Πολυτεχνείο Κρήτης
Τμήμα Μηχανικών Παραγωγής και Διοίκησης

Μεθευρετικοί και υβριδικοί αλγόριθμοι για προβλήματα δρομολόγησης οχημάτων πολλαπλών αντικειμενικών συναρτήσεων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γκούβερης Κωνσταντίνος (ΑΜ: 2020010154)

Επιβλέπων: Ιωάννης Μαρινάκης, Καθηγητής

Εξεταστική επιτροπή:

Νικόλαος Ματσατσίνης, Ομότιμος Καθηγητής

Μαγδαληνή Μαρινάκη, Μέλος ΕΔΙΠ

Χανιά, Νοέμβριος 2024

Πρόλογος

Η παρούσα διπλωματική εργασία με τίτλο «Μεθευρετικοί και υβριδικοί αλγόριθμοι για προβλήματα δρομολόγησης οχημάτων πολλαπλών αντικειμενικών συναρτήσεων», εκπονήθηκε στο πλαίσιο του προπτυχιακού προγράμματος σπουδών της Σχολής Μηχανικών Παραγωγής και Διοίκησης του Πολυτεχνείου Κρήτης.

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή του Π.Κ. κ. Ιωάννη Μαρινάκη, για την ανάθεση και επίβλεψη της παρούσας εργασίας καθώς και για την πολύτιμη βοήθειά του κατά την συγγραφή της.

Τέλος, δε θα μπορούσα να μην ευχαριστήσω την οικογένεια μου, τους συναδέλφους αλλά και φίλους μου, που με στήριξαν καθ’ όλη τη διάρκεια των προπτυχιακών μου σπουδών.

Περίληψη

Στον τομέα της εφοδιαστικής (logistics) και της διαχείρισης μεταφορών (transportation management), η βελτιστοποίηση της δρομολόγησης των οχημάτων διαδραματίζει κεντρικό ρόλο στη διασφάλιση της αποτελεσματικής χρήσης των διαθέσιμων πόρων, στην ελαχιστοποίηση του λειτουργικού κόστους και στην ικανοποίηση των απαιτήσεων των πελατών. Καθώς οι αλυσίδες εφοδιασμού συνεχίζουν να αυξάνονται σε πολυπλοκότητα και σε κλίμακα, οι προκλήσεις που σχετίζονται με τα προβλήματα δρομολόγησης οχημάτων (vehicle routing problems - VRP) γίνονται ολοένα και δυσκολότερα διαχειρίσιμες.

Στο σύγχρονο εργασιακό περιβάλλον των μεγάλων επιχειρήσεων, η πλειονότητα των υλικοτεχνικών προκλήσεων εκδηλώνονται ως προβλήματα πολλαπλών στόχων. Πιο συγκεκριμένα, τα προβλήματα δρομολόγησης οχημάτων πολλαπλών αντικειμενικών συναρτήσεων (multi-objective VRP) περιλαμβάνουν πάνω από έναν, αντικρουόμενους στην πλειοψηφία των περιπτώσεων στόχους, όπως η ελαχιστοποίηση του κόστους μεταφοράς, η μείωση του περιβαλλοντικού αποτυπώματος, η μείωση του χρόνου παράδοσης των προϊόντων αλλά και η βέλτιστη αξιοποίηση του προσωπικού μεταφορών (επαρκής ανάπαυση, αποφυγή της υπερεργασίας κ.α.).

Η διαχείριση της εφοδιαστικής αλυσίδας και η εφοδιαστική, ως αναπόσπαστα στοιχεία των σύγχρονων επιχειρηματικών λειτουργιών, αναζητούν συνεχώς καινοτόμες λύσεις για την αντιμετώπιση αυτών των προκλήσεων. Οι τεχνικές βελτιστοποίησης έχουν αναδειχθεί ως απαραίτητα εργαλεία σε αυτή την προσπάθεια, επιτρέποντας στους οργανισμούς να εκσυγχρονίσουν τις διαδικασίες εφοδιαστικής τους και να βελτιώσουν τη συνολική απόδοση της εφοδιαστικής αλυσίδας.

Η παρούσα διπλωματική εργασία επιχειρεί να συμβάλει στην κατανόηση και στην πρακτική εφαρμογή των αλγορίθμων βελτιστοποίησης στον τομέα της επιχειρησιακής έρευνας και εφοδιαστικής. Με τη εφαρμογή, αξιολόγηση και σύγκριση των αποτελεσμάτων μιας ποικιλίας ευρετικών και μεθευρετικών αλγορίθμων, αλλά και προτεινόμενων υβριδικών εκδόσεών τους, θα προκύψουν χρήσιμες πληροφορίες για την αποτελεσματικότητά αυτών στην αντιμετώπιση πολύπλοκων προβλημάτων δρομολόγησης.

Αρχικά γίνεται αναλυτική παρουσίαση πλήθους μεθευρετικών αλγορίθμων (tabu search, simulated annealing, iterated local search, ant colony optimization κ.α.) που εφαρμόζονται

συνήθως κατά τη βελτιστοποίηση μίας μοναδικής αντικειμενικής συνάρτησης. Στη συνέχεια παρουσιάζονται αλγόριθμοι βελτιστοποίησης πολλαπλών αντικειμενικών συναρτήσεων (MOTS, NSGA-II, SPEA-II) και έπειτα προτείνονται προσαρμοσμένες υβριδικές προσεγγίσεις που συνδυάζουν δύο τουλάχιστον από τις παραπάνω τεχνικές με σκοπό την επίτευξη καλύτερης ποιότητας λύσεων ή/και την ταχύτερη σύγκλιση στο βέλτιστο μέτωπο Pareto. Οι αλγόριθμοι αυτοί εφαρμόζονται για την επίλυση δύο διαφορετικών προβλημάτων δρομολόγησης οχημάτων για την εξυπηρέτηση 200 πελατών, βελτιστοποιώντας ταυτόχρονα πολλαπλές αντικειμενικές συναρτήσεις. Τέλος, τα αποτελέσματα αυτών των εφαρμογών συγκρίνονται με τα αποτελέσματα που θα εξήγαγε η εφαρμογή ενός μόνο μεθευρετικού αλγορίθμου.

Abstract

In the field of logistics and transportation management, the optimization of vehicle routing plays a central role in ensuring the efficient use of available resources, minimizing operational costs and meeting customer requirements. As supply chains continue to grow in complexity and scale, the challenges associated with vehicle routing problems (VRP) become increasingly difficult to manage.

In the modern work environment of large enterprises, the majority of logistical challenges manifest themselves as multi-objective problems. More specifically, multi-objective vehicle routing problems typically involve multiple conflicting objectives, such as minimizing transportation costs, reducing the environmental footprint, reducing product delivery time, and the optimal utilization of transport staff (adequate rest, avoidance of overtime work, etc.)

Supply chain management and logistics, as integral elements of modern business operations, are constantly looking for innovative solutions to address these challenges. Optimization techniques have emerged as essential tools in this effort, enabling organizations to modernize their logistics processes and improve the overall supply chain performance.

This thesis attempts to contribute to the understanding and practical application of optimization algorithms in the field of operational research and logistics. By applying, evaluating and comparing the results of a variety of heuristic and meta-heuristic algorithms, as well as their hybridized versions, useful information will emerge on their effectiveness in dealing with complex routing problems.

Firstly, several metaheuristic algorithms, such as tabu search, simulated annealing, iterated local search, and ant colony optimization, typically used for solving single-objective optimization problems, are introduced. Subsequently, multi-objective optimization algorithms, including MOTS, NSGA-II, and SPEA-II, are discussed. Lastly, customized hybrid approaches that combine at least two of these techniques are proposed to achieve better solutions and/or faster convergence to the optimal Pareto frontier. These hybrid algorithms are applied to address two different multi-objective vehicle routing problems involving 200 customers. The results of these applications are then compared to those produced by the utilization of a single metaheuristic algorithm.

Περιεχόμενα

Κεφάλαιο 1. Εισαγωγή

1.1. Η εφοδιαστική αλυσίδα

Στη σημερινή παγκόσμια οικονομία, η στρατηγική διαχείριση των αλυσίδων εφοδιασμού αποτελεί βασικό στοιχείο για την απόκτηση ανταγωνιστικού πλεονεκτήματος. Η διοίκηση της εφοδιαστικής αλυσίδας (supply chain management) δεν αφορά μόνο την εφοδιαστική (logistics), αλλά περιλαμβάνει ένα σύνθετο δίκτυο διασυνδεδεμένων οργανισμών που συνεργάζονται για την παράδοση αγαθών και υπηρεσιών στους τελικούς καταναλωτές. Όπως διατυπώθηκε από τον Harland (1996), ο όρος SCM περιλαμβάνει την διαχείριση της ροής των υλικών, από το σημείο προέλευσης τους μέχρι την κατανάλωση, διασχίζοντας διάφορα στάδια όπως η παραγωγή, η αποθήκευση, η μεταφορά και η διανομή αυτών. Τελικός σκοπός είναι η ικανοποίηση των απαιτήσεων των πελατών και για την επίτευξη αυτού εμπλέκονται προμηθευτές, κατασκευαστές, χώροι αποθήκευσης, αποθέματα, έτοιμα προϊόντα, κέντρα διανομής, μεταφορείς, πωλητές και πελάτες.

Στον πυρήνα της, η εφοδιαστική αλυσίδα αντιμετωπίζει ένα πλήθος προκλήσεων, καθεμία από τις οποίες παρουσιάζει προβλήματα που απαιτούν επίλυση για την αποφυγή εμποδίων και διακοπών στις επιχειρηματικές λειτουργίες. Αυτά περιλαμβάνουν τη διαμόρφωση των δικτύων διανομής (αριθμός και θέση των εγκαταστάσεων παραγωγής, των χώρων αποθήκευσης και των κέντρων διανομής), τις στρατηγικές διανομής (αποφάσεις σχετικά με τον τρόπο διαχείρισης και ελέγχου των λειτουργιών), την εναρμόνιση μεμονωμένων δραστηριοτήτων εντός της αλυσίδας και την συμβατότητα αυτών, τη διευκόλυνση της ροής πληροφοριών (στοιχεία ζήτησης, προβλέψεις, διαθέσιμο απόθεμα, παρακολούθηση μεταφορών κ.α.), την διαχείριση των αποθεμάτων και τη βελτιστοποίηση των χρηματοοικονομικών συναλλαγών.

Εικόνα 1.1. *Η διαχείριση της εφοδιαστικής αλυσίδας*

Η διαμόρφωση των δικτύων διανομής παρουσιάζει μια ποικιλία επιμέρους υπο-προβλημάτων που χωρίζονται σε τρεις κύριες κατηγορίες: προβλήματα δρομολόγησης (routing problems), χρονοπρογραμματισμού (scheduling problems) και χωροθέτησης (facility location problems). Κάθε κατηγορία περιλαμβάνει ένα φάσμα προκλήσεων, από την αυστηρά χρονοεξαρτώμενη δρομολόγηση έως τη διαχείριση της ζήτησης δυναμικά και σε πραγματικό χρόνο, απαιτώντας διαφοροποιημένες λύσεις για την επίτευξη των τελικών στόχων της βελτιστοποίησης.

Η εργασία αυτή εμβαθύνει στην πρώτη κατηγορία προβλημάτων, εστιάζοντας στη δρομολόγηση οχημάτων. Θα επικεντρωθεί στην πολυπλοκότητα των προβλημάτων δρομολόγησης και των διαφόρων παραλλαγών αυτών, με μία ή και περισσότερες αντικειμενικές συναρτήσεις, διερευνώντας διάφορες αλγοριθμικές προσεγγίσεις. Ενσωματώνοντας στην ανάλυση πλήθος μεθοδολογιών βελτιστοποίησης θα παράσχει μια ολοκληρωμένη κατανόηση του κάθε προβλήματος και της ποιότητας των λύσεών του.

1.2. Εφοδιαστική

Η εφοδιαστική, ως έννοια, βρίσκει τις ρίζες της στους αρχαίους πολιτισμούς όπου η αποτελεσματική μετακίνηση και η παροχή αγαθών ήταν ζωτικής σημασίας για στρατιωτικές εκστρατείες, εμπορικές αποστολές και κοινωνική ανάπτυξη. Ο όρος «logistics» προέρχεται από την αρχαία Ελληνική λέξη «λογιστικός», που σημαίνει «ικανός ή «κατάλληλος να κάνει υπολογισμούς». Οι αρχαίοι στρατιωτικοί ηγέτες, όπως ο Μέγας Αλέξανδρος και ο Ιούλιος Καίσαρας, βασίζονταν σε μεγάλο βαθμό στον υλικοτεχνικό σχεδιασμό για να συντηρήσουν τους στρατούς τους κατά τη διάρκεια των κατακτήσεων.

Η σύγχρονη κατανόηση των logistics άρχισε να διαμορφώνεται κατά τη διάρκεια της Βιομηχανικής Επανάστασης με την άνοδο της μαζικής παραγωγής και την ανάγκη για αποτελεσματικά συστήματα μεταφορών. Η εμφάνιση των σιδηροδρόμων και των ατμόπλοιων έφερε επανάσταση στη διακίνηση εμπορευμάτων σε μεγάλες αποστάσεις, προκαλώντας την επισημοποίηση των πρακτικών εφοδιαστικής.

Η εφοδιαστική έπαιξε καθοριστικό ρόλο και στους δύο παγκόσμιους πολέμους, τονίζοντας τη στρατηγική της σημασία. Η κλίμακα των στρατιωτικών επιχειρήσεων απαιτούσε πολύπλοκες υλικοτεχνικές επιχειρήσεις, συμπεριλαμβανομένων των προμηθειών, της μεταφοράς και της διανομής πόρων στην πρώτη γραμμή. Καινοτομίες στα logistics, όπως το μοντέλο παραγωγής γραμμής συναρμολόγησης που εισήγαγε ο Henry Ford, μεταμόρφωσαν περαιτέρω τις βιομηχανικές αλυσίδες εφοδιασμού.

Η μετά τον Β' Παγκόσμιο Πόλεμο εποχή γνώρισε σημαντικές προόδους στον τομέα της εφοδιαστικής με γνώμονα τις τεχνολογικές καινοτομίες και την παγκοσμιοποίηση. Η εμφάνιση της μεταφοράς εμπορευματοκιβωτίων (intermodal shipping containers), όπου πρωτοστάτησε ο επιχειρηματίας Malcom McLean τη δεκαετία του 1950, έφερε επανάσταση στις θαλάσσιες μεταφορές, καθιστώντας τις ταχύτερες, πιο αξιόπιστες και οικονομικά αποδοτικές.

Το δεύτερο μισό του 20ου αιώνα είδε την άνοδο πολυεθνικών εταιρειών και ομίλων ετερογενών δραστηριοτήτων με παγκόσμιες αλυσίδες εφοδιασμού. Εταιρείες όπως η Walmart, η Amazon και η UPS ταυτίστηκαν με τις πλέον αποτελεσματικές λειτουργίες logistics, αξιοποιώντας την τεχνολογία για τη βελτιστοποίηση της διαχείρισης αποθεμάτων, της αποθήκευσης και των δικτύων διανομής.

Η έλευση της τεχνολογίας πληροφοριών στα τέλη του 20ου αιώνα μεταμόρφωσε τις πρακτικές εφοδιαστικής, επιτρέποντας την παρακολούθηση σε πραγματικό χρόνο, τη βελτιστοποίηση διαδρομών και την πρόβλεψη της ζήτησης. Τεχνολογίες όπως το GPS, το RFID και το IoT έχουν βελτιώσει περαιτέρω την ορατότητα και τη διαφάνεια στις αλυσίδες εφοδιασμού, διευκολύνοντας τη λήψη αποφάσεων και την καλύτερη εξυπηρέτηση των πελατών.

Στον 21ο αιώνα, τα logistics συνεχίζουν να εξελίσσονται γρήγορα ως απάντηση στις μεταβαλλόμενες προσδοκίες των καταναλωτών, τις περιβαλλοντικές ανησυχίες και τις γεωπολιτικές αλλαγές. Τάσεις όπως το λιανικό εμπόριο πολλαπλών καναλιών (multi-channel retailing) και παντός καναλιού (omni-channel retailing), η παράδοση αυθημερόν και η βιωσιμότητα αναδιαμορφώνουν τις στρατηγικές logistics, ωθώντας τις εταιρείες να υιοθετήσουν ευέλικτες και φιλικές προς το περιβάλλον πρακτικές.

Κοιτάζοντας στο μέλλον, η εφοδιαστική είναι έτοιμη να υποστεί περαιτέρω μετασχηματισμό λόγω των εξελίξεων στον αυτοματισμό, την τεχνητή νοημοσύνη και τα αυτόνομα οχήματα. Έννοιες όπως η παράδοση με τη χρήση drones, οι αυτόνομες ρομποτικές αποθήκες και η διαχείριση της εφοδιαστικής αλυσίδας που βασίζεται σε τεχνολογίες blockchain αναμένεται να επαναπροσδιορίσουν το τοπίο της βιομηχανίας, εγκαινιάζοντας μια νέα εποχή αποτελεσματικότητας και καινοτομίας.

1.3. Βασικές έννοιες στη διαχείριση της εφοδιαστικής αλυσίδας

Η εφοδιαστική διανομής (distribution logistics) περιλαμβάνει ένα σύνολο διαδικασιών και δραστηριοτήτων που στοχεύουν στη διαχείριση της μετακίνησης, αποθήκευσης και διακίνησης αγαθών εντός του δικτύου της εφοδιαστικής αλυσίδας. Περιλαμβάνει τη λήψη στρατηγικών αποφάσεων σχετικά με το σχεδιασμό και τη λειτουργία των καναλιών διανομής, των αποθηκών και των συστημάτων μεταφοράς για τη διασφάλιση της έγκαιρης παράδοσης των προϊόντων με παράλληλη βελτιστοποίηση των επιπέδων αποθεμάτων και ελαχιστοποίηση του λειτουργικού κόστους.

Η εισερχόμενη εφοδιαστική (inbound logistics) αναφέρεται στις διαδικασίες που εμπλέκονται στην προμήθεια, παραλαβή και αποθήκευση πρώτων υλών, εξαρτημάτων και αγαθών από προμηθευτές. Περιλαμβάνει δραστηριότητες όπως η προμήθεια, η μεταφορά, η αποθήκευση και η διαχείριση αποθεμάτων, όλα με στόχο τη διασφάλιση ομαλής ροής υλικών στις εγκαταστάσεις εργασιών παραγωγής και διανομής. Οι αποτελεσματικές στρατηγικές εισερχόμενης εφοδιαστικής είναι απαραίτητες για την ελαχιστοποίηση των χρόνων παράδοσης, τη μείωση του κόστους διατήρησης αποθεμάτων και τη διατήρηση υψηλών επιπέδων διαθεσιμότητας προϊόντων για την υποστήριξη των παραπάνω εργασιών.

Από την άλλη πλευρά, η εξερχόμενη εφοδιαστική (outbound logistics) επικεντρώνεται στη διανομή τελικών προϊόντων σε πελάτες ή τελικούς χρήστες. Περιλαμβάνει δραστηριότητες όπως η επεξεργασία παραγγελιών, η παραλαβή, η συσκευασία και η αποστολή, καθώς και η διαχείριση των δικτύων μεταφοράς και των καναλιών παράδοσης. Στόχος είναι η εξυπηρέτηση των παραγγελιών των πελατών έγκαιρα και με ακρίβεια, βελτιστοποιώντας παράλληλα τις διαδρομές παράδοσης, ελαχιστοποιώντας το κόστος μεταφοράς και μεγιστοποιώντας την ικανοποίηση των πελατών. Η αποτελεσματική εξερχόμενη εφοδιαστική είναι απαραίτητη για τη διατήρηση του ανταγωνιστικού πλεονεκτήματος στην αγορά και τη δημιουργία ισχυρών σχέσεων με τους πελάτες.

Τα συστήματα παράδοσης (delivery systems) αφορούν το τελικό στάδιο της αλυσίδας εφοδιασμού, διασφαλίζοντας ότι τα προϊόντα φτάνουν στον προορισμό τους έγκαιρα και αξιόπιστα. Ένα σύστημα παράδοσης περιλαμβάνει την υποδομή, τις διαδικασίες και τις τεχνολογίες που χρησιμοποιούνται για τη διευκόλυνση της μετακίνησης και παράδοσης αγαθών από τα κέντρα διανομής ή τις αποθήκες στις πόρτες των πελατών. Περιλαμβάνει διάφορους τρόπους μεταφοράς, όπως φορτηγά, πλοία, τρένα και αεροπλάνα, καθώς και

λύσεις παράδοσης τελευταίου μιλίου (last mile delivery), όπως υπηρεσίες ταχυμεταφορών, θυρίδες δεμάτων και αυτόνομα drones για παραδόσεις.

