο αποτελεί μια σημαντική πρόκληση για τη βιώσιμη ανάπτυξη της ηλεκτροκίνησης. Στη βιβλιογραφία, το πρόβλημα έχει προσεγγιστεί από διαφορετικές σκοπιές, αυτή των διαχειριστών δικτύου, των επενδυτών και των ίδιων των οδηγών, με στόχο τη βέλτιστη επιλογή τοποθεσιών. Έχουν εφαρμοστεί τόσο ακριβείς μέθοδοι, όπως οι Branch and Bound και η αποσύνθεση Benders, όσο και προσεγγιστικές, ευρετικές και μεθευρετικές τεχνικές, όπως άπληστοι αλγόριθμοι, local search, γενετικοί αλγόριθμοι και simulated annealing. Στο πλαίσιο αυτό, η παρούσα εργασία επέλεξε να εστιάσει στη βελτίωση της προσβασιμότητας και της ποιότητας εξυπηρέτησης των σταθμών φόρτισης, μέσω μεθόδων επίλυσης που συνδυάζουν υπολογιστική αποδοτικότητα και πρακτική εφαρμοσιμότητα.

Για την επίλυση του προβλήματος χωροθέτησης p median, υλοποιήθηκαν 13 αλγόριθμοι οι οποίοι αξιολογήθηκαν ως προς την απόδοση και την αποτελεσματικότητα τους. Συγκεκριμένα, υλοποιήθηκε αλγόριθμος για τη μοντελοποίηση του ακέραιου γραμμικού προβλήματος με τη χρήση του λύτη Gurobi ώστε να βρεθεί η βέλτιστη λύση για κάθε πρόβλημα. Στη συνέχεια, αναπτύχθηκαν και υλοποιήθηκαν 4 διαφορετικές μέθοδοι Branch and Bound, οι οποίες διαφοροποιούνται ως προς την τεχνική αναζήτησης: δύο βασίζονται σε βάθος πρώτα (DFS) και δύο σε βέλτιστη αναζήτηση πρώτα (BestFS). Επιπλέον, για τη βελτίωση της αποδοτικότητας, κάθε μέθοδος συνδυάστηκε με δύο διαφορετικές ευρετικές στρατηγικές επιλογής κόμβων: τη μέθοδο Min Distance και τη μέθοδο Distance from 0.5. Ακολουθούν 2 υλοποιήσεις του Branch and Bound με χρήση της βέλτιστης αναζήτησης (BestFS) και της ευρετικής Min Distance, όπου στην πρώτη υλοποίηση ως ανώτατο όριο (upper bound)

χρησιμοποιείται η λύση που προκύπτει από το μεθευρετικό αλγόριθμο Simulated Annealing, ενώ στη δεύτερη η λύση που παράγεται από το μεθευρετικό αλγόριθμο Variable Neighborhood Search (VNS). Για την παραγωγή αρχικών λύσεων, οι οποίες αξιοποιούνται ως αφετηρία από τις μεθευρετικές μεθόδους, χρησιμοποιήθηκαν ο μυωπικός αλγόριθμος και ο Randomized Construction. Τέλος, αναπτύχθηκε ο γενετικός αλγόριθμος GAFacilityOpt, ο οποίος αξιοποιεί τόσο το μυωπικό αλγόριθμο όσο και τον Initialize Population για την αρχικοποίηση του πληθυσμού.

Από την εκτέλεση κάθε αλγορίθμου μεμονωμένα, προέκυψαν τα εξής βασικά συμπεράσματα: Αρχικά, πραγματοποιήθηκε η επίλυση του προβλήματος μέσω της μοντελοποίησης του ως γραμμικό ακέραιο πρόβλημα, με τη χρήση του λύτη Gurobi. Η διαδικασία αυτή επέτρεψε την εύρεση των βέλτιστων λύσεων για κάθε πρόβλημα, ώστε να μπορούν να χρησιμοποιηθούν ως σημείο αναφοράς για τη σύγκριση των υπόλοιπων μεθόδων. Ο γεωμετρικός μέσος χρόνος εκτέλεσης για τη συγκεκριμένη διαδικασία ήταν 0.32 δευτερόλεπτα.