Εκτός από τα logistics διανομής, τα εισερχόμενα και εξερχόμενα logistics και τα συστήματα παράδοσης, άλλες χρήσιμες έννοιες περιλαμβάνουν τη διαχείριση αποθεμάτων (inventory management), την πρόβλεψη ζήτησης (demand forecasting), την εκπλήρωση παραγγελιών (order fulfillment) και την αντίστροφη εφοδιαστική (reverse logistics). Η διαχείριση αποθεμάτων περιλαμβάνει τη διατήρηση των βέλτιστων επιπέδων αποθέματος για την κάλυψη της ζήτησης των πελατών, ελαχιστοποιώντας ταυτόχρονα το κόστος αποθήκευσής του. Η πρόβλεψη ζήτησης βοηθά τις επιχειρήσεις να εκτιμήσουν τη μελλοντική ζήτηση για τα προϊόντα τους και να σχεδιάσουν ανάλογα τις δραστηριότητες παραγωγής και διανομής. Η εκπλήρωση παραγγελιών περιλαμβάνει ολόκληρη τη διαδικασία λήψης, επεξεργασίας και παράδοσης παραγγελιών των πελατών, από την τοποθέτηση της παραγγελίας έως την παράδοσή της. Η αντίστροφη εφοδιαστική περιλαμβάνει τη διαχείριση της επιστροφής και απόρριψης προϊόντων, την ανακύκλωση υλικών και τον χειρισμό των ανακλήσεων ή επισκευών προϊόντων, διασφαλίζοντας τη βιωσιμότητα και ελαχιστοποιώντας τα απόβλητα από την αλυσίδα εφοδιασμού. Τέλος, η κατανόηση αυτών των εννοιών καθίσταται απαραίτητη για το σχεδιασμό και τη βελτιστοποίηση των λειτουργιών της εφοδιαστικής.

Κεφάλαιο 2. Δρομολόγηση Οχημάτων

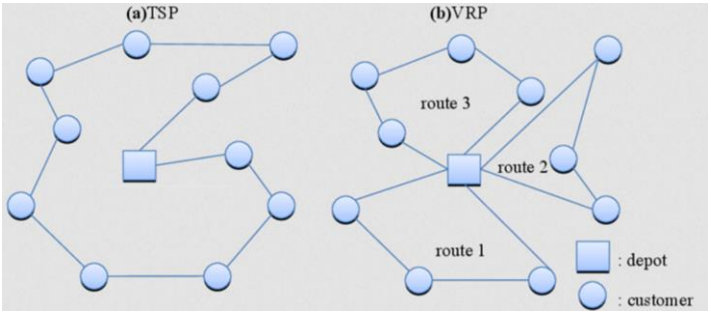
2.1. Πρόβλημα δρομολόγησης οχημάτων (ΠΔΟ)

Το ΠΔΟ αποτελεί ένα ευρέως διαδεδομένο πρόβλημα συνδυαστικής βελτιστοποίησης στον τομέα της επιχειρησιακής έρευνας και της εφοδιαστικής. Κύριος στόχος του κλασικού ΠΔΟ είναι ο καθορισμός του βέλτιστου συνόλου διαδρομών για έναν στόλο οχημάτων που ξεκινάει από μία αποθήκη με προϊόντα για την εξυπηρέτηση ενός συνόλου πελατών, ελαχιστοποιώντας το κόστος των μεταφορών, με την κατά το δυνατόν μείωση των συνολικά διανυόμενων αποστάσεων ή/και χρόνων, ικανοποιώντας παράλληλα συγκεκριμένους περιορισμούς. Η βελτιστοποίηση αυτή δεν μεταφράζεται απλώς σε απτή εξοικονόμηση χρημάτων για επιχειρήσεις και οργανισμούς, αλλά συμβάλλει επίσης στην ικανοποίηση εργαζομένων και πελατών, στη επιβράδυνση της φθοράς του εξοπλισμού μεταφορών, στην περιβαλλοντική βιωσιμότητα με μείωση των εκπομπών άνθρακα και πολλά άλλα. Το ΠΔΟ βρίσκει εφαρμογή σε διάφορα σενάρια του πραγματικού κόσμου, όπως οι μεταφορές, οι

υπηρεσίες παράδοσης, η συλλογή απορριμμάτων, ακόμη και στρατιωτικές επιχειρήσεις ή και επιχειρήσεις αντιμετώπισης καταστάσεων έκτακτης ανάγκης, όπως πυρκαγιές, σεισμοί και πλημμύρες.

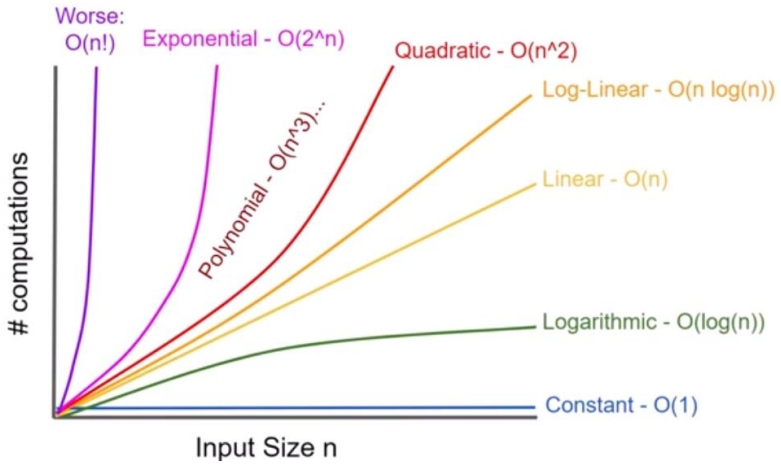
Το ΠΔΟ περιλαμβάνει πολλές παραλλαγές, κάθε μία προσαρμοσμένη για να αντιμετωπίζει συγκεκριμένους περιορισμούς και επιχειρησιακά σενάρια. Μία διακεκριμένη παραλλαγή αποτελεί το πρόβλημα δρομολόγησης οχημάτων με χρονικά παράθυρα (VRP with time windows), όπου οι πελάτες δεσμεύουν τους μεταφορείς με συγκεκριμένα χρονικά περιθώρια εντός των οποίων είναι διαθέσιμοι και απαιτείται να εξυπηρετηθούν. Μία ακόμη παραλλαγή αποτελεί το πρόβλημα δρομολόγησης οχημάτων με περιορισμούς χωρητικότητας και χρόνων μετάβασης (capacitated VRP) όπου λόγω της περιορισμένης χωρητικότητας και του χρόνου των οχημάτων απαιτείται αποτελεσματική κατανομή των διαθέσιμων πόρων ώστε να μην χρειαστούν περισσότερα οχήματα (και οδηγοί) από τα απαραίτητα. Επιπλέον, το πρόβλημα δρομολόγησης οχημάτων με πολλαπλές αποθήκες (multi-depot VRP) επεκτείνει τις εφαρμογές του ΠΔΟ περιλαμβάνοντας πάνω από μία αποθήκες από όπου μπορούν τα οχήματα να ξεκινήσουν τη διαδρομή τους. Μερικές από τις παραλλαγές του ΠΔΟ θα παρουσιαστούν αναλυτικότερα στο Κεφ. 2.2.

Το ΠΔΟ αποτελεί πρόκληση για τους υπολογιστές λόγω της συνδυαστικής του φύσης, των περίπλοκων αλληλεξαρτήσεων μεταξύ των διαδρομών και των αναθέσεων των πελατών στα οχήματα αλλά και των πρόσθετων περιορισμών λόγω της δυναμικής φύσης των πραγματικών σεναρίων. Συγκριτικά με το επίσης διαδεδομένο πρόβλημα του πλανόδιου πωλητή (travelling salesman problem, TSP), το οποίο εστιάζει σε έναν μόνο πωλητή (όχημα) που επισκέπτεται όλες τις πόλεις (πελάτες) ακριβώς μία φορά και επιστρέφει στο σημείο εκκίνησης διανύοντας την ελάχιστη συνολικά απόσταση, το ΠΔΟ βελτιστοποιεί τις διαδρομές για πολλά οχήματα ταυτόχρονα. Καθώς ο αριθμός των πελατών αυξάνεται, ο αριθμός των πιθανών λύσεων αυξάνεται παραγοντικά, καθιστώντας την εύρεση βέλτιστων ή σχεδόν βέλτιστων λύσεων μέσα σε εύλογο χρονικό διάστημα μία υπολογιστικά απαιτητική διαδικασία.



Εικόνα 2.1. *Απεικόνιση των TSP και VRP*

Στο επίκεντρο της μελέτης των ΠΔΟ βρίσκεται η υπολογιστική πολυπλοκότητα (computational complexity) των προβλημάτων αυτών. Η υπολογιστική πολυπλοκότητα στα προβλήματα βελτιστοποίησης αναφέρεται στον χρόνο και τους πόρους που απαιτούνται για να βρεθεί η βέλτιστη λύση. Αυτό εξαρτάται από το μέγεθος των δεδομένων εισόδου, την ποιότητα της λύσης που ζητείται και τους διαθέσιμους πόρους. Γενικότερα, τα προβλήματα βελτιστοποίησης ταξινομούνται σε διάφορες κατηγορίες πολυπλοκότητας με την κάθε μία να εμφανίζει διακριτές υπολογιστικές προκλήσεις. Χαρακτηριστικά παραδείγματα, μεταξύ άλλων, τέτοιων κατηγοριών αποτελούν οι κατηγορίες (ή κλάσεις) πολυπλοκότητας "P" (polynomial) και "EXP" (exponential).



Εικόνα 2.2. *Η σχέση μεταξύ των δεδομένων εισόδου και των απαιτούμενων υπολογισμών*

Τα προβλήματα που ταξινομούνται στην κατηγορία "P" είναι αυτά για τα όποια υπάρχουν αλγόριθμοι πολυωνυμικού χρόνου, που στην πράξη σημαίνει ότι μπορούν να βρεθούν λύσεις βέλτιστες, ή πολύ κοντά σε αυτές, σε εύλογο χρονικό διάστημα. Παραδείγματα τέτοιων μεθόδων επίλυσης αποτελούν οι αλγόριθμοι ταξινόμησης (sorting algorithms), οι αλγόριθμοι εύρεσης του συντομότερου μονοπατιού (shortest path algorithms), οι αλγόριθμοι εύρεσης του ελαχίστου τανύοντος δέντρου (minimum spanning tree algorithms) κ.α.

Αντιθέτως, το ΠΔΟ εμπίπτει στη κατηγορία πολυπλοκότητας "NP-hard" (Lenstra & Rinnooy Kan, 1979). Η κλάση "NP" (nondeterministic polynomial time) υποδηλώνει ότι η εύρεση μιας βέλτιστης λύσης απαιτεί στο χειρότερο σενάριο εκθετικό χρόνο καθώς αυξάνεται το μέγεθος του προβλήματος, ενώ η κλάση "NP-hard" αναφέρεται σε προβλήματα που είναι τουλάχιστον όσο δύσκολα είναι τα δυσκολότερα προβλήματα της κατηγορίας "NP" και ίσως ακόμη δυσκολότερα έτσι ώστε να μην εμπίπτουν καν στην κατηγορία αυτή. Ωστόσο, οι παραλλαγές του ΠΔΟ μπορούν να παρουσιάζουν διαφορετικά επίπεδα πολυπλοκότητας, με ορισμένα εξ αυτών να κατατάσσονται στην κατηγορία "NP-complete", μία υπο-κατηγορία

της κλάσης “NP-hard” που δηλώνει ότι σίγουρα δεν ξεφεύγουν από την κλάση “NP”. Συμπερασματικά, η σωστή ταξινόμηση του ΠΔΟ είναι απαραίτητη για την κατανόηση της δυσκολίας στην επίλυση των προβλημάτων αυτών και την ανάγκη ανάπτυξης αποτελεσματικών αλγορίθμων για την αντιμετώπισή τους.

2.1.1. Μαθηματική μοντελοποίηση

Ακολουθεί η μοντελοποίηση του **ΠΔΟ με περιορισμούς χωρητικότητας:**

1. Παράμετροι του προβλήματος:

- N: Το σύνολο των πελατών ($N = \{1,2,...,n\}$).
- M: Το σύνολο των (πανομοιότυπων) οχημάτων ($M = \{1,2,...,m\}$).
- d_{ij}: Απόσταση μετάβασης (ή χρόνος μετάβασης τ_{ij}) από τον πελάτη i στον πελάτη j.
- q_i: Η ζήτηση του πελάτη i.
- Q: Η χωρητικότητα του κάθε οχήματος (θεωρούμε πανομοιότυπα οχήματα).
- s: Η αποθήκη (source).

2. Μεταβλητές απόφασης:

- x_{ij}^k: Δυαδική μεταβλητή απόφασης που υποδεικνύει εάν το όχημα k ταξιδεύει (απευθείας) από τον πελάτη i στον πελάτη j ($x_{ij}^k = 1$ εάν επιλεγεί η διαδρομή, 0 διαφορετικά).
- y_i^k: Δυαδική μεταβλητή απόφασης που υποδεικνύει εάν το όχημα k επισκέπτεται τον πελάτη i ($y_i^k = 1$ εάν τον επισκέπτεται, 0 διαφορετικά).

3. Αντικειμενική συνάρτηση:

$$F: Min \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} \cdot x_{ij}^k)$$

4. Περιορισμοί:

- Κάθε πελάτης θα εξυπηρετηθεί μία φορά και από ένα ακριβώς όχημα:
  $\sum_{k=1}^{m} y_i^k = 1 \ \forall i \in N$
- Μη υπέρβαση της χωρητικότητας των οχημάτων:
  $\sum_{i=1}^{n} q_i \cdot y_i^k \leq Q \ \ \forall k \in M$
- Κάθε όχημα που επισκέπτεται έναν πελάτη i θα πρέπει και να αποχωρήσει από αυτόν:
  $\sum_{i=1}^{n} x_{ij}^k = \sum_{i=1}^{n} x_{ji}^k = y_i^k \ \forall j \in N, \ \forall k \in M$
- Η διαδρομή του κάθε οχήματος ξεκινάει και τελειώνει στην αποθήκη:
  $\sum_{i=1}^{n} x_{si}^k = 1$ και $\sum_{i=1}^{n} x_{is}^k = 1 \ \forall k \in M$
- Εξάλειψη εμφάνισης υπο-διαδρομών:
  $\sum_{i \in S} \sum_{j \notin S} x_{ij}^k \geq 1 \ \forall S \subset N, S \neq \emptyset, \forall k \in M$
- Δυαδικοί περιορισμοί στις μεταβλητές απόφασης:
  x_{ij}^k , $y_i^k \in \{0,1\} \ \forall i,j \in N, \forall k \in M, \ i \neq j$

2.2. Παραλλαγές του ΠΔΟ

Πολλές παραλλαγές του ΠΔΟ έχουν προταθεί και μελετηθεί για την αντιμετώπιση της πολυπλοκότητας των σεναρίων του πραγματικού κόσμου. Τα προβλήματα αυτά επεκτείνουν το κλασικό πλαίσιο εισάγοντας πρόσθετους περιορισμούς ή στόχους, καλύπτοντας διαφορετικές επιχειρησιακές απαιτήσεις. Από τη διασφάλιση έγκαιρων παραδόσεων εντός καθορισμένων χρονικών παραθύρων μέχρι τον ταυτόχρονο χειρισμό ετερογενών στόλων, κάθε παραλλαγή παρουσιάζει τις δικές τις ιδιαιτερότητες.

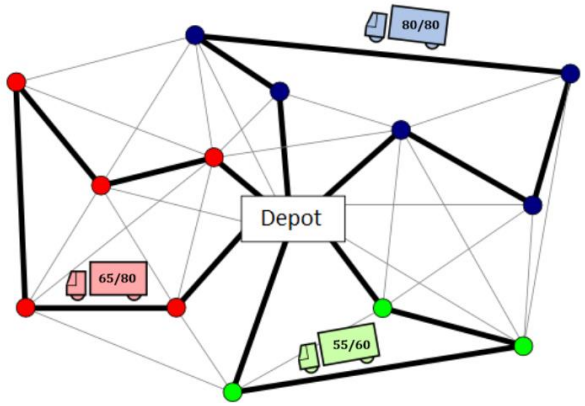
2.2.1. ΠΔΟ με περιορισμό στον χρόνο μετάβασης (Time-Constrained VRP, TCVRP)

Η παραλλαγή αυτή εστιάζει αποκλειστικά στην βελτιστοποίηση των διαδρομών των οχημάτων τηρώντας παράλληλα περιορισμούς που σχετίζονται με τον μέγιστο επιτρεπόμενο χρόνο που διαρκεί η κάθε διαδρομή. Η μέγιστη διάρκεια για τη διαδρομή του κάθε οχήματος συχνά αναφέρεται ως παράμετρος «max route time» και μπορεί να έχει την ίδια τιμή για όλα τα οχήματα ή και διαφορετική για το κάθε ένα.

Τα πραγματικά σενάρια όπου η παραλλαγή αυτή έχει εφαρμογή αφορούν κυρίως την αστική εφοδιαστική (urban logistics), όπου οι εταιρείες πρέπει να σχεδιάσουν διαδρομές για πλοήγηση σε περιοχές με κυκλοφοριακή συμφόρηση εντός καθορισμένων χρονικών περιορισμών. Επιπλέον, τα συστήματα διαχείρισης των μέσων μαζικής μεταφοράς οφείλουν να συμμορφώνονται με προκαθορισμένα δρομολόγια με σκοπό την παροχή αξιόπιστων υπηρεσιών στους επιβάτες.

2.2.2. ΠΔΟ με περιορισμό χωρητικότητας (Capacity-Constrained VRP, CVRP)

Το CCVRP είναι μια παραλλαγή του κλασικού ΠΔΟ που επιβάλλει περιορισμούς στη χωρητικότητα (capacity) των οχημάτων. Κάθε όχημα έχει περιορισμένη χωρητικότητα σε προϊόντα και ο στόχος είναι η βελτιστοποίηση των διαδρομών για την εξυπηρέτηση ενός συνόλου πελατών, διασφαλίζοντας παράλληλα ότι η συνολική ζήτηση των πελατών που εξυπηρετούνται από κάθε όχημα δεν υπερβαίνει τη χωρητικότητά του. Υπάρχει επίσης περίπτωση τα οχήματα να έχουν διαφορετική χωρητικότητα μεταξύ τους και παράλληλος στόχος είναι η ελαχιστοποίηση των οχημάτων που θα χρησιμοποιηθούν αξιοποιώντας πλήρως τη χωρητικότητα του κάθε οχήματος.

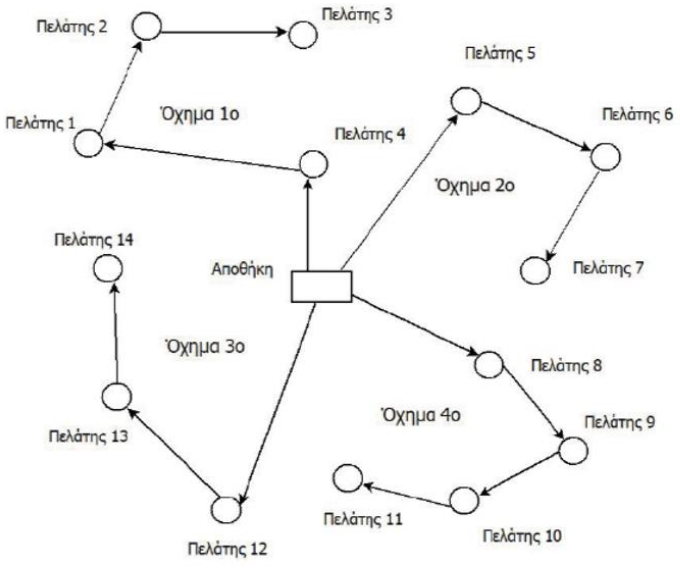


Εικόνα 2.3. *Παράδειγμα CVRP με 13 πελάτες και 3 οχήματα*

Η παραλλαγή CVRP βρίσκει πρακτική εφαρμογή σε διάφορους τομείς, συμπεριλαμβανομένου του λιανικού και ηλεκτρονικού εμπορίου για διαδρομές παράδοσης τελευταίου μιλίου, της συλλογής απορριμμάτων και της ανακύκλωσης μέσω αποτελεσματικού σχεδιασμού δρομολογίων μεγαλύτερων και μικρότερων απορριμματοφόρων, των ταχυδρομικών υπηρεσιών για διαδρομές παράδοσης αλληλογραφίας, στα δημόσια μέσα μεταφοράς και στην εφοδιαστική υγειονομικής περίθαλψης για τη μεταφορά ιατρικών προμηθειών.

2.2.3. Ανοιχτό ΠΔΟ (Open VRP, OVRP)

Η βασική διαφορά του OVRP είναι ότι τα οχήματα δεν απαιτείται να επιστρέψουν σε κάποια κεντρική αποθήκη μετά την ολοκλήρωση των διαδρομών τους. Αντ' αυτού, κάθε όχημα, όταν ολοκληρώνει την εξυπηρέτηση των πελατών, μπορεί να συνεχίσει προς την επόμενη δραστηριότητα ή πελάτη στο πλαίσιο της εργασίας του.

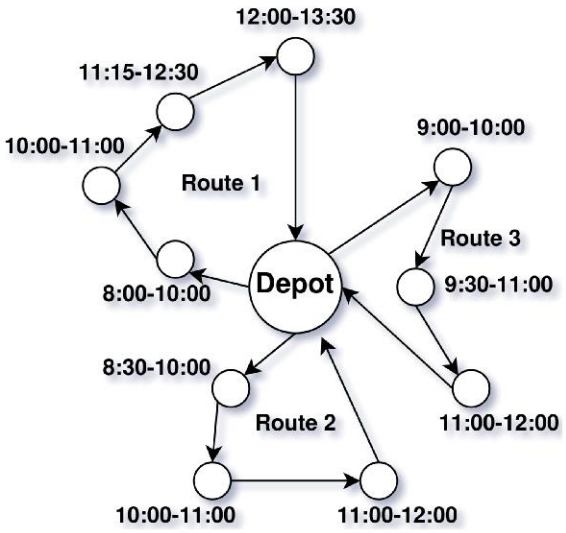


Εικόνα 2.4. *Παράδειγμα OVRP με 14 πελάτες και 4 οχήματα*

Το ανοιχτό ΠΔΟ εφαρμόζεται στην πραγματικότητα όταν μία εταιρεία Α δεν έχει στην κατοχή της δικά της οχήματα, αλλά για κάθε μεταφορά που θέλει να κάνει νοικιάζει οχήματα από άλλες εταιρείες. Η βέλτιστη δρομολόγηση των οχημάτων θα οδηγήσει σε γρηγορότερη εξυπηρέτηση των πελατών της εταιρείας Α αλλά και συνολικά σε μικρότερη διανυόμενη απόσταση, άρα και σε μικρότερο κόστος, αφού θα αποδεσμευθούν νωρίτερα τα νοικιαζόμενα οχήματα. Τα οχήματα την ξένης εταιρείας ξεκινούν τη διαδρομή τους από την αποθήκη της εταιρείας Α, εξυπηρετούν το καθένα τους πελάτες που τους αναθέτει η εταιρεία Α και έπειτα επιστρέφουν στην εταιρεία όπου ανήκουν (ή συνεχίζουν τις διανομές για κάποια άλλη εταιρεία Β, κάτι που δεν μας αφορά).

2.2.4. ΠΔΟ με χρονικά παράθυρα (VRP with Time Windows, VRPTW)

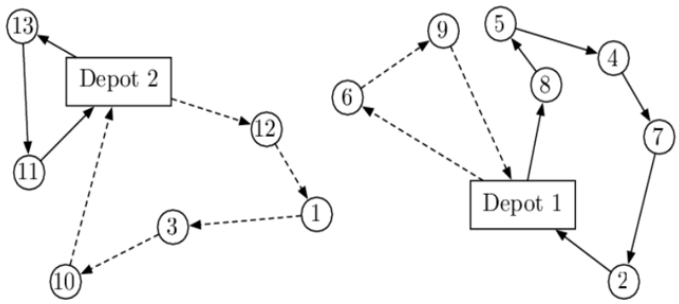
Το VRPTW εφαρμόζεται ευρέως σε όλους τους τομείς της σύγχρονης βιομηχανίας και αναφέρεται στο πρόβλημα που προκύπτει κατά τον σχεδιασμό και προγραμματισμό των διαδρομών οχημάτων με χρονικά παράθυρα για την εκτέλεση εργασιών. Αντίστοιχα και κάποιο προϊόν ή υπηρεσία μπορεί να παραδοθεί στον κάθε πελάτη όχι οποιαδήποτε ώρα μέσα στην ημέρα, αλλά σε κάποιο ορισμένο διάστημα που αυτός έχει ορίσει και γνωστοποιήσει. Αυτό εισάγει πρόσθετη πολυπλοκότητα καθώς οι διαδρομές πρέπει να σχεδιάζονται όχι μόνο για να ελαχιστοποιηθεί η συνολική απόσταση των διαδρομών αλλά και για να διασφαλιστεί ότι όλες οι παραδόσεις γίνονται εντός των αντίστοιχων χρονικών παραθύρων των πιο «ελαστικών» αλλά και «απαιτητικών» πελατών. Η επίλυση του απαιτεί προσαρμοσμένους και πιο εξελιγμένους αλγόριθμους, καθώς η μη σωστή προσέγγιση του θα οδηγήσει σε λύσεις με πολύ περισσότερα οχήματα από τα απαραίτητα ή/και μεγάλους **χρόνους αναμονής** στις τοποθεσίες των πελατών μέχρι να ανοίξει το χρονικό τους παράθυρο.



Εικόνα 2.5. *Παράδειγμα VRPTW με 10 πελάτες και 3 οχήματα*

2.2.5. ΠΔΟ με πολλαπλές αποθήκες (Multi-Depot VRP, MDVRP)

Το MDVRP είναι μια επέκταση του κλασικού ΠΔΟ, όπου υπάρχουν πάνω από μία αποθήκες από όπου μπορούν τα οχήματα να ξεκινήσουν τη διαδρομή τους. Κάθε αποθήκη έχει ένα σύνολο οχημάτων, προκαθορισμένο ή μη, και ο στόχος είναι να καθοριστούν οι βέλτιστες διαδρομές για αυτά τα οχήματα ώστε να εξυπηρετηθεί ένα σύνολο πελατών που βρίσκονται σε διαφορετικές τοποθεσίες. Έτσι προστίθεται επιπλέον πολυπλοκότητα στο πρόβλημα καθώς πρέπει να καθοριστεί και το σημείο εκκίνησης και τερματισμού του κάθε οχήματος. Ταυτόχρονα πρέπει να ικανοποιούνται περιορισμοί προερχόμενοι και από άλλες παραλλαγές που μπορεί να αφορούν την περιορισμένη χωρητικότητα των οχημάτων, ή/και την μη παραβίαση κάποιων χρονικών παραθύρων (time windows), ή/και την μη υπέρβαση των ορίων μέγιστου χρόνου ταξιδιού (max route time) του κάθε οχήματος. Αυτοί οι περιορισμοί μπορεί να λαμβάνουν χώρα μεμονωμένα ή και ταυτόχρονα σε πιο πολύπλοκα προβλήματα.

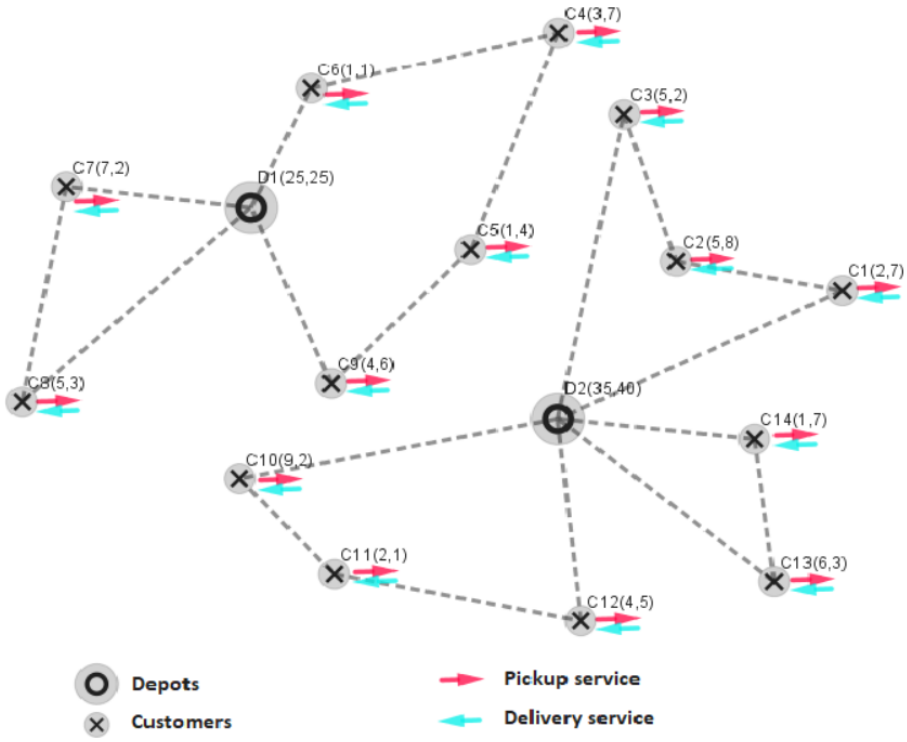


Εικόνα 2.6. *Παράδειγμα MDVRP με 2 αποθήκες, 4 οχήματα και 13 πελάτες*

Πρακτικές εφαρμογές του MDVRP συναντώνται σε διάφορες βιομηχανίες όπου πολλές αποθήκες εμπλέκονται στη διανομή αγαθών ή υπηρεσιών, όπως η μεταφορά καυσίμων στα πρατήρια καυσίμων μεγάλων πολιτειών, ο σχεδιασμός που αφορά τις υπηρεσίες έκτακτης ανάγκης (πυροσβεστική, αστυνομία, ασθενοφόρα), η συλλογή απορριμμάτων κ.α.

2.2.6. ΠΔΟ με ταυτόχρονη διανομή και παραλαβή (VRP with Pickup and Delivery, VRPPD)

Στο VRPDP τα οχήματα έχουν την αποστολή να παραλαμβάνουν προϊόντα από καθορισμένες τοποθεσίες παραλαβής (pickup locations) και να τα παραδίδουν σε αντίστοιχες τοποθεσίες παράδοσης (drop-off locations), ενώ βελτιστοποιούνται οι διαδρομές για να ελαχιστοποιείται η συνολική διανυόμενη απόσταση. Στη βασική μορφή του προβλήματος αντιστοιχούνται σε κάθε πελάτη δύο ποσότητες, αυτή που θα διανεμηθεί και αυτή που θα παραληφθεί.

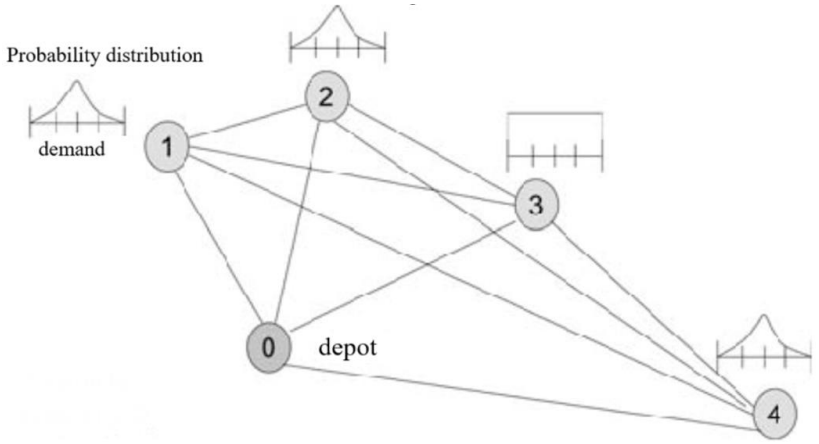


Εικόνα 2.7*. Παράδειγμα VRPPD με 2 αποθήκες και 14 πελάτες*

Στις εμπορευματικές μεταφορές, το VRPDP βοηθά τις εταιρείες logistics να βελτιστοποιούν τις διαδρομές των οχημάτων τους ώστε να παραλαμβάνουν αγαθά από προμηθευτές και να τα παραδίδουν σε αποθήκες ή λιανοπωλητές. Χρησιμοποιείται επίσης σε εργασίες εξυπηρέτησης (service), όπως επισκευή οικιακών συσκευών, όπου οι τεχνικοί παραλαμβάνουν ανταλλακτικά από προμηθευτές και τα παραδίδουν σε πελάτες.

2.2.7. ΠΔΟ με στοχαστική ζήτηση (VRP with Stochastic Demand, VRPSD)

Το VRPSD είναι μια ακόμη παραλλαγή όπου η ζήτηση για προϊόντα ή υπηρεσίες σε κάθε τοποθεσία πελάτη είναι αβέβαιη και υπόκειται σε τυχαία μεταβλητότητα. Έτσι, η ζήτηση του κάθε πελάτη δεν είναι ντετερμινιστική αλλά ακολουθεί μια κατανομή πιθανοτήτων, όπως μια κανονική ή Poisson κατανομή. Αυτή η αβεβαιότητα προσθέτει πολυπλοκότητα στη διαδικασία βελτιστοποίησης, καθώς λαμβάνεται υπόψη η πιθανολογική φύση της ζήτησης και οι αποφάσεις ισορροπούν μεταξύ της ελαχιστοποίησης του κόστους και της διαχείρισης των κινδύνων που σχετίζονται με πιθανές διακυμάνσεις της ζήτησης. Η επίλυση προσεγγίζεται με πιθανολογικά μοντέλα και αλγόριθμους στοχαστικής βελτιστοποίησης για τη δημιουργία ισχυρών αλλά και ελαστικών διαδρομών οχημάτων που μπορούν να προσαρμοστούν στην μεταβλητότητα της ζήτησης.



Εικόνα 2.8. *Παράδειγμα VRPSD με 4 πελάτες*

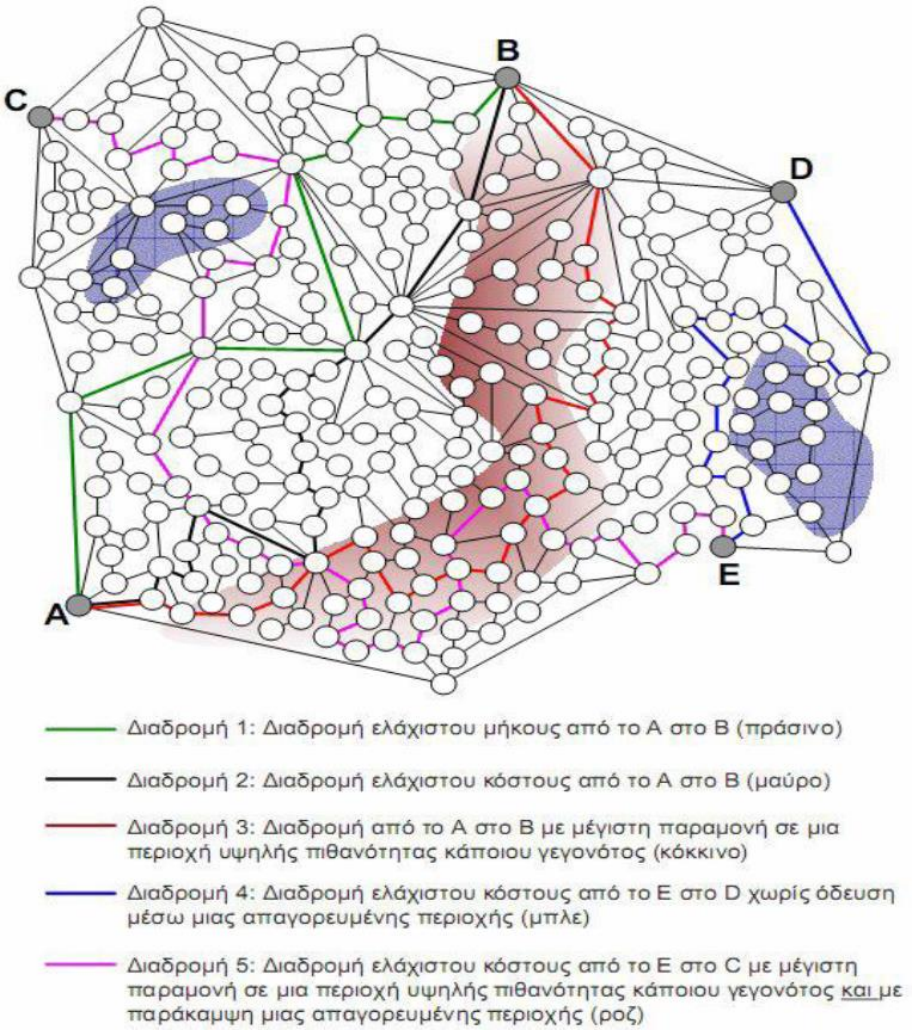
Το VRPSD εφαρμόζεται σε σενάρια όπου η μεταβλητότητα της ζήτησης είναι σημαντική ή δύσκολο να προβλεφθεί με ακρίβεια. Για παράδειγμα, στις επιχειρήσεις αστικής εφοδιαστικής το VRPSD βοηθά τις εταιρείες να βελτιστοποιούν τις διαδρομές των οχημάτων τους ώστε να προσαρμόζονται στα κυμαινόμενα μοτίβα ζήτησης που προκαλούνται από παράγοντες όπως κυκλοφοριακή συμφόρηση, καιρικές συνθήκες ή εποχιακές αλλαγές στη ζήτηση. Ομοίως, στη διαχείριση της εφοδιαστικής αλυσίδας, χρησιμοποιείται για τη βελτιστοποίηση των δικτύων διανομής και των στρατηγικών διαχείρισης αποθεμάτων για τον μετριασμό του αντίκτυπου της αβεβαιότητας της ζήτησης στις διαδικασίες παραγωγής και διανομής.

2.2.8. Δυναμικό ΠΔΟ (Dynamic VRP, DVRP)

Μία από τις πιο σύνθετες παραλλαγές του ΠΔΟ είναι αυτή στην οποία η δρομολόγηση καθορίζεται όχι μόνο με στόχο τη διανομή, αλλά και την αποτελεσματικότερη εξυπηρέτηση αιτήσεων οι οποίες μπορεί να προκύψουν κατά τη διάρκεια των δρομολογίων. Το DVRP λαμβάνει υπόψη τις δυναμικές αλλαγές στο περιβάλλον (απαγορευμένες περιοχές, κυκλοφοριακή συμφόρηση κ.α.) ή στα αιτήματα των πελατών με την πάροδο του χρόνου. Αυτές οι αλλαγές θα μπορούσαν να περιλαμβάνουν νέα αιτήματα πελατών, ακυρώσεις, καθυστερήσεις ή τροποποιήσεις (π.χ. αλλαγή χρονικών παραθύρων, μείωση/αύξηση ζήτησης) σε υπάρχοντα αιτήματα.

Παραδείγματα επίλυσης του DVRP είναι οι εφαρμογές τύπου courier, οδικής βοήθειας, υπηρεσιών παράδοσης φαγητού (E-food, Wolt, Deliveroo κ.α.) όπου δεν απασχολεί μόνο η διανομή, αλλά είναι επιδιωκόμενο ο διανομέας να βρίσκεται μέσα σε μια γεωγραφική περιοχή στην οποία υπάρχει υψηλή πιθανότητα να εκδηλωθεί ζήτηση, όπως αντιλαμβανόμαστε εύκολα στην περίπτωση των υπηρεσιών delivery. Τα δεδομένα εισόδου

ανανεώνονται συνεχώς με παρακολούθηση σε πραγματικό χρόνο των θέσεων των διανομέων, την εκτίμηση του χρόνου παράδοσης, τον έλεγχο των αποθεμάτων και της διαθεσιμότητας των προϊόντων από τα εστιατόρια.



Εικόνα 2.9. *Παράδειγμα DVRP*

2.3. ΠΔΟ πολλαπλών αντικειμενικών συναρτήσεων (Multi-Objective VRP, MOVRP)

Το MOVRP εξετάζει πολλαπλούς αντικρουόμενους (ή μη) στόχους που πρέπει να βελτιστοποιηθούν ταυτόχρονα, και όπως γίνεται αντιληπτό οι παραλλαγές του μπορεί να είναι πολλές. Σε κάθε τέτοια περίπτωση, αντί να αναζητείται μία ενιαία βέλτιστη λύση, στόχος είναι να βρεθεί ένα σύνολο πολλών και διαφορετικών λύσεων που αντιπροσωπεύουν συμβιβασμούς (trade-offs) μεταξύ διαφορετικών στόχων. Αυτές οι λύσεις αποτελούν το σύνολο «Pareto βέλτιστων» λύσεων, η αναλυτικότερη περιγραφή των οποίων θα γίνει στο Κεφ. 3.4.

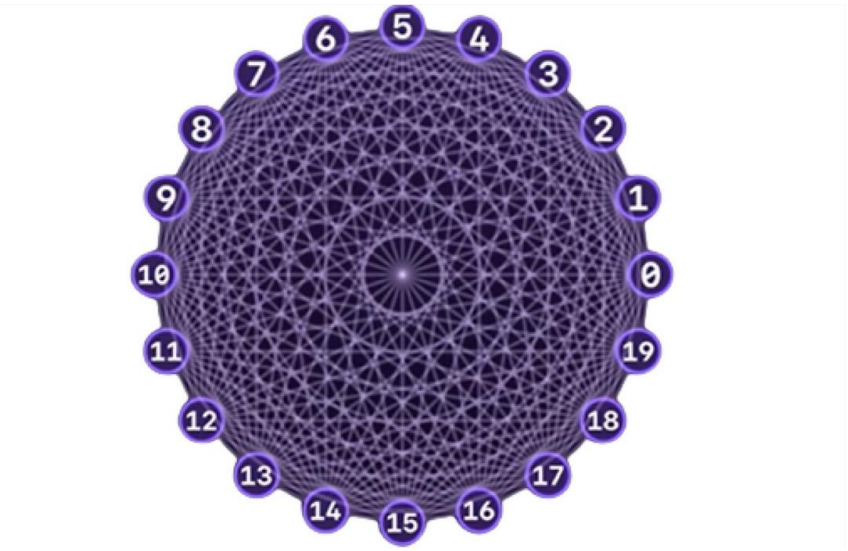
Οι στόχοι αυτοί μπορεί να περιλαμβάνουν την ελαχιστοποίηση της συνολικής απόστασης των διαδρομών, τη μεγιστοποίηση των επιπέδων εξυπηρέτησης των πελατών, την ελαχιστοποίηση του μεγέθους του στόλου των οχημάτων, την ελαχιστοποίηση της ανισορροπίας φορτίων μεταξύ των οχημάτων (για λόγους ασφαλείας κ.α.), την ελαχιστοποίηση των περιβαλλοντικών επιπτώσεων από εκπομπές άνθρακα, την ελαχιστοποίηση της αναμονής σε τοποθεσίες πελατών σε περίπτωση που υπάρχουν χρονικά παράθυρα και πολλοί άλλοι, ανάλογα με τη φύση του προβλήματος που επιλύεται.