Στη συνέχεια, εκτελέστηκαν οι μέθοδοι Branch and Bound. Αρχικά εφαρμόστηκε η παραλλαγή με αναζήτηση κατά βάθος (DFS) σε συνδυασμό με την ευρετική μέθοδο Min Distance. Η συγκεκριμένη προσέγγιση κατάφερε να βρεί βέλτιστη λύση και για τα 120 προβλήματα εντός του προκαθορισμένου χρονικού ορίου. Ο γεωμετρικός μέσος όρος του χρόνου εκτέλεσης ανήλθε σε 1.51 δευτερόλεπτα, δηλαδή 1.19 δευτερόλεπτα περισσότερα σε σύγκριση με τη λύση που προέκυψε μέσω του Gurobi.

Ακολούθως, εφαρμόστηκε η παραλλαγή του Branch and Bound με αναζήτηση κατά βάθος (DFS) σε συνδυασμό με την ευρετική μέθοδο Distance from 0.5. Ο αλγόριθμος πέτυχε επίσης βέλτιστη λύση για όλα τα προβλήματα, με γεωμετρικό μέσο χρόνο εκτέλεσης 1.70 δευτερόλεπτα, δηλαδή 0.19 δευτερόλεπτα περισσότερο σε σύγκριση με τη μέθοδο που χρησιμοποιεί την ευρετική Min Distance.

Στη συνέχεια, εφαρμόστηκε η παραλλαγή του Branch and Bound με στρατηγική βέλτιστης αναζήτησης πρώτα (BestFS), σε συνδυασμό με την ευρετική μέθοδο Min Distance. Η προσέγγιση αυτή κατάφερε να εντοπίσει βέλτιστη λύση για όλα τα προβλήματα, με γεωμετρικό μέσο χρόνο εκτέλεσης 1.25 δευτερόλεπτα. Συγκριτικά, η μέθοδος αυτή αποδείχθηκε ταχύτερη από τις δύο προηγούμενες παραλλαγές με DFS, ενώ παραμένει κατά 0.93 δευτερόλεπτα πιο αργή από τη λύση που προέκυψε

μέσω του Gurobi.

Η παραλλαγή του Branch and Bound με στρατηγική βέλτιστης αναζήτησης πρώτα και χρήση της ευρετικής μεθόδου Distance from 0.5 εντόπισε τη βέλτιστη λύση για κάθε πρόβλημα, με γεωμετρικό μέσο χρόνο εκτέλεσης 1.60 δευτερόλεπτα. Ο χρόνος αυτός είναι αυξημένος κατά 0.35 δευτερόλεπτα σε σύγκριση με την αντίστοιχη παραλλαγή του BestFS που χρησιμοποιεί την ευρετική Min Distance.

Αφού εκτελέστηκαν οι παραπάνω παραλλαγές του Branch and Bound, διαπιστώθηκε ότι η μέθοδος με βέλτιστη αναζήτηση πρώτα (BestFS) σε συνδυασμό με την ευρετική Min Distance παρουσιάζει την καλύτερη απόδοση (1.25 δευτερόλεπτα) σε σχέση με τις υπόλοιπες μεθόδους που μελετήθηκαν. Για το λόγο αυτό, επιλέχθηκε αυτή η παραλλαγή ως βάση για την υλοποίηση των υπόλοιπων Branch and Bound μεθόδων, οι οποίες χρησιμοποιούν ως ανώτατο όριο (upper bound) τις λύσεις που προκύπτουν από τους μεθευρετικούς αλγορίθμους Simulated Annealing και Variable Neighborhood Search (VNS).

Πιο συγκεκριμένα, για την υλοποίηση του Branch and Bound με άνω όριο τη λύση του μεθευρετικού αλγορίθμου Simulated Annealing, ο γεωμετρικός μέσος χρόνος εκτέλεσης διαμορφώνεται στα 1.40 δευτερόλεπτα, δηλαδή κατά 0.15 δευτερόλεπτα πιο αργός σε σύγκριση με την απλή εκδοχή του Branch and Bound. Από την ανάλυση των αποτελεσμάτων προκύπτει ότι στο 43.30% των περιπτώσεων, η χρήση του άνω ορίου συμβάλλει σε ταχύτερη επίλυση. Η διαφορά αυτή φανερώνει μια μερική βελτίωση στο χρόνο εκτέλεσης μέσω της χρήσης άνω ορίων από τον Simulated Annealing, με τη θετική επίδραση να είναι πιο αισθητή σε επιλεγμένες περιπτώσεις.

Όσον αφορά την υλοποίηση του αλγορίθμου Branch and Bound με αξιοποίηση της λύσης του μεθευρετικού VNS ως άνω όριο, ο γεωμετρικός μέσος χρόνος εκτέλεσης ανέρχεται σε 0.74 δευτερόλεπτα, δηλαδή είναι ταχύτερος κατά 0.51 δευτερόλεπτα σε σύγκριση με την απλούστερη εκδοχή του αλγορίθμου. Σε ποσοστό 77.50% των περιπτώσεων, η ενσωμάτωση του άνω ορίου συνέβαλε σε ταχύτερη επίλυση, γεγονός που υποδηλώνει μια περιορισμένη, αλλά υπαρκτή, συνεισφορά στη μείωση του χρόνου εκτέλεσης. Πρέπει, ωστόσο, να επισημανθεί ότι σε τέσσερις κλάσεις προβλημάτων δεν κατέστη δυνατή η εύρεση εφικτής λύσης από τους μεθευρετικούς αλγορίθμους, γεγονός που οδήγησε στην εκτέλεση του Branch and Bound χωρίς αρχικοποίηση στο άνω όριο. Το γεγονός αυτό ενδέχεται να έχει επηρεάσει την α-

ποτίμηση της αποδοτικότητας υπέρ της παραλλαγής με warm start, υποδηλώνοντας ότι η διαφορά στους χρόνους εκτέλεσης δεν μπορεί να αποδοθεί αποκλειστικά στην επίδραση του άνω ορίου, αλλά πιθανόν και σε εξωγενείς υπολογιστικούς παράγοντες.

Επομένως, η αξιοποίηση των μεθευρετικών αλγορίθμων Simulated Annealing και VNS για την παραγωγή άνω ορίων στον Branch and Bound επιφέρει οριακές μόνο βελτιώσεις στο χρόνο επίλυσης. Παρά το γεγονός ότι σε ορισμένες περιπτώσεις επιταχύνουν τη διαδικασία, το συνολικό όφελος κρίνεται αμελητέο. Αυτό οφείλεται κυρίως στο ότι απαιτείται επιπλέον υπολογιστικός χρόνος τόσο για την εύρεση μιας αρχικής εφικτής λύσης που θα χρησιμοποιηθεί ως είσοδος στους μεθευρετικούς αλγόριθμους, καθώς και για την ίδια την εκτέλεσή τους. Συνεπώς, η προστιθέμενη πολυπλοκότητα δεν αντισταθμίζεται επαρκώς από τη μικρή μείωση του χρόνου εκτέλεσης σε ορισμένες περιπτώσεις και η προσέγγιση αυτή δεν θεωρείται ιδιαίτερα αποδοτική για το συγκεκριμένο πρόβλημα.

Αναφορικά με την απόδοση του γενετικού αλγορίθμου GAFacilityOpt, παρατηρούνται σημαντικά μεγαλύτεροι χρόνοι εκτέλεσης σε σύγκριση με τον ακριβή λύτη Gurobi (0.32 και 437.74 δευτερόλεπτα αντίστοιχα), γεγονός που περιορίζει την πρακτική του χρήση σε προβλήματα μεγάλης κλίμακας. Ωστόσο, οι λύσεις που παράγει διατηρούνται σε αποδεκτά επίπεδα ακρίβειας, όπως καταδεικνύουν τα σχετικά σφάλματα (GAP %) για τις περισσότερες κλάσεις προβλημάτων. Συγκεκριμένα, σε αρκετές περιπτώσεις το GAP είναι χαμηλό, υποδεικνύοντας ότι ο γενετικός αλγόριθμος μπορεί να προσεγγίσει τη βέλτιστη λύση με ικανοποιητική ποιότητα (γεωμετρικός μέσος όρος GAP: 11.83%). Από τα δεδομένα προκύπτει επίσης ότι η απόδοση του αλγορίθμου διαφοροποιείται ανάλογα με τα χαρακτηριστικά και το μέγεθος του προβλήματος, με τα πιο σύνθετα και μεγάλης κλίμακας προβλήματα να οδηγούν σε υψηλότερα σφάλματα και αυξημένους χρόνους εκτέλεσης. Σημαντικό πλεονέκτημα του αλγορίθμου αποτελεί η δυνατότητα εξερεύνησης ευρύτερου χώρου αναζήτησης μέσω της αποδοχής, κατά την αρχικοποίηση, μη απολύτως εφικτών λύσεων, οι οποίες δύνανται να μετασχηματιστούν σε εφικτές μέσω των γενετικών τελεστών. Συνολικά, ο GAFacilityOpt αποτελεί μία εναλλακτική προσέγγιση με ικανοποιητική ποιότητα λύσεων, ωστόσο ο υψηλός υπολογιστικός χρόνος συνιστά ουσιαστικό περιορισμό για την ευρεία εφαρμογή του.