Κεφάλαιο 3. Βελτιστοποίηση

3.1. Εισαγωγή

Η βελτιστοποίηση, ιδιαίτερα στον τομέα της επιχειρησιακής έρευνας, εστιάζει στην εύρεση της καλύτερης λύσης από ένα σύνολο εφικτών λύσεων για ένα δεδομένο πρόβλημα. Ειδικότερα, η συνδυαστική βελτιστοποίηση ασχολείται με προβλήματα όπου ο χώρος λύσεων (solution space) είναι διακριτός και συνήθως πολύ μεγάλος. Σκοπός είναι η εύρεση της καλύτερης διάταξης ή συνδυασμού στοιχείων για την επίτευξη του βέλτιστου αποτελέσματος. Τα προβλήματα “NP-hard”, όπως το πρόβλημα δρομολόγησης οχημάτων (VRP) και το πρόβλημα του πλανόδιου πωλητή (TSP) ανήκουν σε αυτήν την κατηγορία. Επιπλέον, τα προβλήματα αυτά συχνά στερούνται εγγενών δομών ή ιδιοτήτων που θα επέτρεπαν την αποτελεσματική βελτιστοποίηση με τη χρήση παραδοσιακών τεχνικών, όπως οι μέθοδοι brute-force (εξαντλητική ανάπτυξη δέντρου εφικτών λύσεων), οι αλγόριθμοι ακέραιου γραμμικού προγραμματισμού (π.χ. Branch and bound), οι προσεγγιστικές μέθοδοι αριθμητικής ανάλυσης (π.χ. Newton-Raphson), ο δυναμικός προγραμματισμός κ.α.

Όπως προαναφέρθηκε, στην περίπτωση του VRP και του TSP, ο αριθμός των πιθανών λύσεων αυξάνεται εκθετικά με την αύξηση του αριθμού των πελατών ή των πόλεων αντίστοιχα. Για παράδειγμα, στο TSP με 10 πόλεις, υπάρχουν ήδη πάνω από 3,6 εκατομμύρια (10! = 3.628.800) πιθανές διαδρομές προς εξέταση. Καθώς ο αριθμός των πόλεων αυξάνεται (11! = 39.916.800, 12! = 479.001.600, …), ο χώρος αναζήτησης γίνεται αστρονομικά μεγάλος, καθιστώντας ανέφικτη ή απαγορευτικά χρονοβόρα την εξερεύνηση όλων των πιθανών λύσεων ακόμη και από σύγχρονους υπολογιστές.



Εικόνα 3.1. *Παράδειγμα TSP με 19 πόλεις (121 τετράκις εκατομμύρια πιθανές λύσεις)*

Όπως γίνεται αντιληπτό, η επίλυση ενός προβλήματος όπως το ΠΔΟ, με περισσότερους από έναν πωλητές (οχήματα), με την δυνητική ύπαρξη περισσότερων του ενός σημείων εκκίνησης (αποθηκών) και υπό αυστηρούς περιορισμούς που επιβάλλονται από χρονικά παράθυρα, χωρητικότητες οχημάτων, ταυτόχρονες διανομές και παραλαβές, μονόδρομους, κυκλοφοριακή συμφόρηση, δυναμικά περιβάλλοντα ζήτησης κ.α., καθίσταται ακόμη πιο πολύπλοκη διαδικασία.

3.2. Ευρετικοί αλγόριθμοι (Heuristics)

Στη μαθηματική βελτιστοποίηση και την επιστήμη των υπολογιστών, οι ευρετικές μέθοδοι έχουν σχεδιαστεί για την ταχύτερη επίλυση προβλημάτων όταν οι κλασικές μέθοδοι είναι πολύ αργές για την εύρεση μιας ακριβούς ή προσεγγιστικής λύσης ή όταν αποτυγχάνουν να βρουν μία ακριβής λύση σε έναν χώρο λύσεων. Αυτό επιτυγχάνεται με το να ανταλλάσσεται η ποιότητα και η πληρότητα μίας λύσης για την ταχύτητα εύρεσης αυτής. Κατά κάποιον τρόπο, μπορεί να θεωρηθεί συντόμευση που μας βοηθά να παράγουμε βέλτιστες λύσεις (για μικρού μεγέθους προβλήματα) ή ικανοποιητικά υπο-βέλτιστες λύσεις (για μεγαλύτερα προβλήματα) σε εύλογο χρονικό διάστημα. Μία συνήθης κατηγοριοποίηση των ευρετικών αλγορίθμων είναι η ακόλουθη:

1. Κατασκευαστικοί ευρετικοί αλγόριθμοι
2. Ευρετικοί αλγόριθμοι δύο φάσεων
3. Ευρετικοί αλγόριθμοι βελτίωσης των διαδρομών

3.2.1. Κατασκευαστικοί ευρετικοί αλγόριθμοι (Constructive heuristics)

Οι κατασκευαστικοί ευρετικοί αλγόριθμοι ξεκινούν με μια κενή λύση και επεκτείνουν επανειλημμένα την τρέχουσα λύση προσθέτοντας πελάτες βάσει κάποιου κριτηρίου (μικρότερο κόστος μετάβασης, μεγαλύτερη ζήτηση κ.α.) ελέγχοντας ταυτόχρονα τη μη παραβίαση των περιορισμών του προβλήματος, μέχρι να ληφθεί μια πλήρης λύση, δηλαδή μέχρι να μην υπάρχει πελάτης που δεν έχει εξυπηρετηθεί ή μέχρι να ικανοποιηθεί κάποιο άλλο κριτήριο τερματισμού. Οι επιμέρους κατηγορίες των αλγορίθμων αυτών περιλαμβάνουν τους ακολουθιακούς και τους παράλληλους κατασκευαστικούς αλγόριθμους.

Μεταξύ αυτών, υπάρχουν και οι **άπληστες προσεγγίσεις (greedy algorithms)** κατά τις οποίες κάθε φορά επιλέγεται η καλύτερη επιλογή μετάβασης για την παρούσα κατάσταση, χωρίς να λαμβάνονται υπόψη οι πιθανές μελλοντικές επιπτώσεις στο συνολικό κόστος της λύσης. Η χρήση αυτών γίνεται με την ελπίδα ότι επιλέγοντας σε κάθε βήμα το τοπικό βέλτιστο, θα καταλήξουμε σε μία ολικά βέλτιστη λύση. Αυτό προφανώς στην πλειοψηφία των περιπτώσεων δεν συμβαίνει, αλλά αποτελεί εργαλείο για τη δημιουργία καλών αρχικών λύσεων που στη συνέχεια θα βελτιωθούν με πιο προηγμένες τεχνικές.

3.2.1.1. Ακολουθιακοί κατασκευαστικοί αλγόριθμοι (Sequential constructive heuristics)

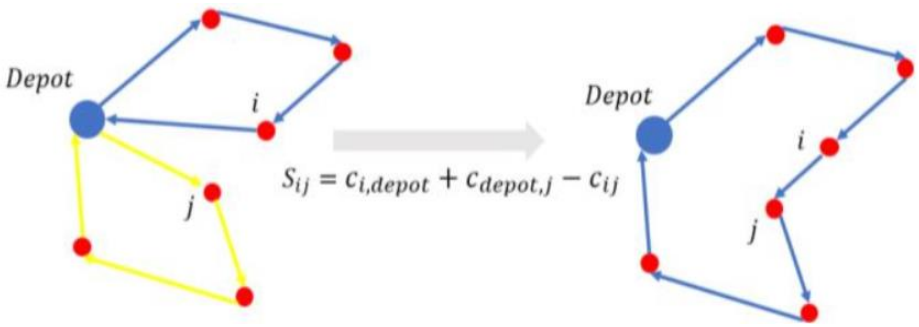
Πρόκειται για μεθόδους που χρησιμοποιούνται για τη δημιουργία αρχικών, εφικτών βάσει των περιορισμών, λύσεων. Οι αλγόριθμοι αυτοί ξεκινούν με μια κενή λύση και δημιουργούν σταδιακά διαδρομές μία-μία προσθέτοντας επαναληπτικά πελάτες μέχρι να εξυπηρετηθούν όλοι. Μερικοί αλγόριθμοι αυτής της κατηγορίας που χρησιμοποιούνται για την επίλυση ΠΔΟ είναι οι ακόλουθοι:

- **Πλησιέστερος γείτονας (Nearest neighbor):** Πρόκειται για έναν άπληστο αλγόριθμο που ξεκινά με έναν αυθαίρετο πελάτη και επιλέγει επαναληπτικά τον πλησιέστερο βάσει απόστασης μη εξυπηρετημένο πελάτη για να δημιουργήσει μία διαδρομή. Όταν παραβιαστεί κάποιος περιορισμός (χωρητικότητα οχήματος, μέγιστος χρόνος διαδρομής κ.α.) τότε η διαδρομή ολοκληρώνεται με επιστροφή στην αποθήκη και ξεκινάει η δημιουργία της διαδρομής του επόμενου οχήματος, μέχρι να εξυπηρετηθούν όλοι οι πελάτες. Είναι απλός και υπολογιστικά αποδοτικός αλγόριθμος, χωρίς όμως να εγγυάται πάντα τη δημιουργία καλών αρχικών λύσεων.
- **Αλγόριθμος εξοικονομήσεων (Savings algorithm):** Επίσης γνωστός και ως αλγόριθμος εξοικονομήσεων των Clarke & Wright, υπολογίζει τις πιθανές

εξοικονομήσεις σε απόσταση συνδυάζοντας διαφορετικές διαδρομές. Αναλυτικότερα, τα βήματα του αλγορίθμου είναι:

1. Υπολογισμός εξοικονομήσεων: Για κάθε ζεύγος πελατών *i* και *j* (εξαιρουμένης της αποθήκης), υπολόγισε την εξοικονόμηση S_{ij} που θα επιτυγχανόταν με τη συγχώνευση των διαδρομών των *i* και *j* σε σύγκριση με την εξυπηρέτησή τους ξεχωριστά. Η εξοικονόμηση S_{ij} υπολογίζεται ως η απόσταση που εξοικονομείται συνδυάζοντας τις απευθείας διαδρομές προς τους πελάτες *i* και *j* μείον την απόσταση μεταξύ αυτών. Δηλαδή $S_{ij} = d_{0i} + d_{0j} - d_{ij}$ όπου d_{0i} και d_{0j} είναι οι αποστάσεις από την αποθήκη έως τους πελάτες *i* και *j*, και d_{ij} είναι η απόσταση μεταξύ των πελατών *i* και *j*.
2. Ταξινόμηση εξοικονομήσεων: Ταξινόμηση όλων των εξοικονομήσεων S_{ij} σε φθίνουσα σειρά.
3. Συγχώνευση διαδρομών: Εκκίνηση με μία διαδρομή για κάθε πελάτη. Στη συνέχεια, αναζήτησε επαναληπτικά μέσα στην ταξινομημένη λίστα εξοικονομήσεων και συγχώνευσε τις διαδρομές των πελατών *i* και *j*, εάν κάτι τέτοιο έχει ως αποτέλεσμα την μείωση της συνολικής απόστασης και δεν παραβιάζει κανέναν περιορισμό.
4. Ολοκλήρωση διαδρομών: Μετά τη συγχώνευση, οι διαδρομές που προκύπτουν μπορεί να χρειαστεί να προσαρμοστούν για να ικανοποιηθούν τυχόν πρόσθετοι περιορισμοί σε πιο σύνθετα προβλήματα.

Καθώς αφορά επίσης μια ευρετική μέθοδο με άπληστη λογική (κατά το βήμα 3) δεν μας εγγυάται μια βέλτιστη λύση. Ωστόσο, συχνά παράγει γρήγορα λύσεις καλής ποιότητας και προτιμάται από τον αλγόριθμο του πλησιέστερου γείτονα, ειδικά για μεγάλης κλίμακας προβλήματα δρομολόγησης.



Εικόνα 3.2. *Παράδειγμα υπολογισμού εξοικονόμησης και συγχώνευσης διαδρομών*

3.2.1.2. Παράλληλοι κατασκευαστικοί αλγόριθμοι (Parallel constructive heuristics)

Οι αλγόριθμοι παράλληλης κατασκευής περιλαμβάνουν την κατασκευή πολλαπλών τμημάτων της λύσης ταυτόχρονα ή παράλληλα. Αυτή η προσέγγιση μπορεί να είναι επωφελής κατά την κατασκευή μεγάλων λύσεων ή όταν η δομή του προβλήματος επιτρέπει την παραλληλοποίηση. Αυτές οι μέθοδοι μπορούν δυνητικά να επιταχύνουν τη διαδικασία κατασκευής μίας λύσης αξιοποιώντας τις δυνατότητες παράλληλης επεξεργασίας (parallel processing). Αφού κάθε διαδρομή έχει κατασκευαστεί ανεξάρτητα, τα αποτελέσματα συνδυάζονται για να σχηματίσουν την πλήρη λύση. Αυτή η ενοποίηση μπορεί να περιλαμβάνει συγχώνευση διαδρομών ή προσαρμογή των αναθέσεων των πελατών από διαδρομή σε διαδρομή. Παραδείγματα αλγορίθμων αυτής της υπο-κατηγορίας είναι:

- **Αλγόριθμος παράλληλης εισαγωγής (Parallel insertion algorithm):** Με τη χρήση της μεθόδου αυτής δημιουργούνται επαναληπτικά διαδρομές εισάγοντας πελάτες ταυτόχρονα σε πολλαπλές διαδρομές, που μπορεί για αρχή να είναι προκαθορισμένες άδειες διαδρομές όταν ο στόλος οχημάτων είναι συγκεκριμένου μεγέθους, με βάση ορισμένα κριτήρια (π.χ. επιλογή κάθε φορά του πελάτη με τη μεγαλύτερη ζήτηση, ή του κοντινότερου). Οι πελάτες προστίθενται σε κάθε διαδρομή μέχρι να παραβιαστεί κάποιος περιορισμός. Σε άλλες περιπτώσεις προστίθενται πελάτες και επιτρέπεται η παραβίαση (ή «χαλάρωση») των περιορισμών ελπίζοντας ότι στη συνέχεια βελτιώνοντας κάθε διαδρομή ξεχωριστά (με χρήση αλγορίθμων τοπικής αναζήτησης που θα δούμε παρακάτω) θα προκύψει μια εφικτή λύση. Η διαδικασία σταματάει όταν προκύψουν μόνο εφικτές διαδρομές και καμία δεν επιδέχεται περαιτέρω βελτίωση.
- **Αλγόριθμος παράλληλων εξοικονομήσεων (Parallel savings algorithm):** Πρόκειται για παραλλαγή του αλγορίθμου των Clarke & Wright παραλληλίζοντας τη διαδικασία υπολογισμού των εξοικονομήσεων και επιτρέποντας την ταυτόχρονη αξιολόγηση πολλών ζευγών πελατών, με αποτέλεσμα τη σημαντική μείωση του υπολογιστικού χρόνου και την ταχεία κατασκευή λύσεων.

3.2.2. Ευρετικοί αλγόριθμοι δύο φάσεων (Two-phased heuristics)

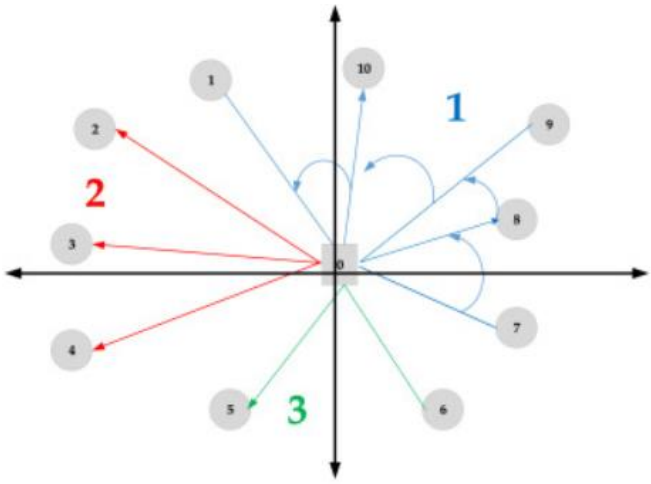
Οι μέθοδοι δύο φάσεων βασίζονται στην αποσύνθεση της διαδικασίας δημιουργίας μίας λύσης χωρίζοντας το πρόβλημα σε δύο ξεχωριστά υπο-προβλήματα (ή φάσεις):

1. Ομαδοποίηση (Clustering): Το σύνολο των πελατών χωρίζεται σε ομάδες με βάση κριτήρια όπως η χωρική εγγύτητα ή η ομοιότητα ζήτησης.

2. Δρομολόγηση (Routing): Αφού σχηματιστούν οι ομάδες, κατασκευάζονται διαδρομές εξυπηρέτησης πελατών με τη χρήση κάποιου κοινού κατασκευαστικού αλγορίθμου (πλησιέστερος γείτονας, αλγόριθμος εξοικονομήσεων κ.α.), αντιμετωπίζοντας κάθε ομάδα πελατών σαν ένα αυτόνομο ΠΔΟ.

Παραδείγματα αλγορίθμων αυτής της υπο-κατηγορίας είναι:

- **Αλγόριθμος σάρωσης (Sweep algorithm):** Ο αλγόριθμος σάρωσης των Gillet & Miller (1974) είναι μια ευρετική μέθοδος δύο φάσεων που χρησιμοποιείται για την επίλυση ΠΔΟ, και πιο συχνά για την παραλλαγή CVRP. Αναλυτικότερα, τα βήματα του αλγορίθμου είναι:
    1. Γωνιακή ταξινόμηση: Υπολόγισε τη γωνία μεταξύ κάθε πελάτη σε σχέση με την αποθήκη, ταξινομώντας τους πελάτες με βάση αυτές τις γωνίες.
    2. Σάρωση: Μία γραμμή σάρωσης περιστρέφεται γύρω από την αποθήκη, ξεκινώντας από κάποια τυχαία αρχική κατεύθυνση και σαρώνοντας με τη φορά των δεικτών του ρολογιού (ή αριστερόστροφα), συναντά πελάτες με ταξινομημένη σειρά.
    3. Σχηματισμός ομάδων (φάση 1): Καθώς η γραμμή σάρωσης συναντά πελάτες, τους εκχωρεί σε ομάδες. Όταν μία ομάδα υπερβαίνει τη χωρητικότητα του οχήματος, ξεκινά η δημιουργία της επόμενης ομάδας, δηλαδή των πελατών που θα εξυπηρετήσει το επόμενο όχημα. Ο αριθμός των ομάδων που θα προκύψουν τελικά αντιπροσωπεύει και τον αριθμό των οχημάτων που απαιτείται να χρησιμοποιηθούν.
    4. Κατασκευή διαδρομής (φάση 2): Χρήση ενός οποιουδήποτε κατασκευαστικού αλγορίθμου για τον καθορισμό της σειράς με την οποία θα εξυπηρετηθούν οι πελάτες σε κάθε διαδρομή.
    5. Έξοδος: Το αποτέλεσμα είναι μία αρχική λύση με ομάδες πελατών και σειρές εξυπηρέτησης για κάθε όχημα.



Εικόνα 3.3. *Απεικόνιση της περιστροφής της γραμμής σάρωσης*

- **Αλγόριθμοι CFRS (Cluster First, Route Second):** Όπως ακριβώς υποδηλώνει και το όνομα τους, αποτελούνται από μία φάση ομαδοποίησης των πελατών με χρήση κάποιου αλγορίθμου συσταδοποίησης και μία φάση δρομολόγησης με χρήση κάποιου κατασκευαστικού αλγορίθμου. Τα βήματα που ακολουθούνται είναι:
    1. Χρήση κάποιου αλγορίθμου ιεραρχικής (hierarchical clustering) ή διαχωριστικής (partitional clustering) συσταδοποίησης:

       Η ιεραρχική συσταδοποίηση συνήθως δεν επιβάλλει προκαθορισμένο αριθμό ομάδων και μπορεί να γίνει είτε συσσωρευτικά (agglomerative clustering) ξεκινώντας από μία ομάδα για κάθε πελάτη και ενώνοντας ομάδες επαναληπτικά βάσει κάποιου κριτηρίου (π.χ. ένωση κάθε φορά των δύο κοντινότερων ομάδων με βάση το μέσο όρο των συντεταγμένων των πελατών που ανήκουν σε αυτές), ελέγχοντας παράλληλα την μη παραβίαση περιορισμών, είτε διαιρετικά (divisive clustering) ξεκινώντας από μία ομάδα με όλους τους πελάτες και διαιρώντας επαναληπτικά βάσει κάποιου κριτηρίου μέχρι να μην υπάρχει ο ίδιος πελάτης σε πάνω από μία ομάδα.

       Η διαχωριστική συσταδοποίηση συνήθως απαιτεί από τον χρήστη να ορίσει εκ των προτέρων τον αριθμό των ομάδων, και αυτό μπορεί να γίνει με διάφορους τρόπους. Ένα από τα πιο δημοφιλή παραδείγματα αλγορίθμων διαχωριστικής συσταδοποίησης είναι ο αλγόριθμος K-means. Ο K-means ξεκινάει με την τυχαία επιλογή Κ, όπου Κ σταθερά ορισμένη από τον χρήστη, κέντρων ομάδων και εκχωρεί κάθε πελάτη στο πλησιέστερο κέντρο δημιουργώντας Κ ομάδες. Στη συνέχεια υπολογίζονται επαναληπτικά τα κέντρα των ομάδων ως ο μέσος όρος των συντεταγμένων των πελατών τους και γίνεται ξανά

εκχώρηση πελατών σε ομάδες (αλλαγή από ομάδα σε ομάδα) βάσει απόστασης από τα νέα κάθε φορά κέντρα. Η διαδικασία επαναλαμβάνεται έως ότου τα κέντρα συγκλίνουν οπότε και ο κάθε πελάτης οριστικοποιείται στην ομάδα όπου βρίσκεται.

2. Κατασκευή διαδρομών με χρήση κάποιου κατασκευαστικού αλγορίθμου στην κάθε ομάδα πελατών.

Γενικότερα, η επιλογή για το αν θα γίνει ή όχι χρήση αλγορίθμων δύο φάσεων και με ποιον τρόπο εξαρτάται από την φύση του προβλήματος προς επίλυση. Για ορισμένες παραλλαγές του ΠΔΟ, όπως το VRPTW, έχει αποδειχτεί στην πράξη ότι προκύπτουν μη ποιοτικές λύσεις σε περίπτωση που εφαρμοστεί άμεσα κάποιος αλγόριθμος που αδιαφορεί για την ύπαρξη των χρονικών παραθύρων εξυπηρέτησης. Ο πλησιέστερος γείτονας και οποιοσδήποτε κατασκευαστικός αλγόριθμος βασίζεται μόνο στην απόσταση αποτυγχάνει καταστροφικά με αποτέλεσμα να δημιουργούνται λύσεις με πολύ περισσότερα οχήματα από τα απαιτούμενα λόγω της συνεχούς παραβίασης των χρονικών παραθύρων κατά την ακολουθιακή κατασκευή της λύσης. Σε αυτή την περίπτωση προτείνεται η ομαδοποίηση των πελατών, όχι βάσει απόστασης, αλλά με τον διαχωρισμό «ελαστικών» και «πιεστικών» πελατών με βάση τον χρόνο κλεισίματος του χρονικού τους παραθύρου. Για παράδειγμα, αν υπάρχουν 30 πελάτες και ο τελευταίος κλείνει στις 15:00 μ.μ., χωρισμός των πελατών σε 4 ομάδες όπου στην ομάδα 1 ανήκουν όσοι κλείνουν μέχρι τις 9:00 π.μ., στην ομάδα 2 όσοι κλείνουν μέχρι τις 11:00 π.μ. κ.ο.κ. Ακολούθως, η εφαρμογή ενός κατασκευαστικού αλγορίθμου στις ομάδες αυτές θα οδηγήσει σε ένα πιο ποιοτικό σύνολο αρχικών λύσεων.

3.2.3. Ευρετικοί αλγόριθμοι βελτίωσης των διαδρομών (Improvement heuristics)

Πρόκειται για τεχνικές που χρησιμοποιούνται για τη βελτίωση των αρχικών λύσεων ενός ΠΔΟ μετά τη χρήση κάποιου κατασκευαστικού αλγορίθμου. Ουσιαστικά αναφερόμαστε σε **αλγόριθμους τοπικής αναζήτησης (local search heuristics)** που λειτουργούν ψάχνοντας τη βέλτιστη λύση μέσω της «δοκιμής και σφάλματος» (trial & error) μέσω μικρών τροποποιήσεων, και στην πράξη έχουν αποδειχθεί πολύ επιτυχημένοι σε έναν μεγάλο αριθμό προβλημάτων συνδυαστικής βελτιστοποίησης, συμπεριλαμβανομένου και του ΠΔΟ.

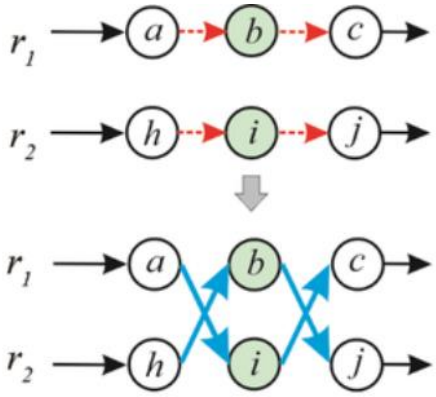
Μερικές κατηγορίες τέτοιων αλγορίθμων είναι οι αλγόριθμοι που βασίζονται σε ανταλλαγές στοιχείων, αλγόριθμοι επανατοποθέτησης στοιχείων, αλγόριθμοι επανασύνδεσης ακμών και αλγόριθμοι ελεγχόμενων διαταραχών. Η διαδικασία της τοπικής αναζήτησης γίνεται συνήθως

επαναληπτικά για κάθε πιθανό πελάτη ή συνδυασμό πελατών και για κάθε πιθανό συνδυασμό διαφορετικών διαδρομών. Στην περίπτωση που βρεθεί μία καλύτερη λύση με ανταλλαγή/επανατοποθέτηση/επανασύνδεση/διαταραχή, η νέα λύση δίνεται ξανά ως είσοδος στην επαναληπτική διαδικασία, έως ότου δεν μπορεί να βρεθεί κάποια καλύτερη λύση για οποιοδήποτε συνδυασμό.

3.2.3.1. Αλγόριθμοι βασισμένοι σε ανταλλαγές (Swap-based heuristics)

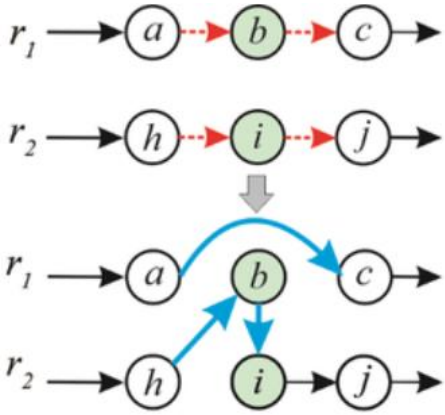
Οι αλγόριθμοι αυτοί πραγματοποιούν επαναληπτικά ανταλλαγή ενός ή περισσότερων πελατών μεταξύ διαφορετικών διαδρομών ή εντός της ίδιας διαδρομής. Έχουν προταθεί τέσσερεις τρόποι ανταλλαγής που μπορούν να εφαρμοστούν (Waters, 1987):

- **1-1 Ανταλλαγή (1-1 Exchange):** Ανταλλαγή δύο πελατών που βρίσκονται στην ίδια ή διαφορετική διαδρομή.



Εικόνα 3.4. *Παράδειγμα 1-1 ανταλλαγής (πελάτες b,i) μεταξύ δύο διαδρομών*

- **1-0 Ανταλλαγή (1-0 Exchange):** Διαγραφή ενός πελάτη από μία διαδρομή και τοποθέτηση σε κάποια τυχαία θέση άλλης διαδρομής.



Εικόνα 3.5. *Παράδειγμα 1-0 ανταλλαγής (πελάτης b) μεταξύ δύο διαδρομών*

- **1-2 Ανταλλαγή (1-2 Exchange):** Ανταλλαγή ενός πελάτη μίας διαδρομής με δύο πελάτες κάποιας άλλης διαδρομής.

- **2-2 Ανταλλαγή (2-2 Exchange):** Ανταλλαγή δύο πελατών μίας διαδρομής με δύο πελάτες κάποιας άλλης διαδρομής.

3.2.3.2. Αλγόριθμοι βασισμένοι σε επανατοποθέτηση (Relocate heuristics)

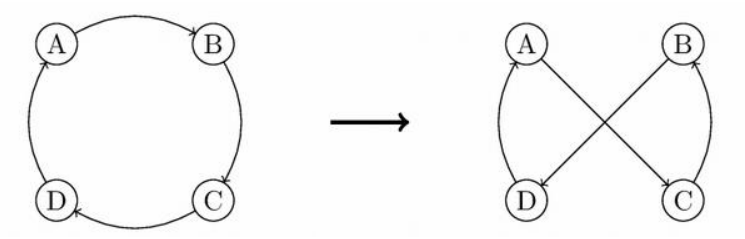
Οι αλγόριθμοι αυτοί πραγματοποιούν επαναληπτικά μεταφορά (ή επανατοποθέτηση) ενός ή περισσότερων πελατών μεταξύ διαφορετικών διαδρομών ή εντός της ίδιας διαδρομής.

- **Βέλτιστη επανατοποθέτηση (Best relocate):** Αξιολογούνται όλες τις πιθανές επιλογές μεταφοράς για έναν δεδομένο τυχαία επιλεγμένο πελάτη προς κάθε άλλη θέση κάθε διαδρομής που θα οδηγήσει σε εφικτή λύση, και κρατείται η λύση με την μεγαλύτερη συνολικά βελτίωση.
- **Τυχαία επανατοποθέτηση (Random relocate):** Επιλέγεται τυχαία ένας πελάτης και μεταφέρεται σε διαφορετική θέση στη λύση, σε μία τυχαία θέση, αρκεί η λύση που θα προκύψει να είναι εφικτή και όχι απαραίτητα καλύτερη. Ο λόγος που υπάρχει ο τελεστής αυτός αφορά την εξερεύνηση του χώρου λύσεων, εισάγοντας τυχαιότητα και με μικρότερο υπολογιστικό κόστος συγκριτικά με τη χρήση του «Best relocate».
- **Άπληστη επανατοποθέτηση (Greedy relocate):** Αξιολογούνται όλες τις πιθανές επιλογές μεταφοράς για κάθε πελάτη και προς κάθε άλλη θέση κάθε διαδρομής που θα οδηγήσει σε εφικτή λύση, και κρατείται σε κάθε επανάληψη η λύση με την μεγαλύτερη συνολικά βελτίωση.

3.2.3.3. Αλγόριθμοι διαγραφής και επανασύνδεσης ακμών

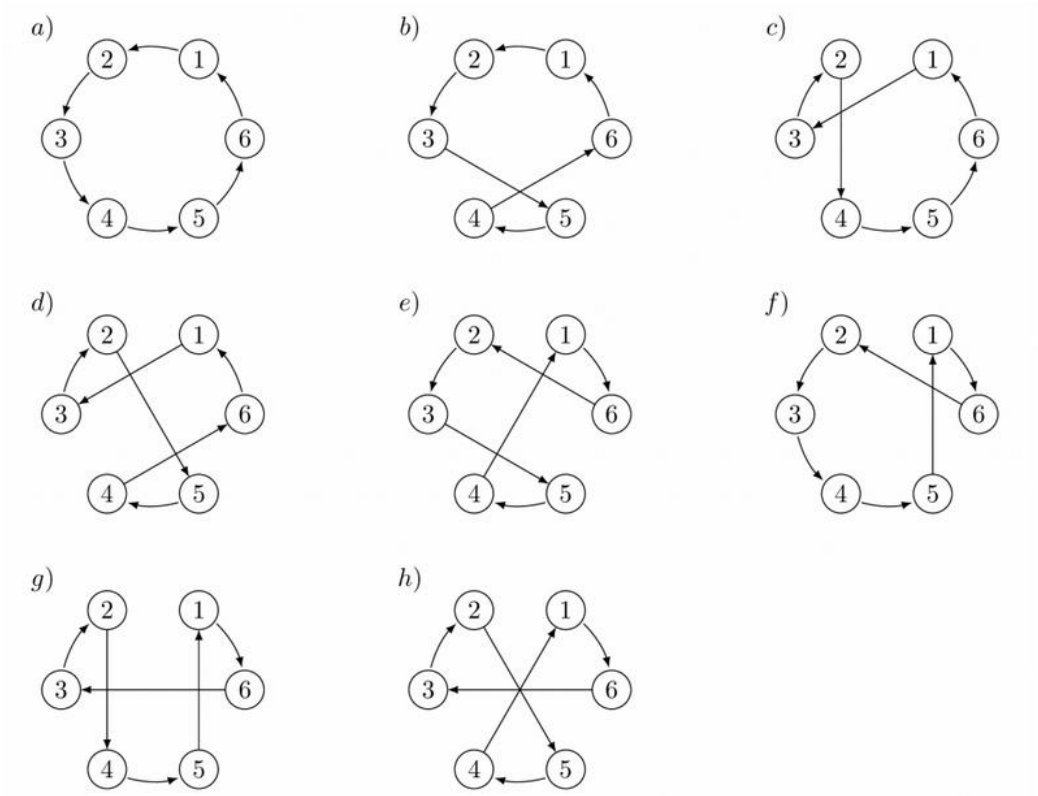
Οι αλγόριθμοι αυτοί πραγματοποιούν επαναληπτικά αφαίρεση ορισμένων ακμών από μια διαδρομή και στη συνέχεια την επανασύνδεσή τους με διαφορετική σειρά με στόχο την διατήρηση της εφικτότητας και την πιθανή βελτίωση του συνολικού κόστους της διαδρομής, άρα και συνολικά της λύσης. Η διαδικασία γίνεται κι εδώ επαναληπτικά για κάθε συνδυασμό k ακμών που αφαιρούνται και επανασυνδέονται, και κάθε καλύτερη διαδρομή που βρίσκεται δίνεται ξανά ως είσοδος στον αλγόριθμο, μέχρις ότου δεν μπορεί να βελτιωθεί περαιτέρω.

- **2-opt:** Διαγραφή και επανασύνδεση δύο ακμών με διαφορετικό τρόπο. Για κάθε δυάδα ακμών υπάρχει μόνο ένας εφικτός τρόπος επανασύνδεσης ώστε να μην προκύψουν δύο κύκλοι. Οι δύο ακμές αυτές μπορεί να επιλεγούν με διάφορους τρόπους, όπως για παράδειγμα με την διαγραφή των δύο χειρότερων βάσει απόστασης ακμών, ή και τυχαία.



Εικόνα 3.6. *Παράδειγμα εφαρμογής 2-opt με διαγραφή των ακμών AB και CD*

- **3-opt:** Διαγραφή και επανασύνδεση τριών ακμών με διαφορετικό τρόπο. Για κάθε τριάδα ακμών υπάρχουν 7 διαφορετικοί τρόποι επανασύνδεσης (εκ των οποίων οι τρεις αφορούν κινήσεις 2-opt).

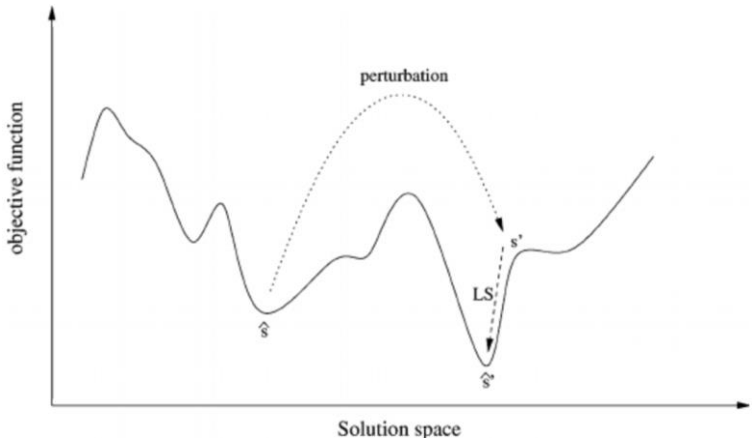


Εικόνα 3.6. *Όλες οι πιθανές 3-opt κινήσεις με αφαίρεση των ακμών 1→2, 3→4, 5→6*

3.2.3.4. Αλγόριθμοι διαταραχής (Perturbation heuristics)

Σκοπός των αλγορίθμων αυτών είναι η εισαγωγή ελεγχόμενων διαταραχών ή αλλαγών στην τρέχουσα λύση με στόχο τη αποφυγή τοπικών βέλτιστων, την εξερεύνηση εναλλακτικών περιοχών στον χώρο λύσεων και τελικά τη βελτίωση της ποιότητας της λύσης. Η βασική διαφορά με τις προηγούμενες κατηγορίες είναι ότι δεν είναι απαραίτητο να προκύψει καλύτερη λύση για να γίνει αποδεκτή, αλλά αρκεί το να είναι εφικτή. Στη συνέχεια με την εφαρμογή αλγορίθμων τοπικής αναζήτησης των προηγούμενων κατηγοριών επιδιώκεται η βελτίωση της διαταραγμένης λύσης επαναληπτικά μέχρι να φτάσει ξανά σε τοπικό βέλτιστο.

- **Ανακίνηση διαδρομής (Route shaking):** Διαταραχή που αφορά το πλήρες τυχαίο ανακάτεμα της σειράς εξυπηρέτησης των πελατών.
- **Ανακίνηση λύσης (Solution shaking):** Ισχυρή διαταραχή που αφορά πολλαπλές ανακινήσεις διαδρομών, τυχαίες επανατοποθετήσεις κ.α.



Εικόνα 3.7. *Χρήση διαταραχής για την διαφυγή από τοπικό ελάχιστο*

3.3. Μεθευρετικοί αλγόριθμοι (Metaheuristics)

Οι μεθευρετικοί αλγόριθμοι (ΜΑ) είναι ισχυρές μεθοδολογίες επίλυσης προβλημάτων που χρησιμοποιούνται για την αντιμετώπιση σύνθετων προβλημάτων βελτιστοποίησης όπου οι παραδοσιακοί αλγόριθμοι αδυνατούν λόγω της υπολογιστικής τους πολυπλοκότητας. Σε αντίθεση με τους ακριβείς αλγόριθμους (exact algorithms), οι οποίοι στοχεύουν στην εύρεση βέλτιστων λύσεων, οι αλγόριθμοι αυτοί δίνουν προτεραιότητα στην αποτελεσματικότητα και την επεκτασιμότητα εξερευνώντας τον χώρο των λύσεων με ευρετικό τρόπο. Λειτουργούν με επαναληπτική πλοήγηση σε έναν τεράστιο χώρο αναζήτησης, καθοδηγούμενη από ένα συνδυασμό στρατηγικών εκμετάλλευσης και εξερεύνησης.

Η **εκμετάλλευση (exploitation)** αφορά την εστίαση στην τοπική αναζήτηση βασιζόμενη στην πληροφορία ότι έχει βρεθεί μια καλή λύση σε κάποια περιοχή. Αντιθέτως, η **εξερεύνηση (exploration)** αναφέρεται στην εξερεύνηση του χώρου των λύσεων, εξασφαλίζοντας ότι δεν θα παγιδευτεί ο αλγόριθμος σε κάποιο τοπικό βέλτιστο, προς αναζήτηση μίας νέας βέλτιστης λύσης. Με τη σωστή αναλογία των δύο παραπάνω στοιχείων και ενσωματώνοντας προσαρμοστικούς μηχανισμούς, μπορεί να εξασφαλιστεί μία λύση πολύ κοντά στο ολικό βέλτιστο.

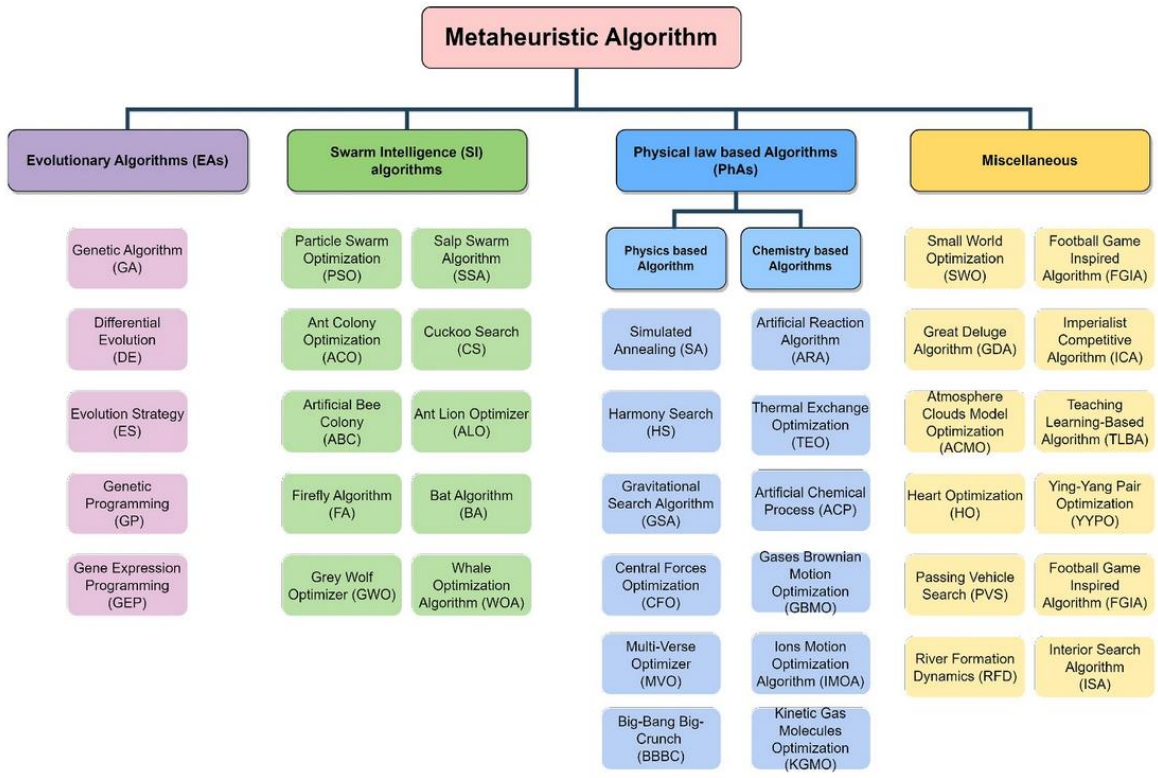
3.3.1. Ταξινόμηση των μεθευρετικών αλγορίθμων (Metaheuristics taxonomy)

Οι μεθευρετικοί αλγόριθμοι μπορούν να κατηγοριοποιηθούν σε ομάδες με πολλούς τρόπους με βάση τις προαναφερθέντες στρατηγικές (εκμετάλλευση, εξερεύνηση) και τα

χαρακτηριστικά της αναζήτησης. Τέσσερεις διαδεδομένοι τρόποι κατηγοριοποίησης των MA (Kanchan Rajwar et al., 2023), των οποίων η ανάλυση ακολουθεί παρακάτω, έχουν προταθεί από διάφορους ερευνητές.