Τέλος, όσον αφορά τις στρατηγικές δημιουργίας αρχικών λύσεων, παρατηρήθηκε ότι η επιλογή της κατάλληλης μεθόδου εξαρτάται από τα χαρακτηριστικά του εκάστοτε αλγορίθμου. Η μέθοδος Initialize Population αποδείχθηκε ιδιαίτερα αποτελεσματική για το γενετικό αλγόριθμο, καθώς παρέχει έναν αρχικό πληθυσμό από ποιοτικές λύσεις, που είτε πληρούν τους περιορισμούς είτε παρουσιάζουν μικρές παραβιάσεις αντιμετωπίσιμες μέσω ποινών. Αυτό επιτρέπει στον αλγόριθμο να διατηρεί υψηλή ποιότητα λύσεων, ενώ ταυτόχρονα διερευνά ευρύτερα το χώρο αναζήτησης, ενισχύοντας την πιθανότητα αποφυγής τοπικών βελτίστων. Αντίθετα, στους μεθευρετικούς αλγορίθμους Simulated Annealing και Variable Neighborhood Search, η μέθοδος Randomized Constructive αποδείχθηκε πιο κατάλληλη, καθώς δίνει έμφαση στην ποικιλία και ευελιξία των αρχικών λύσεων, στοιχείο κρίσιμο για την αποδοτική εξερεύνηση του χώρου λύσεων και τη σταδιακή βελτίωσή τους. Η δοκιμαστική εφαρμογή της μεθόδου Initialize Population στους μεθευρετικούς αλγορίθμους δεν οδήγησε σε ουσιαστική βελτίωση των αποτελεσμάτων, γεγονός που αποδίδεται στη συγκριτικά αυστηρότερη δομή των λύσεων που αυτή παράγει.

Η εργασία αυτή ανέδειξε τη σημασία της αποδοτικής χωροθέτησης σταθμών φόρτισης υπό ρεαλιστικούς περιορισμούς, εφαρμόζοντας και συγκρίνοντας 13 αλγορίθμους που κυμαίνονται από ακριβείς εως μεθευρετικές τεχνικές. Η παραλλαγή του Branch & Bound με BestFS και ευρετική Min Distance αποδείχθηκε η πιο σταθερή και αποτελεσματική, ενώ η χρήση warm start από μεθευρετικούς αλγορίθμους παρείχε περιορισμένα οφέλη σε σχέση με το επιπλέον υπολογιστικό κόστος. Τα αποτελέσματα δείχνουν ότι, για τα εξεταζόμενα σενάρια, μια καλά ρυθμισμένη ακριβής μέθοδος μπορεί να υπερέχει σε ποιότητα και σταθερότητα έναντι πιο σύνθετων προσεγγίσεων. Μελλοντικές μελέτες μπορούν να εξετάσουν δυναμικά ή στοχαστικά περιβάλλοντα, εναλλακτικές μορφές κόστους και συμπεριφοράς χρηστών ή την ενσωμάτωση μηχανικής μάθησης για την καθοδήγηση της διαδικασίας βελτιστοποίησης.

Βιβλιογραφία

[1] M. Kchaou-Boujelben and C. Gicquel. Efficient solution approaches for locating electric vehicle fast charging stations under driving range uncertainty. *Computers Operations Research*, 109:288–299, 2019.

[2] Mouna Kchaou-Boujelben. Charging station location problem: A comprehensive review on models and solution approaches. *Transportation Research Part C: Emerging Technologies*, 132:103376, 2021. Available online 8 November 2021.

[3] Susan Hesse Owen and Mark S. Daskin. Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423–447, 1998.

[4] Mark S. Daskin and Kayse Lee Maass. The p-median problem. In Gilbert Laporte, Stefan Nickel, and François Saldanha da Gama, editors, *Location Science*, chapter 2, pages 21–45. Springer, 2015.

[5] O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems. ii: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[6] Mark S. Daskin and Kayse Lee Maass. *The p-Median Problem*, pages 21–45. Springer International Publishing, Cham, 2015.

[7] Simon J. Blanchard, Daniel Aloise, and Wayne S. DeSarbo. The heterogeneous p-median problem for categorization based clustering. *Psychometrika*, 77(4):741–762, October 2012.

[8] Fareed Ahmad, Atif Iqbal, Imtiaz Ashraf, Mousa Marzband, and Irfan Khan. Optimal location of electric vehicle charging station and its impact on distribution network: A review. *Energy Reports*, 8:2314–2333, 2022. Review article.

[9] A. Shukla, K. Verma, and R. Kumar. Multi-objective synergistic planning of ev fast-charging stations in the distribution system coupled with the transportation network. *IET Generation, Transmission & Distribution*, 13:3421–3432, 2019.

[10] M. Z. Zeb, K. Imran, A. Khattak, A. K. Janjua, A. Pal, M. Nadeem, J. Zhang, and S. Khan. Optimal placement of electric vehicle charging stations in the active distribution network. *IEEE Access*, 8:68124–68134, 2020.

[11] L. Chen, C. Xu, H. Song, and K. Jermsittiparsert. Optimal sizing and sitting of evcs in the distribution system using metaheuristics: A case study. *Energy Reports*, 7:208–217, 2021.

[12] A. Pal, A. Bhattacharya, and A. K. Chakraborty. Allocation of electric vehicle charging station considering uncertainties. *Sustainable Energy Grids and Networks*, 25:100422, 2021.

[13] B. Faridpak, H. F. Gharibeh, M. Farrokhifar, and D. Pozo. Two-step lp approach for optimal placement and operation of ev charging stations. In *Proceedings of 2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe 2019)*, pages 1–5, 2019.

[14] M. Moradijoz, F. Moazzen, S. Allahmoradi, M. P. Moghaddam, and M. R. Haghifam. A two-stage model for optimum allocation of electric vehicle parking lots in smart grids. In *Proceedings - 2018 Smart Grid Conference (SGC 2018)*, 2018.

[15] Z. Tian, W. Hou, X. Gu, F. Gu, and B. Yao. The location optimization of electric vehicle charging stations considering charging behavior. *Simulation*, 94(7):625–636, 2018.

[16] T. Yi, X.-b. Cheng, H. Zheng, and J.-p. Liu. Research on location and capacity optimization method for electric vehicle charging stations considering user's comprehensive satisfaction. *Energies*, 2019.

[17] H. Zhang, Z. Hu, Z. Xu, and Y. Song. An integrated planning framework for different types of pev charging facilities in urban area. *IEEE Transactions on Smart Grid*, 7(5):2273–2284, 2016.

[18] Edward A. Silver, R. Victor, V. Vidal, and Dominique de Werra. A tutorial on heuristic methods. *European Journal of Operational Research*, 5(3):153–162, 1980.

[19] O. Arslan, O.E. Karasan, A.R. Mahjoub, and H. Yaman. A branch-and-cut algorithm for the alternative fuel refueling station location problem with routing. *Transportation Science*, 53(4):1107–1125, 2019.

[20] M. Xu and Q. Meng. Optimal deployment of charging stations considering path deviation and nonlinear elastic demand. *Transportation Research Part B: Methodological*, 135:120–142, 2020.

[21] B. Yildiz, O. Arslan, and O.E. Karasan. A branch and price approach for routing and refueling station location model. *European Journal of Operational Research*, 248(3):815–826, 2016.

[22] Okan Arslan and Oya Ekin Karaşan. A benders decomposition approach for the charging station location problem with plug-in hybrid electric vehicles. *Transportation Research Part B: Methodological*, 93:670–695, 2016.

[23] Sunith Bandaru and Kalyanmoy Deb. Metaheuristic techniques. *Decision Sciences*, pages 693–750, 2016.

[24] O. Berman, R.C. Larson, and N. Fouska. Optimal locations of discretionary service facilities. *Transportation Science*, 26(3):201–211, 1992.

[25] M.J. Hodgson, K.E. Rosing, and A.L.G. Storrier. Applying the flow-capturing location-allocation model to an authentic network: Edmonton, canada. *European Journal of Operational Research*, 90(3):427–443, 1996.

[26] S. Lim and M. Kuby. Heuristic algorithms for siting alternative-fuel stations using the flow-refueling location model. *European Journal of Operational Research*, 204(1):51–61, 2010.

[27] Fred Glover. Exploiting local optimality in metaheuristic search. *arXiv preprint arXiv:2010.05394*, 2020.

[28] J. Dong, C. Liu, and Z. Lin. Charging infrastructure planning for promoting battery electric vehicles: An activity-based approach using multiday travel data. *Transportation Research Part C: Emerging Technologies*, 38:44–55, 2014.