3.3.1.1. Ταξινόμηση με βάση την πηγή έμπνευσης

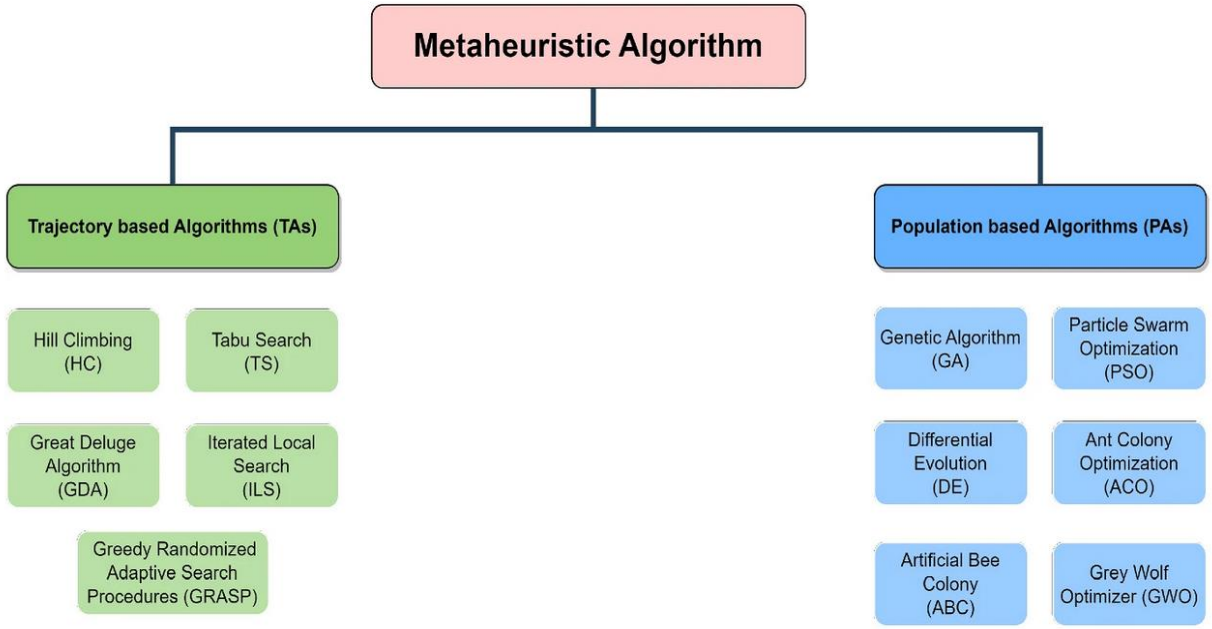
Αποτελεί τον παλαιότερο τρόπο ταξινόμησης. Είναι μια εύκολη στην κατανόηση ταξινόμηση επειδή οι εμπνευσμένοι από τη φύση αλγόριθμοι και η έννοια των μεθευρετικών προσεγγίσεων βασίζεται κυρίως σε φυσικά ή βιολογικά φαινόμενα. Ανάλογα με την πηγή έμπνευσης, οι MA έχουν κατηγοριοποιηθεί με διάφορους τρόπους από διαφορετικούς συγγραφείς. Οι Fister Jr et al. (2013) τους έχουν ταξινομήσει σε τέσσερις κατηγορίες ως αλγόριθμους βασισμένους σε ευφυή σμήνη (swam intelligence), εξελικτικούς (evolutionary) ή βιο-εμπνευσμένους αλγόριθμους (bio-inspired), αλγόριθμους βασισμένους στη φυσική-χημεία και τους υπόλοιπους ως «άλλο», ενώ οι Siddique και Adeli (2015) τους έχουν χωρίσει σε τρεις υπο-ομάδες ως αλγόριθμους που βασίζονται στη φυσική, στη χημεία και στη βιολογία. Οι Molina et al. (2020) τους έχουν ταξινομήσει σε έξι υπο-ομάδες ως εξελικτικούς αλγόριθμους βασισμένους στην αναπαραγωγή (breeding based), αλγόριθμους βασισμένους σε ευφυή σμήνη, αλγόριθμους βασισμένους στη φυσική-χημεία, αλγόριθμους βασισμένους στην κοινωνική συμπεριφορά του ανθρώπου (human social behavior based) και αλγόριθμους βασισμένους σε φυτά (plant based), ενώ οι υπόλοιποι αναφέρονται ως «άλλο».



Εικόνα 3.8. *Ταξινόμηση μεθευρετικών αλγορίθμων με βάση την πηγή έμπνευσης*

3.3.1.2. Ταξινόμηση με βάση το μέγεθος του πληθυσμού

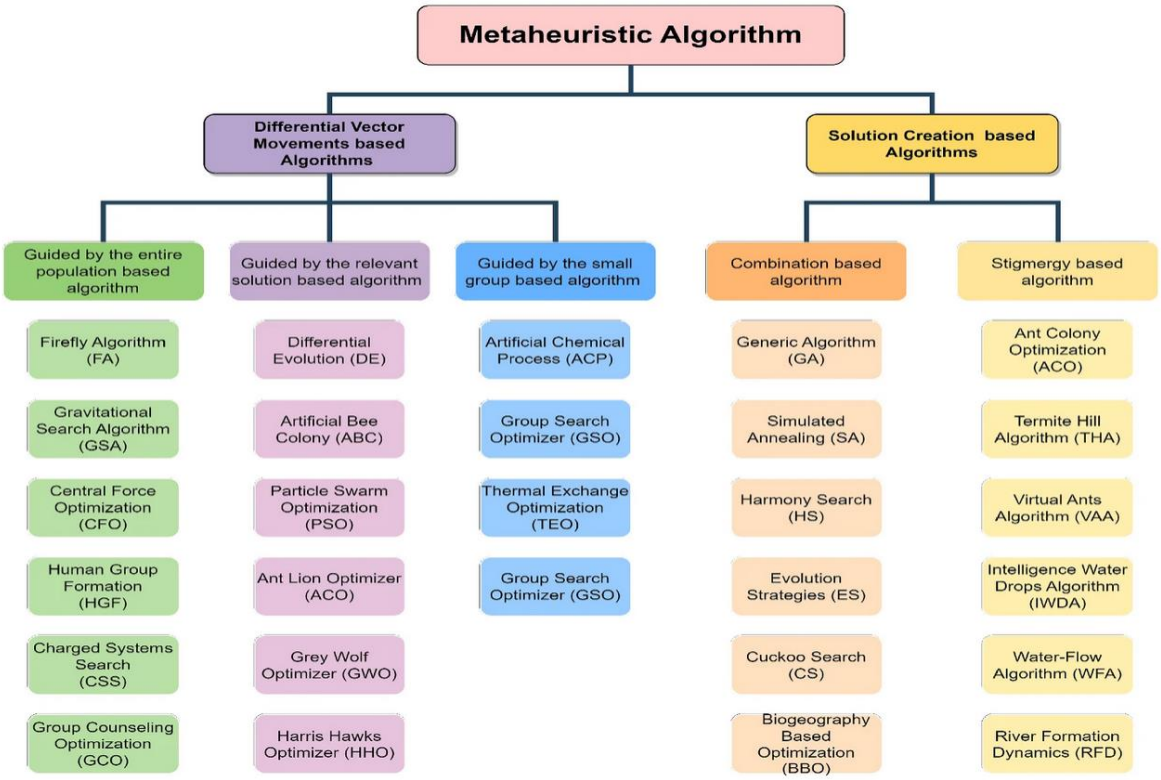
Πολλαπλοί πράκτορες λειτουργούν καλύτερα μαζί από έναν μεμονωμένο πράκτορα., καθώς υπάρχουν πολλά πλεονεκτήματα, όπως κοινή χρήση πληροφοριών, απομνημόνευση δεδομένων κ.α. Εμπνευσμένοι από αυτό, οι ερευνητές προσπαθούν να ανακαλύψουν την βέλτιστη λύση με πολλούς πράκτορες που την αναζητούν ταυτόχρονα και σε πολλές περιπτώσεις συνεργατικά. Οι υπάρχοντες αλγόριθμοι ταξινομούνται σε δύο κατηγορίες (Yang, 2020) ως αλγόριθμους που βασίζονται στην τροχιά (trajectory based) και σε αλγόριθμους που χρησιμοποιούν πληθυσμούς (population based).



Εικόνα 3.9. *Ταξινόμηση μεθευρετικών αλγορίθμων με βάση το μέγεθος του πληθυσμού*

3.3.1.3. Ταξινόμηση με βάση την κίνηση του πληθυσμού

Οι Molina et al. (2020) προσπάθησαν να κατηγοριοποιήσουν τους ΜΑ με βάση τη συμπεριφορά τους και όχι την πηγή έμπνευσής του. Το βασικό χαρακτηριστικό αυτής της ταξινόμησης είναι ο τρόπος με τον οποίο ενημερώνεται ο πληθυσμός σε κάθε επόμενη επανάληψη. Η ταξινόμηση αυτή βοηθάει στην κατανόηση των αλγορίθμων του ίδιου τύπου. Έτσι, οι ΜΑ μπορούν να ταξινομηθούν σε δύο κατηγορίες, ως αλγόριθμοι που βασίζονται στη διαφορική διανυσματική κίνηση (differential vector movements based) και σε αλγόριθμους που βασίζονται στη δημιουργία λύσεων (solution creation based).

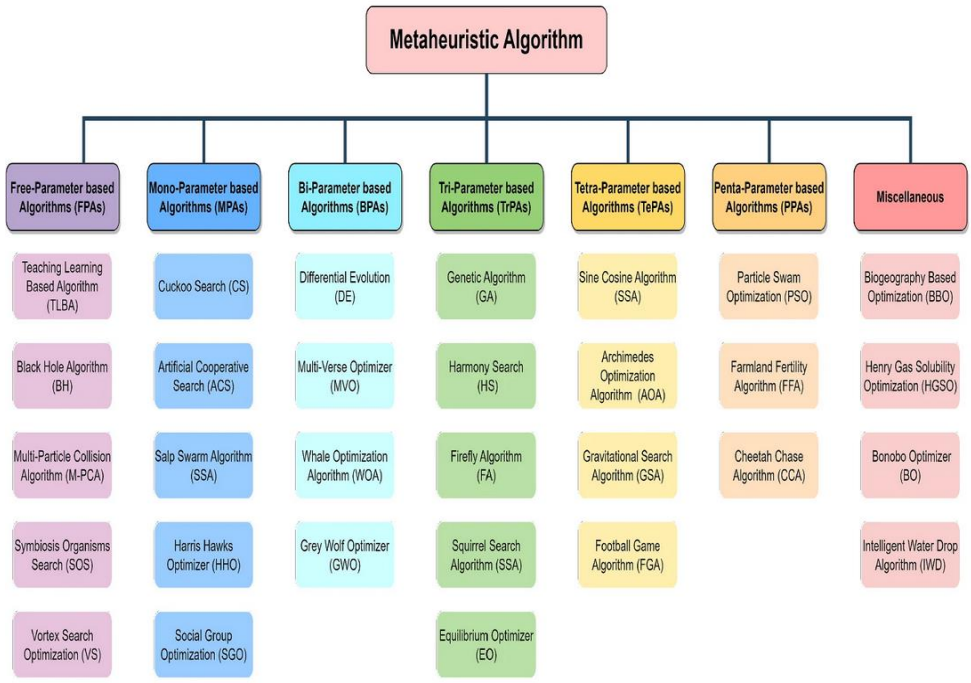


Εικόνα 3.10. *Ταξινόμηση μεθευρετικών αλγορίθμων με βάση την κίνηση του πληθυσμού*

3.3.1.3. Ταξινόμηση με βάση τον αριθμό των παραμέτρων

Η απόδοση των ΜΑ εξαρτάται σε μεγάλο βαθμό από τις ρυθμίσεις των παραμέτρων τους και οι «ιδανικές» τιμές αυτών εξαρτώνται κάθε φορά από το πρόβλημα που επιλύεται. Η επιλογή των καλύτερων τιμών των παραμέτρων (parameter tuning) είναι ένα περίπλοκο πρόβλημα που μπορεί να χρειάζεται τη δική του περιοχή μελέτης (Talbi, 2009). Ένας μικρότερος αριθμος παραμέτρων απλοποιεί την ρύθμιση αυτών, ενώ ένας μεγαλύτερος αριθμός επηρεάζει αισθητά την πολυπλοκότητα ενός αλγορίθμου.

Απαιτείται λοιπόν ένας ακόμη τρόπος ταξινόμησης για τον εντοπισμό αλγορίθμων που χρησιμοποιούν τον ίδιο αριθμό παραμέτρων, με σκοπό την απόκτηση μιας πρόσθετης μαθηματικής κατανόησης των ΜΑ. Η παρακάτω ταξινόμηση βασίζεται στον αριθμό των λεγόμενων «κύριων» παραμέτρων (primary parameters) που χαρακτηρίζουν κάθε διαφορετικό αλγόριθμο. Δευτερεύουσες παράμετροι (secondary parameters) όπως το μέγεθος του πληθυσμού ή ο αριθμός των επαναλήψεων, παρόλο που έχουν πολύ σημαντικό αντίκτυπο στην ποιότητα της λύσης που τελικά εξάγεται, δεν λαμβάνονται υπόψη καθώς μοιράζονται σε όλες τις αλγοριθμικές προσεγγίσεις. Έτσι, οι ΜΑ μπορούν να ταξινομηθούν σε έξι κατηγορίες, ως αλγόριθμοι που βασίζονται σε 0, 1, 2, 3, 4 ή 5 κύριες παραμέτρους και στην κατηγορία «άλλο» για όσους βασίζονται σε πάνω από 5.



Εικόνα 3.11. *Ταξινόμηση μεθευρετικών αλγορίθμων με βάση τον αριθμό των παραμέτρων*

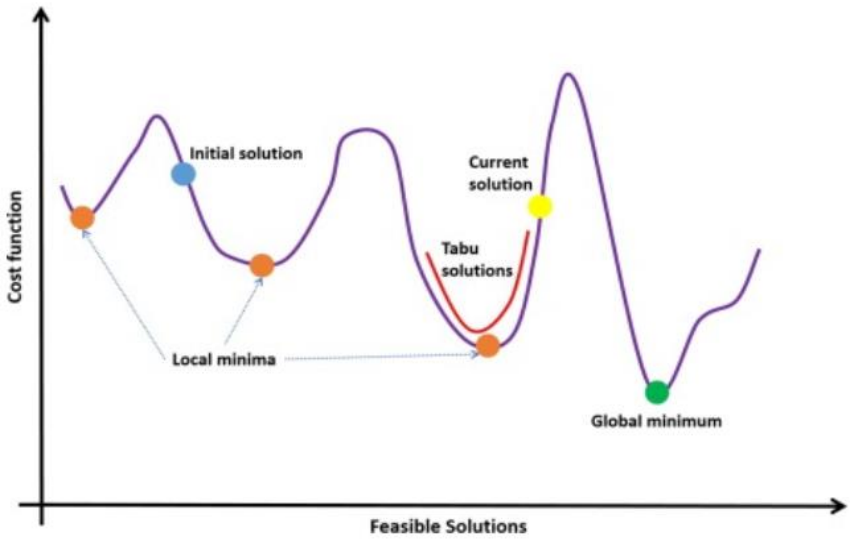
3.3.2. Περιορισμένη αναζήτηση (Tabu search)

Η περιορισμένη αναζήτηση (TS) είναι αλγόριθμος εμπνευσμένος από την έννοια της μνήμης των ανθρώπων κατά την λήψη αποφάσεων. Βασίζεται σε μία λογική αναζήτησης γειτονικών λύσεων χρησιμοποιώντας μια **λίστα «Tabu»** για να αποφεύγει την επανάληψη κινήσεων-λύσεων που έχουν ήδη εξεταστεί. Αυτό τον βοηθάει να ξεφεύγει από τοπικά βέλτιστα, αφού δεν μπορεί να επιστρέψει σε προηγούμενες λύσεις που περιέχονται στη λίστα Tabu, για αριθμό επαναλήψεων όσο και το μέγεθός της, και να εντοπίσει ίσως ένα καλύτερο τοπικό βέλτιστο που βρίσκεται πιο «μακριά» στο χώρο των λύσεων. Αναλυτικά, τα βήματα του αλγορίθμου είναι:

1. Ορισμός του **μεγέθους *T* της λίστας Tabu**, του **κριτηρίου τερματισμού** (αριθμός επαναλήψεων/ εύρεση λύσης με επιθυμητή ποιότητα/ χρόνος «τρεξίματος» του προγράμματος), και ενός **κριτηρίου φιλοδοξίας (aspiration criterion)** το οποίο επιτρέπει την αγνόηση της λίστας tabu σε περίπτωση που βρεθεί μία πολύ καλύτερη λύση συγκριτικά με την τρέχουσα.
2. **Εκκίνηση με μία αρχική εφικτή λύση** η οποία μπορεί να δημιουργηθεί είτε τυχαία είτε χρησιμοποιώντας κάποια ευρετική κατασκευαστική μέθοδο.
3. **Ορισμός της γειτονιάς γύρω από την τρέχουσα λύση**. Αυτή η γειτονιά αντιπροσωπεύει όλες τις πιθανές λύσεις που μπορούν να προκύψουν από την τρέχουσα λύση κάνοντας μικρές αλλαγές. Αυτές οι αλλαγές καθορίζονται από τον

χρήστη, απαιτούν πειραματισμό και θα μπορούσαν να είναι εναλλαγή στοιχείων, αφαίρεση ή η προσθήκη στοιχείων ή οποιαδήποτε άλλη σχετική τροποποίηση ανάλογα με τη φύση του προβλήματος. Πρέπει να τονιστεί ότι σε μεγάλης κλίμακας ΠΔΟ είναι αδύνατον να δοκιμαστούν όλες οι εναλλαγές ή αφαιρέσεις/προσθήκες στοιχείων (πελατών) μεταξύ των διαδρομών λόγω των αμέτρητων πολλές φορές συνδυασμών, οπότε επιλέγεται ένας αριθμός γειτονικών λύσεων που θα δημιουργούνται με τυχαιότητα σε κάθε επανάληψη.

4. **Αξιολόγηση όλων των λύσεων της γειτονιάς**, υπολογίζοντας την τιμή της αντικειμενικής συνάρτησης (fitness score) για κάθε λύση.
5. **Επιλογή της καλύτερης κίνησης-λύσης** από τη γειτονιά που δεν απαγορεύεται από τη λίστα Tabu (ακόμη κι αν είναι χειρότερη από την τρέχουσα) ή πληροί το κριτήριο φιλοδοξίας. Η καλύτερη κίνηση καθορίζεται τυπικά με βάση την τιμή της αντικειμενικής συνάρτησης ή κάποιες φορές και από άλλα κριτήρια (π.χ. υπολογιστικό κόστος κίνησης) ειδικά για το κάθε πρόβλημα.
6. **Προσθήκη της καλύτερης λύσης στη λίστα tabu** και ορισμός αυτής ως τρέχουσα λύση. Αν η λίστα tabu περιέχει ήδη T προηγούμενες λύσεις, τότε η λύση που προστέθηκε πρώτη αφαιρείται από τη λίστα (λογική: first in, first out) επιτρέποντας στον αλγόριθμο να την ξαναεπισκεφθεί.
7. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 9.
8. Επανάληψη των βημάτων 3-7 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
9. **Επιστροφή της καλύτερης λύσης που βρέθηκε** κατά τη διαδικασία της αναζήτησης.



Εικόνα 3.12. *Απεικόνιση της λειτουργίας του αλγορίθμου TS*

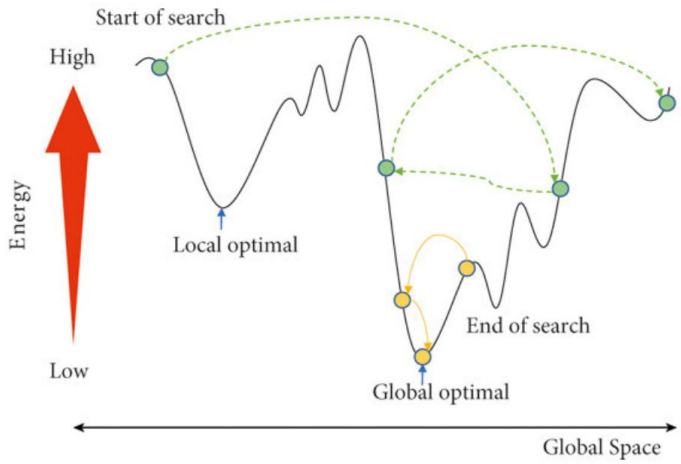
3.3.3. Προσομοιωμένη ανόπτηση (Simulated annealing)

Η προσομοιωμένη ανόπτηση (SA) είναι ένας πιθανοτικός αλγόριθμος βελτιστοποίησης εμπνευσμένος από τη διαδικασία ανόπτησης στη μεταλλουργία. **Ανόπτηση** είναι η διαδικασία θέρμανσης του μετάλλου και στην συνέχεια η σταδιακή ψύξη του με **ελεγχόμενο ρυθμό ψύξης**, έτσι ώστε το μέταλλο να αποκτήσει την ιδανική αναλογία των χαρακτηριστικών της σκληρότητας και ευθραυστότητας. Η ανόπτηση επιδρά στην κρυσταλλική δομή του μετάλλου και αποδίδει στο μέταλλο τις επιθυμητές ιδιότητες. Το επιθυμητό αποτέλεσμα δεν επιτυγχάνεται σε περίπτωση που η ψύξη γίνει με πιο ταχύ ή πιο αργό ρυθμό από αυτόν ακριβώς που πρέπει.

Ο αλγόριθμος της προσομοιωμένης ανόπτησης προτάθηκε από τους Kirkpatrick, Gelatt Jr. και Vecchi, το 1983. Προσομοιώνει αυτήν ακριβώς αυτή τη διαδικασία της ανόπτησης, όπως δηλώνει και το όνομα του. Η διαδικασία της ανόπτησης ξεκινά με μία υψηλή «θερμοκρασία» (μεγάλη πιθανότητα αποδοχής χειρότερων λύσεων) και σταδιακά η θερμοκρασία μειώνεται (μείωση της πιθανότητας αποδοχής χειρότερων λύσεων) με την πάροδο του χρόνου (επαναλήψεων). Σκοπός είναι η παρεμπόδιση της πρόωρης σύγκλισης αλλά και η διαφυγή από τοπικά βέλτιστα. Αναλυτικά, τα βήματα του αλγορίθμου είναι:

1. Ορισμός της **αρχικής θερμοκρασίας T_0** και ορισμός της **διαδικασίας «ψύξης»**, που μπορεί να είναι εκθετική, γραμμική ή οποιαδήποτε άλλη παραλλαγή.
2. **Ορισμός του κριτηρίου τερματισμού**. Συνήθως, η διαδικασία τερματίζεται όταν η θερμοκρασία πέσει κάτω από κάποιο όριο ή μετά από συγκεκριμένο αριθμό επαναλήψεων.
3. **Αρχικοποίηση με μία εφικτή λύση x** (αρχικά τρέχουσα λύση που δημιουργήθηκε τυχαία ή με κάποια ευρετική κατασκευαστική μέθοδο) και υπολογισμός της τιμής της αντικειμενικής συνάρτησης $f(x)$.
4. **Δημιουργία μίας νέας γειτονικής λύσης x'**, προερχόμενη από την τρέχουσα λύση με μικρές τυχαίες τροποποιήσεις που μπορεί να είναι εναλλαγή στοιχείων, αφαίρεση ή η προσθήκη στοιχείων, διαταραχή της τρέχουσας λύσης κ.α.
5. Αν η νέα λύση είναι εφικτή, υπολογισμός της διαφοράς της τιμής της αντικειμενικής συνάρτησης $\Delta E = f(x') - f(x)$, όπου f η αντικειμενική συνάρτηση προς ελαχιστοποίηση. Αν η λύση δεν είναι εφικτή, επιστροφή στο βήμα 4.
6. Αν $\Delta E \leq 0$, αποδοχή της νέας λύσης x' ως τρέχουσα, αλλιώς αν $\Delta E > 0$, αποδοχή της νέας λύσης x' ως τρέχουσα με πιθανότητα $e^{-\Delta E / T}$.

7. **Μείωση της θερμοκρασίας σύμφωνα με την διαδικασίας ψύξης**. Για εκθετική μείωση θα ισχύει $T_{k+1} = \alpha \cdot T_k$ (όπου $\alpha<1$), για γραμμική ψύξη θα ισχύει $T_{k+1} = T_k - \beta$ (όπου β>0), ή οποιαδήποτε άλλη παραλλαγή χρησιμοποιηθεί.
8. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 10.
9. Επανάληψη των βημάτων 4-8 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
10. **Επιστροφή της καλύτερης λύσης που βρέθηκε** κατά τη διαδικασία της αναζήτησης.



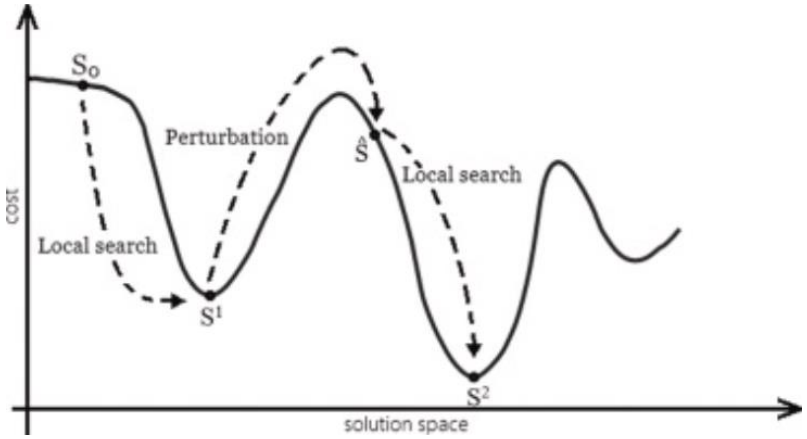
Εικόνα 3.13. *Απεικόνιση της λειτουργίας του αλγορίθμου SA*

3.3.4. Επαναληπτική τοπική αναζήτηση (Iterated local search)

Η επαναλαμβανόμενη τοπική αναζήτηση (ILS) είναι μεθευρετικός αλγόριθμος βελτιστοποίησης που ανήκει στην ευρύτερη κατηγορία των αλγορίθμων τοπικής αναζήτησης. Η βασική ιδέα πίσω από τον ILS είναι η **επαναληπτική βελτίωση μιας λύσης** εφαρμόζοντας έναν αλγόριθμο **τοπικής αναζήτησης** και **διαταράσσοντας τη λύση (perturbation)** για την εξερεύνηση διαφορετικών περιοχών στο χώρο των λύσεων. Ο αλγόριθμος στοχεύει στο να μην εγκλωβιστεί σε κάποιο από τα τοπικά βέλτιστα αλλά να βρει καλύτερες λύσεις εξερευνώντας επαναληπτικά τον χώρο λύσεων εφαρμόζοντας ισχυρές διαταραχές. Αναλυτικά, τα βήματα του αλγορίθμου είναι:

1. **Δημιουργία μίας αρχικής εφικτής λύσης *x*** είτε με χρήση κάποιου ευρετικού κατασκευαστικού αλγορίθμου είτε τυχαία.
2. **Ορισμός του κριτηρίου τερματισμού**, που συνήθως είναι ο αριθμός των επαναλήψεων ή η εύρεση κάποιας συγκεκριμένης ποιότητας λύσης.
3. Επιλογή και εφαρμογή ενός ή περισσότερων αλγορίθμων **τοπικής αναζήτησης** στην τρέχουσα λύση *x* με σκοπό την εύρεση ενός τοπικού βέλτιστου x_{local}.

4. **Εφαρμογή μίας ισχυρής διαταραχής στην τοπικά βέλτιστη λύση** με σκοπό τη διαφυγή από τη γειτονιά των λύσεων και τη δημιουργία μίας νέας λύσης $x_{perturbed}$. Ισχυρές διαταραχές θα μπορούσαν να είναι οι ευρετικές μέθοδοι 2-2 exchange, random relocate, route shaking, solution shaking οι οποίες θα εφαρμόζονται κάθε μία με κάποια πιθανότητα, ή και όλες μαζί σε κάθε επανάληψη.
5. Εφαρμογή ενός ή περισσότερων αλγορίθμων **τοπικής αναζήτησης** (για δεύτερη φορά) στην διαταραγμένη λύση $x_{perturbed}$ με σκοπό την εύρεση ενός νέου τοπικού βέλτιστου x_{new}.
6. **Αξιολόγηση της λύσης x_{new}**. Έστω f η αντικειμενική συνάρτηση προς ελαχιστοποίηση. Αν $f(x_{new})<f(x)$ τότε ορισμός της x_{new} ως τρέχουσα.
7. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 9.
8. Επανάληψη των βημάτων 3-7 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
9. **Επιστροφή της καλύτερης λύσης που βρέθηκε** κατά τη διαδικασία της αναζήτησης.



Εικόνα 3.13. *Απεικόνιση της λειτουργίας του αλγορίθμου ILS*

3.3.5. Γενετικοί αλγόριθμοι (Genetic algorithms)

Οι γενετικοί αλγόριθμοι (GA) είναι μια κατηγορία εξελικτικών αλγορίθμων βελτιστοποίησης που εμπνέονται από τη Δαρβινική θεωρία της φυσικής επιλογής και βιολογικής εξέλιξης. Λειτουργούν διατηρώντας έναν **πληθυσμό υποψήφιων λύσεων** (άτομα), εξελίσσοντάς τες σε διαδοχικές **γενιές (επαναλήψεις)** μέσω διεργασιών **επιλογής γονέων, διασταύρωσης και μετάλλαξης.** Σκοπός είναι να προωθηθούν οι καλύτερες λύσεις στις επόμενες γενιές και ταυτόχρονα να αποτραπεί η γρήγορη σύγκλιση, διατηρώντας την ποικιλομορφία των ατόμων στον πληθυσμό.

Οι λύσεις παρουσιάζονται κωδικοποιημένες σαν χρωμοσώματα αποτελούμενα από γονίδια, με τις μεταβλητές απόφασης σε τέτοια μορφή ώστε να μπορεί να γίνει αποκωδικοποίηση και

χρήση τους από την αντικειμενική συνάρτηση. Αν και κάθε χρωμόσωμα μπορεί να περιέχει και τιμές πραγματικών αριθμών, ακέραιων και μη, η λύση συνηθίζεται να κωδικοποιείται στο δυαδικό σύστημα. Το χρωμόσωμα όμως έχει νόημα μονάχα στην αποκωδικοποιημένη του μορφή και η αξιολόγηση κάθε λύσης από την αντικειμενική συνάρτηση προκύπτει ως συνδυασμός των χρωμοσωμάτων αυτών.

Αναλυτικά, τα βήματα που ακολουθούνται τυπικά σε έναν γενετικό αλγόριθμο είναι:

1. **Ορισμός των παραμέτρων**: Μέγεθος πληθυσμού, τεχνική επιλογής γονέων, μηχανισμός διασταύρωσης, μηχανισμός μετάλλαξης, πιθανότητα μετάλλαξης.
2. **Ορισμός του κριτηρίου τερματισμού**, που μπορεί να είναι ο αριθμός γενεών (επαναλήψεων), κριτήρια σύγκλισης (βάσει της τυπικής απόκλισης και της μέσης τιμής των τιμών καταλληλότητας (fitness scores) του τρέχοντος πληθυσμού ή όταν δεν παρατηρείται βελτίωση των λύσεων από γενιά σε γενιά για κάποιον αριθμό επαναλήψεων) ή κριτήρια υπολογιστικών πόρων και χρόνου.
3. **Δημιουργία του αρχικού πληθυσμού** εφικτών λύσεων, με κάποια ευρετική κατασκευαστική μέθοδο ή με τυχαιότητα.
4. **Αξιολόγηση όλων των λύσεων του πληθυσμού** από την αντικειμενική συνάρτηση και επιλογή γονέων για τη δημιουργία της δεξαμενής ζευγαρώματος (mating pool).
5. **Διασταύρωση** γονέων ανά ζεύγη για δημιουργία απογόνων.
6. **Μετάλλαξη** ενός ποσοστού των απογόνων με χρήση της επιλεγμένης τεχνικής μετάλλαξης και βάσει της πιθανότητας μετάλλαξης.
7. **Αντικατάσταση του τρέχοντος πληθυσμού** με τον πληθυσμό των απογόνων ή/και γονέων.
8. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 10.
9. Επανάληψη των βημάτων 4-8 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
10. **Επιστροφή της καλύτερης λύση**ς του τελευταίου πληθυσμού ή από όλους τους πληθυσμούς που δημιουργήθηκαν.

3.3.5.1. Τεχνικές επιλογής γονέων (Parent selection)

Κάθε επανάληψη ενός GA ξεκινάει με την επιλογή από το σύνολο του πληθυσμού των γονέων που θα διασταυρωθούν. Μερικές συνήθεις τεχνικές επιλογής γονέων παρουσιάζονται παρακάτω:

- **Τεχνική της ρουλέτας (Roulette wheel selection):** Η πιθανότητα επιλογής μίας λύσης *i* ως γονέα καθορίζεται από την τιμή της αντικειμενικής συνάρτησης (έστω μεγιστοποίησης) *f(i) = Fitness(i)*. Τα άτομα-λύσεις με μεγαλύτερο fitness score έχουν

μεγαλύτερη πιθανότητα να επιλεγούν. Έτσι, η πιθανότητα επιλογής μίας λύσης *i* δίνεται από τη σχέση: $P(i) = \frac{Fitness(i)}{\sum_{j=1}^{N} Fitness(j)}$, όπου *N* το μέγεθος του πληθυσμού.

Αλγοριθμικά η διαδικασία που ακολουθείται είναι η παρακάτω:

1. Αξιολόγηση όλων των λύσεων του πληθυσμού και υπολογισμός του $S = \sum_{j=1}^{N} Fitness(j)$.
2. Επιλογή ενός τυχαίου αριθμού *R* στο διάστημα [0, *S*].
3. Άθροισμα των τιμών των *Fitness(j)* για κάθε λύση *j*, μία προς μία, με έλεγχο κάθε φορά και έως ότου η σχέση $\sum_{j=1}^{K} Fitness(j) \geq R$ ικανοποιηθεί.
4. Επιλογή της λύσης *K* ως γονέα.
5. Επανάληψη των βημάτων 2-4 έως ότου ληφθεί αριθμός γονέων ίσος με το μέγεθος του πληθυσμού.

- **Επιλογή διαγωνισμού (Tournament selection):** Η τεχνική αυτή διακρίνεται για την ικανότητα διατήρησης της ποικιλομορφίας στον πληθυσμό, και ακολουθεί 3 απλά βήματα:

  1. Επιλογή *N* τυχαίων λύσεων από τον πληθυσμό.
  2. Από το δείγμα των *N* λύσεων, επιλογή αυτής με το υψηλότερο fitness score.
  3. Επανάληψη των βημάτων 1-2 έως ότου ληφθεί αριθμός γονέων ίσος με το μέγεθος του πληθυσμού.

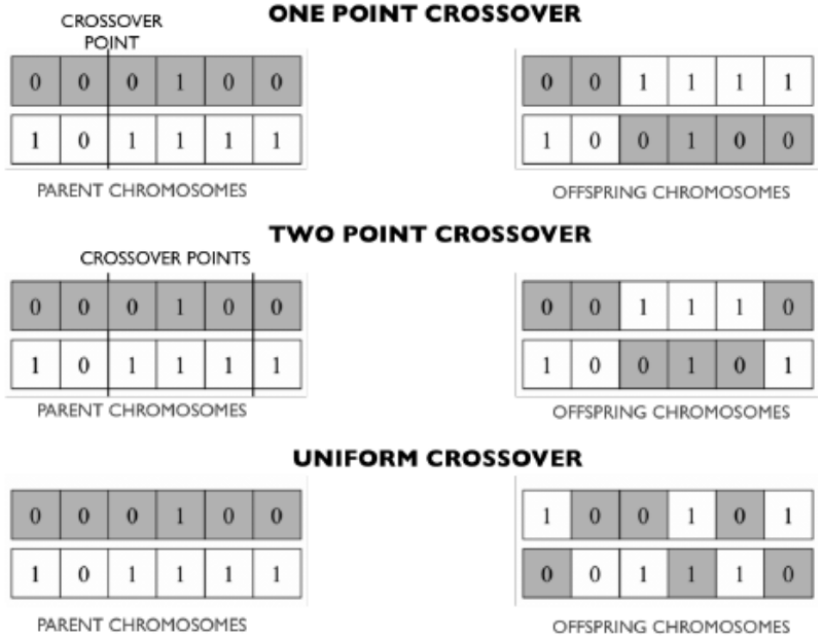
- **Ελιτισμός (Elitism):** Η τεχνική αυτή μπορεί να εφαρμοστεί συνεργατικά με άλλες τεχνικές επιλογής. Μετά από την αξιολόγηση όλων των λύσεων του πληθυσμού, επιλέγεται ένα ποσοστό (έστω 10%) του πληθυσμού ως «ελιτ» και αφορά τις λύσεις με τα υψηλότερα fitness scores. Οι λύσεις αυτές απομονώνονται και περνούν στην επόμενη γενιά ανέπαφες, χωρίς δηλαδή να γίνει διασταύρωση με κάποια άλλη λύση. Έτσι διατηρούνται στις γενιές οι καλύτερες λύσεις και παράλληλα η διαφορετικότητα μέσω των τεχνικών επιλογής που εφαρμόζονται στο υπόλοιπο 90% του πληθυσμού.

3.3.5.2. Αναπαραγωγή

Αφού έχει δημιουργηθεί η δεξαμενή ζευγαρώματος με τους επιλεγμένους γονείς, ακολουθεί η **διασταύρωση (crossover)** αυτών ανά ζεύγη. Η διαδικασία αυτή έχει ως σκοπό την ανταλλαγή γενετικού υλικού (πληροφορίας μέσω των γονιδίων) μεταξύ των γονέων και τη δημιουργία απογόνων (offsprings). Στη διαδικασία της αναπαραγωγής συμπεριλαμβάνεται και η πιθανή **μετάλλαξη (mutation)** γονιδίων, δηλαδή η τυχαία διαταραχή των χαρακτηριστικών ενός μικρού ποσοστού των απογόνων-λύσεων, με σκοπό την προώθηση της

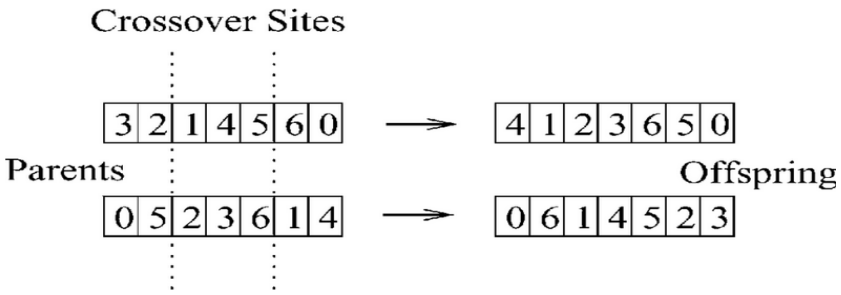
ποικιλομορφίας στον πληθυσμό και την περαιτέρω εξερεύνηση του χώρου αναζήτησης. Κοινές τεχνικές διασταύρωσης είναι οι παρακάτω:

- **Διασταύρωση ενός σημείου (One-point crossover):** Στην τεχνική αυτή ένα μόνο σημείο διασταύρωσης επιλέγεται τυχαία κατά μήκος των χρωμοσωμάτων των γονέων. Οι απόγονοι δημιουργούνται με την ανταλλαγή του γενετικού υλικού μεταξύ των γονέων στο σημείο διασταύρωσης. Αποτελεί εύκολη και απλή στην εφαρμογή τεχνική αλλά έχει περιορισμούς ως προς την εξερεύνηση του χώρου των λύσεων και των αλληλεπιδράσεων μεταξύ των γονιδίων.
- **Διασταύρωση δύο σημείων (Two-point crossover):** Η διασταύρωση δύο σημείων περιλαμβάνει την επιλογή δύο σημείων διασταύρωσης κατά μήκος των χρωμοσωμάτων των γονέων, και το γενετικό υλικό μεταξύ των δύο αυτών σημείων ανταλλάσσεται μεταξύ των γονέων για να δημιουργηθούν απόγονοι.
- **Διασταύρωση Ν σημείων (N-point crossover):** Αποτελεί επέκταση των δύο προηγούμενων τεχνικών, όπου επιλέγονται Ν σημεία διασταύρωσης κατά μήκος των χρωμοσωμάτων των γονέων, με σκοπό την προώθηση της διαφορετικότητας.
- **Ομοιόμορφη διασταύρωση (Uniform crossover):** Κάθε γονίδιο στους απογόνους προέρχεται τυχαία από έναν από τους γονείς με την ίδια πιθανότητα. Η τεχνική αυτή επιτρέπει μία πιο τυχαιοποιημένη ανταλλαγή πληροφορίας οδηγώντας δυνητικά σε καλύτερη εξερεύνηση του χώρου των λύσεων.