[29] F. He, Y. Yin, and J. Zhou. Deploying public charging stations for electric vehicles on urban road networks. *Transportation Research Part C: Emerging Technologies*, 60:227–240, 2015.

[30] A.A. Kadri, R. Perrouault, M. Kchaou-Boujelben, and C. Gicquel. A multi-stage stochastic integer programming approach for locating electric vehicle charging stations. *Computers Operations Research*, 117:104888, 2020.

[31] N. Kang, F.M. Feinberg, and P.Y. Papalambros. Integrated decision making in electric vehicle and charging station location network design. *Journal of Mechanical Design*, 137(10):061402, 2015.

[32] M. Ghamami, A. Zockaie, and Y.M. Nie. A general corridor model for designing plug-in electric vehicle charging infrastructure to support intercity travel. *Transportation Research Part C: Emerging Technologies*, 68:389–402, 2016.

[33] Y.G. Lee, H.S. Kim, S.Y. Kho, and C. Lee. User equilibrium–based location model of rapid charging stations for electric vehicles with batteries that have different states of charge. *Transportation Research Record: Journal of the Transportation Research Board*, 2454:97–106, 2014.

[34] F. Guo, J. Yang, and J. Lu. The battery charging station location problem: Impact of users' range anxiety and distance convenience. *Transportation Research Part A: Policy and Practice*, 114:1–18, 2018.

[35] M. Hosseini and S.A. MirHassani. A heuristic algorithm for optimal location of flow-refueling capacitated stations. *International Transactions in Operational Research*, 24(6):1377–1403, 2017.

[36] Mouna Kchaou-Boujelben. Charging station location problem: A comprehensive review on models and solution approaches. *Transportation Research Part C: Emerging Technologies*, 132:103376, 2021. §3.1.1, Relevant discussion on queueing theory models can be found on pages 7 and 12 (PDF pagination), Relevant discussion on Benders decomposition can be found on pages 11 and 17 (PDF pagination).

[37] Claude Michel and Michel Rueher. Handling software upgradeability problems with milp solvers. In *Proceedings of the First International Workshop on Logics for Component Configuration (LoCoCo 2010)*, volume 29, pages 1–10. EPTCS, 2010.

[38] Rimmi Anand, Divya Aggarwal, and Vijay Kumar. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, 20(4):623–635, 2017.

[39] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: Modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[40] E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[41] M.A. Efroymson and T.L. Ray. A branch-bound algorithm for plant location. *Operations Research*, 14(3):361–368, 1966.

[42] Antonia Chmiela, Elias B. Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. In *Advances in Neural Information Processing Systems (NeurIPS) 34*. Curran Associates, Inc., 2021. `https://arxiv.org/abs/2103.10294`.

[43] Laurence A. Wolsey. Heuristic analysis, linear programming and branch and bound. In V. J. Rayward-Smith, editor, *Combinatorial Optimization II*, pages 121–134. Springer, 1980.

[44] Alfred A. Kuehn and Michael J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

[45] Michael D. Vose. Random heuristic search. *Theoretical Computer Science*, 229(1-2):103–142, 1999.

[46] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[47] Laetitia Jourdan, Michel Basseur, and El-Ghazali Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.

[48] Marcos Vasquez et al. Hybridizations of metaheuristics with branch & bound derivatives. In Christian Blum, Maria José Blesa Aguilera, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 85–116. Springer, 2008.

[49] Danish Shah et al. Hybrid meta-heuristic algorithms for optimal sizing of hybrid renewable energy system: A review of the state-of-the-art. *Archives of Computational Methods in Engineering*, 29:4049–4083, 2022.

[50] Kumar Gdeepak and Ajay Kumar. Facility location problem using genetic algorithm: A review. *International Journal of Engineering Research and Applications*, 2(4):1924–1928, 2012.

[51] Hela Bouziri and Zied Jemai. A genetic algorithm for solving a capacitated p-median problem. In *Metaheuristics for Hard Optimization: Methods and Case Studies*, pages 153–168. Springer, 2008.

[52] R. Kumar and A. Shinde. Implementing genetic algorithm to solve facility location problem. *International Research Journal of Engineering and Technology (IRJET)*, 2(5):1013–1018, 2015.

[53] Eslam Mohamed, Ahmad Izani Bakar, and Sharifah Sakinah Yahaya. A hybrid genetic algorithm for solving facility location-allocation problem. In *Proceedings of the International Conference on Industrial Engineering and Operations Management (IEOM)*, Kuala Lumpur, Malaysia, 2016.