Εικόνα 3.14. *Εφαρμογή κοινών τεχνικών διασταύρωσης*

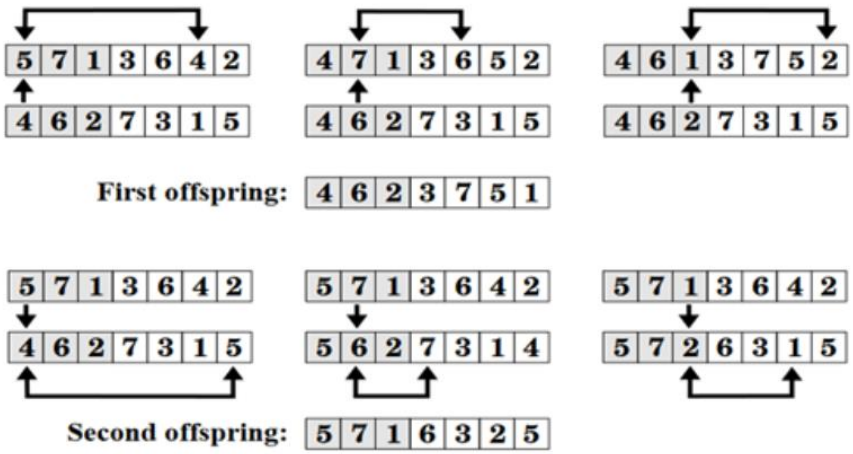
- **Μερικώς αντιστοιχισμένη διασταύρωση (Partially mapped crossover):** Η τεχνική «PMX» διατηρεί τη σχετική σειρά των στοιχείων μεταξύ δύο επιλεγμένων σημείων διασταύρωσης ενώ ανταλλάσσει το γενετικό υλικό έξω από αυτά τα σημεία. Έτσι διασφαλίζεται ότι οι απόγονοι διατηρούν έγκυρες μεταθέσεις στοιχείων, καθιστώντας την τεχνική κατάλληλη για προβλήματα όπου η σειρά των στοιχείων είναι σημαντική.



Εικόνα 3.15. *Εφαρμογή τεχνικής PMX*

Είναι σημαντικό να αναφερθεί ότι σε εφαρμογές γενετικών αλγορίθμων για επίλυση ΠΔΟ (δεδομένου ότι η κωδικοποίηση των χρωμοσωμάτων περιλαμβάνει τους δείκτες των πελατών, και όχι δυαδικούς αριθμούς για χρήση ή όχι συγκεκριμένων ακμών), όπου κάθε πελάτης εξυπηρετείται μία ακριβώς φορά από οποιοδήποτε όχημα, μία απλή διασταύρωση λύσεων με χρήση των παραπάνω τεχνικών (εκτός της PMX) θα οδηγήσει σε μη εφικτές λύσεις, καθώς ένας πελάτης Χ θα παίρνει τη θέση κάποιου πελάτη Υ σε μία άλλη λύση και το αντίστροφο. Έτσι, ο πελάτης Υ θα εξυπηρετείται δύο φορές, από το ίδιο ή διαφορετικό όχημα, στην πρώτη λύση ενώ ο πελάτης Χ δεν θα υπάρχει πλέον σε καμία σειρά εξυπηρέτησης στην λύση αυτή. Αντίστοιχα, ο πελάτης Χ θα εξυπηρετείται δύο φορές στην δεύτερη λύση ενώ ο πελάτης Υ δεν θα βρίσκεται στη διαδρομή κανενός οχήματος στην πρώτη λύση. Για την διασταύρωση λύσεων που περιέχουν σειρές εξυπηρέτησης πελατών απαιτούνται τροποποιημένοι τρόποι διασταύρωσης ώστε να μην παραβιάζεται ο περιορισμός της μίας ακριβώς και υποχρεωτικής εξυπηρέτησης κάθε πελάτη σε κάθε λύση.

Μία απλή προσέγγιση είναι η εύρεση της θέσης ενός πελάτη *i*, σε κάποιο όχημα *j* μίας λύσης-γονέα και ακολούθως η διαγραφή του συγκεκριμένου πελάτη από οποιαδήποτε διαδρομή *k* βρίσκεται στην δεύτερη λύση-γονέα, ώστε να γίνει ξανά εισαγωγή (insert) ακριβώς στην θέση *i* του *j* οχήματος της δεύτερης λύσης, για την δημιουργία του ενός εκ των δύο απογόνων. Η αντίστοιχη διαδικασία θα ακολουθηθεί για και για τον δεύτερο απόγονο με κάποιον άλλο πελάτη *l*. Απαιτείται όμως κι εδώ έλεγχος για την ύπαρξη ίδιου αριθμού οχημάτων και στις δύο λύσεις αλλά και για το επαρκές μήκος διαδρομών για τις εισαγωγές πελατών στις κατάλληλες θέσεις εξυπηρέτησης.

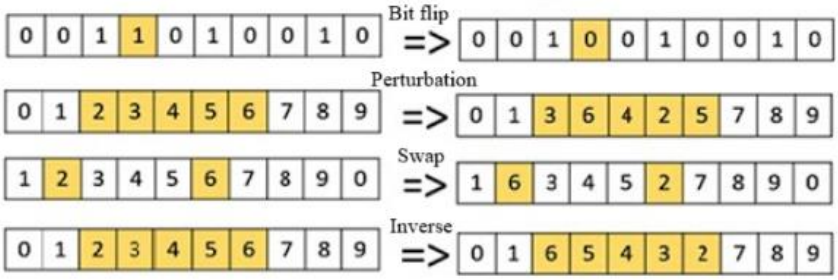


Εικόνα 3.16*. Τεχνική διασταύρωσης για εφαρμογή σε ΠΔΟ (απλή εφαρμογή με 1 όχημα σε κάθε λύση-γονέα)*

Η μετάλλαξη που υπάρχει πιθανότητα να υποστεί μικρός αριθμός λύσεων κατά τη μετάβαση στην επόμενη γενιά, αποτελεί τον ιδιαίτερο τρόπο των γενετικών αλγορίθμων να αποτρέπουν την σύγκλιση όλων των ατόμων του πληθυσμού σε ένα μικρό μέρος του χώρου λύσεων. Έτσι εξερευνώνται νέες περιοχές, και αν αυτές οδηγήσουν τυχαία σε λύσεις με καλά χαρακτηριστικά, θα βοηθήσουν στην βελτίωση του τελικού αποτελέσματος με τη μεταβίβαση αυτών των χαρακτηριστικών σε κάθε επόμενη γενιά, αφού είναι πιθανότερο να επιλεγούν ως γονείς κατά την επόμενη επανάληψη. Κοινές τεχνικές μετάλλαξης είναι:

- **Τυχαία μετάλλαξη (Random mutation):** Επιλέγεται ένα τυχαίο γονίδιο ή ένα σύνολο γονιδίων στο χρωμόσωμα μίας λύσης και η τιμή τους αλλάζει τυχαία.
- **Μετάλλαξη αναστροφής bit (Bit flip mutation):** Η τεχνική αυτή χρησιμοποιείται συνήθως για χρωμοσώματα με δυαδική κωδικοποίηση. Επιλέγεται τυχαία ένα bit στο χρωμόσωμα της λύσης και αναστρέφεται (αλλάζει από 0 σε 1 ή αντίστροφα).
- **Μετάλλαξη ανταλλαγής (Swap mutation):** Χρησιμοποιείται για κωδικοποιήσεις χρωμοσωμάτων όπου η σειρά των στοιχείων έχει σημασία. Αφορά την ανταλλαγή των τιμών μεταξύ δύο τυχαία επιλεγμένων θέσεων στο χρωμόσωμα της λύσης.
- **Μετάλλαξη αναστροφής (Inverse mutation):** Ένα υποσύνολο γονιδίων ανάμεσα σε δύο τυχαία επιλεγμένες θέσεις στο χρωμόσωμα αντιστρέφεται. Σε περίπτωση που εφαρμοστεί σε λύσεις από ΠΔΟ με δείκτες πελατών στην αναπαράσταση των χρωμοσωμάτων, παρατηρούμε ότι ταυτίζεται με μία επανάληψη της ευρετικής μεθόδου 2-opt.
- **Μετάλλαξη διαταραχής (Perturbation mutation):** Ένα υποσύνολο γονιδίων ανάμεσα σε τυχαία επιλεγμένες θέσεις υπόκειται σε τυχαίο ανακάτεμα. Μπορεί να

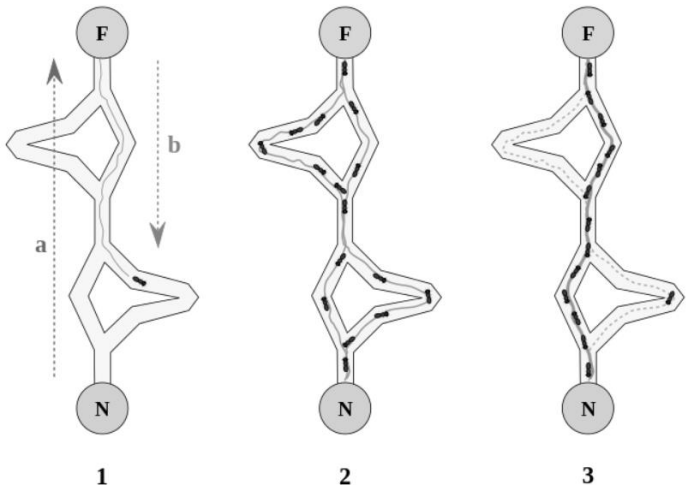
εφαρμοστεί είτε με τυχαίες μεταθέσεις των στοιχείων του υποσυνόλου (όταν η σειρά τους έχει σημασία) είτε ως πολλαπλή τυχαία μετάλλαξη στοιχείων.



Εικόνα 3.17*. Εφαρμογή κοινών τεχνικών μετάλλαξης*

3.3.6. Αλγόριθμος βελτιστοποίησης αποικίας μυρμηγκιών (Ant colony optimization)

Ο αλγόριθμος βελτιστοποίησης αποικίας μυρμηγκιών (ACO) αρχικά προτάθηκε από τον Marco Dorigo (1992) και είναι ένας μεθευρετικός αλγόριθμος βελτιστοποίησης με ευρεία εφαρμογή σε προβλήματα συνδυαστικής βελτιστοποίησης, εμπνευσμένος από τη συμπεριφορά των μυρμηγκιών μίας αποικίας κατά την αναζήτηση τροφής. Ο ACO αξιοποιεί την αρχή της **στιγμεργίας**, σύμφωνα με την οποία τα μυρμήγκια επικοινωνούν έμμεσα μεταξύ τους εναποθέτοντας και ακολουθώντας ίχνη **φερομόνης (pheromone)**. Η φερομόνη είναι χημική ουσία που απελευθερώνεται από ένα ζώο προκειμένου να προκαλέσει μια συγκεκριμένη αντίδραση, ή να μεταδώσει κάποιο μήνυμα σε ένα άλλο άτομο του ίδιου είδους. Τα μυρμήγκια στη φύση εναποθέτουν αυτή τη χημική ουσία ώστε να προσελκύσουν άλλα μυρμήγκια όταν ανακαλύπτουν πηγές τροφής ή όταν εντοπίζουν κινδύνους. Παράλληλα όμως με την πάροδο του χρόνου (επαναλήψεων) οι φερομόνες σταδιακά εξατμίζονται.



Εικόνα 3.18. *Τα μυρμήγκια επιλέγουν κυρίως διαδρομές με μεγαλύτερη ποσότητα φερομονών*

Τα ψηφιακά μυρμήγκια κατασκευάζουν λύσεις επαναληπτικά επιλέγοντας πιθανολογικά **ακμές** (απευθείας διαδρομές μεταξύ πελατών) ή μέρη της λύσης με βάση τα τρέχοντα

επίπεδα φερομόνης αλλά και τις ευρετικές πληροφορίες που καθορίζονται από τον χρήστη ανάλογα με το πρόβλημα. Μαθηματικά, αυτή η διαδικασία μπορεί να αναπαρασταθεί ως εξής:

$$P_{ij,k} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{m \in allowed\ moves} \tau_{im}^{\alpha} \cdot \eta_{im}^{\beta}} \qquad (1)$$

όπου P_{ij} είναι η πιθανότητα επιλογής της ακμής *(i, j)*, *k* ο δείκτης του μυρμηγκιού, τ_{ij} η τρέχουσα ποσότητα φερομόνης στην ακμή *(i, j)*, η_{ij} η ευρετική πληροφορία (για απλά ΠΔΟ συνήθως είναι το αντίστροφο της απόστασης/χρόνου μεταξύ των πελατών *i* και *j*), ενώ οι παράμετροι α και β καθορίζουν το πόσο έντονα επηρεάζονται οι αποφάσεις των μυρμηγκιών από την ποσότητα της φερομόνης και από την τιμή της ευρετικής πληροφορίας, αντίστοιχα. Το σύνολο «allowed moves» περιέχει όλες τις ακμές που ξεκινούν από τον πελάτη *i* προς οποιονδήποτε άλλο μη εξυπηρετημένο πελάτη και που η επιλογή του δεν οδηγεί σε παραβίαση περιορισμών (εναλλακτικά μπορούμε να θέτουμε την πιθανότητα μετάβασης ίση με 0 σε τέτοιες περιπτώσεις). Ο αριθμητής του κλάσματος ονομάζεται και επιθυμία (desire) οπότε ο τύπος γράφεται και ως $P_{ij} = \frac{desire}{sum\ of\ desires}$.

Μετά την κατασκευή των λύσεων από όλα τα μυρμήγκια (τέλος τρέχουσας επανάληψης) και την αξιολόγηση αυτών από την αντικειμενική συνάρτηση, τα επίπεδα φερομόνης σε όλες τις ακμές ενημερώνονται για να αντικατοπτρίζουν την ποιότητα των λύσεων που βρέθηκαν. Τα μυρμήγκια εναποθέτουν φερομόνες κατά μήκος των ακμών που διασχίζουν, με την ποσότητα της φερομόνης ανάλογη με την ποιότητα της λύσης. Στην ενημέρωση επίσης συμπεριλαμβάνεται και η εξάτμιση μέρους της φερομόνης που προϋπήρχε σε κάθε ακμή. Μαθηματικά, ο κανόνας ενημέρωσης των φερομονών μπορεί να εκφραστεί ως εξής:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, & if\ (i,j) \in T_k(t) \\ 0, & if\ (i,j) \notin T_k(t) \end{cases} \qquad (2)$$

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \qquad (3)$$

όπου *m* ο αριθμός των μυρμηγκιών, *k* ο δείκτης του μυρμηγκιού, *t* ο αριθμός της επανάληψης του αλγορίθμου, *Q* μία σταθερά που εκφράζει την ένταση των φερομονών (pheromone intensity) στις ακμές, $L_k(t)$ η συνολική απόσταση/χρόνος της λύσης (ποιότητα της λύσης) του

μυρμηγκιού k, $T_k(t)$ το σύνολο των ακμών που περιέχονται στη λύση του μυρμηγκιού k για την επανάληψη t, ρ ο συντελεστής εξάτμισης (evaporation coefficient) των φερομονών.

Η εξίσωση (2) εκφράζει την αύξηση της ποσότητας της φερομόνης στην ακμή (i, j) από κάθε μυρμήγκι k κατά την t επανάληψη του αλγορίθμου. Όσο καλύτερη είναι η λύση k, τόσο μικρότερο είναι το L_k (αφού αφορά απόσταση ή χρόνο), οπότε η ποσότητα της φερομόνης που αφήνεται σε κάθε ακμή που περιέχει η λύση k είναι μεγαλύτερη (για οποιαδήποτε τιμή της σταθεράς $Q > 0$ που θέτει ο χρήστης).

Η εξίσωση (3) εκφράζει την αλλαγή στην ποσότητα της φερομόνης στην ακμή (i, j) συνολικά από όλα τα μυρμήγκια-λύσεις κατά την επανάληψη $t+1$. Αυτή ισούται με την ποσότητα που προϋπήρχε και την εξάτμιση που ένα μέρος της υπέστη, αυξημένη κατά την ποσότητα που άφησε το σύνολο των m μυρμηγκιών.

Η ρύθμιση των παραμέτρων (parameter tuning) είναι από τα πιο κρίσιμα στοιχεία των αλγορίθμων ACO. Απαιτείται επιλογή του αριθμού m των μυρμηγκιών, η προσαρμογή της τιμής του συντελεστή εξάτμισης ρ, της σχετικής σημασίας α της φερομόνης στις ακμές και της ευρετικής πληροφορίας β αλλά και του συντελεστή Q. Η ποιότητα των αποτελεσμάτων εξαρτάται σε πολύ μεγάλο βαθμό από την κατάλληλη ρύθμιση αυτών των παραμέτρων, η οποία συχνά απαιτεί πειραματισμό με πολλαπλές δοκιμές τιμών με στόχο την κατάλληλη αναλογία στρατηγικών εξερεύνησης-εκμετάλλευσης.

Αναλυτικά, τα βήματα που ακολουθούνται τυπικά σε έναν αλγόριθμο ACO είναι:

1. **Ορισμός των παραμέτρων**: Μέγεθος πληθυσμού μυρμηγκιών, συντελεστής εξάτμισης φερομόνης, ευρετική πληροφορία, παράμετροι σχετικής σημασίας της φερομόνης και της ευρετικής πληροφορίας, σταθερά έντασης φερομονών.
2. **Ορισμός του κριτηρίου τερματισμού**, που μπορεί να είναι ο αριθμός των επαναλήψεων, η εύρεση μίας λύσης συγκεκριμένης ποιότητας ή κριτήρια υπολογιστικών πόρων και χρόνου.
3. **Αρχικοποίηση των επιπέδων των φερομονών με την τιμή 0 σε όλες τις ακμές** και προαιρετικά η αύξηση της φερομόνης σε συγκεκριμένες ακμές με τη χρήση μιας αρχικής λύσης που εξήγαγε κάποια ευρετική μέθοδος (π.χ. πλησιέστερος γείτονας, αλγόριθμος εξοικονομήσεων κ.α.) ώστε να αυξηθούν αρχικά οι πιθανότητες τα μυρμήγκια να επιλέξουν να κινηθούν στις ακμές που η λύση αυτή περιλαμβάνει.
4. **Κάθε μυρμήγκι κατασκευάζει μία λύση** επαναληπτικά επιλέγοντας πιθανολογικά ακμές σύμφωνα με την εξίσωση (1).

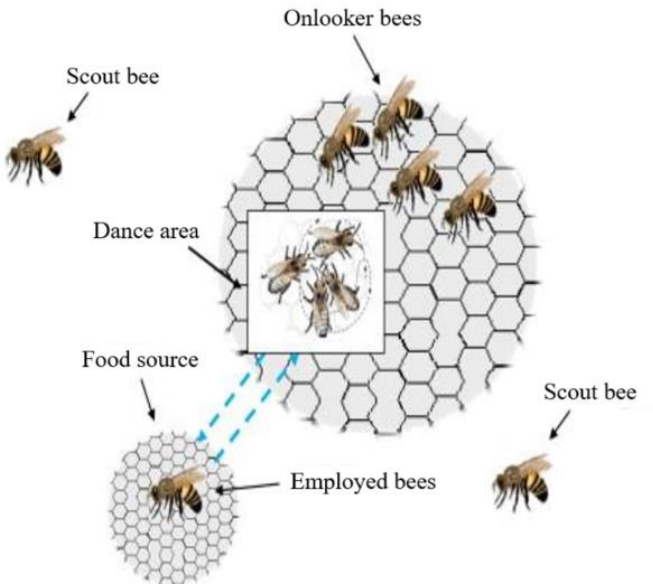
5. **Αξιολόγηση όλων των λύσεων** βάσει της αντικειμενικής συνάρτησης.
6. **Ενημέρωση των επιπέδων φερομόνης**, με αύξηση και εξάτμιση σε όλες τις ακμές σύμφωνα με τις εξισώσεις (2) και (3).
7. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 9.
8. Επανάληψη των βημάτων 4-7 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
9. **Επιστροφή της καλύτερης λύσης που βρέθηκε** κατά τη διάρκεια της αναζήτησης.

Ακόμη κι αν η αντικειμενική συνάρτηση αναζητεί την λύση με την ελάχιστη συνολική απόσταση ή χρόνο σε ένα ΠΔΟ, συχνά επιλέγεται να επιστραφεί και η λύση με τα υψηλότερα (αθροιστικά σε όλες τις ακμές) επίπεδα φερομονών κατά την τελευταία επανάληψη του αλγορίθμου.

3.3.7. Αλγόριθμος τεχνητής αποικίας μελισσών (Artificial bee colony)

Ο αλγόριθμος τεχνητής αποικίας μελισσών (ABC) αποτελεί μεθευρετικό αλγόριθμο βελτιστοποίησης εμπνευσμένο από τη συμπεριφορά των μελισσών κατά την αναζήτηση τροφής. Αποτελεί αλγόριθμο βασισμένο σε **πληθυσμούς λύσεων** και **κάθε λύση αναπαρίσταται ως μία πηγή τροφής**. Προτάθηκε από τον Karaboga το 2005 και έκτοτε χρησιμοποιείται ευρέως για την επίλυση διαφόρων προβλημάτων βελτιστοποίησης.

Ο ABC μοντελοποιεί τη διαδικασία αναζήτησης τροφής των μελισσών, όπου αυτές αναζητούν πηγές τροφής (λουλούδια) και μεταβιβάζουν πληροφορίες σχετικά με την ποιότητα αυτών των πηγών σε άλλες μέλισσες εντός της αποικίας. Η επικοινωνία μεταξύ των μελισσών μπορεί να κατηγοριοποιηθεί σε διάφορους τύπους, συμπεριλαμβανομένης της περίφημης «γλώσσας χορού», όπου οι μέλισσες που επιστρέφουν εκτελούν τον περίφημο χορό της κυψέλης (waggle dance) για να μεταφέρουν λεπτομέρειες όπως η κατεύθυνση, η απόσταση και η ποιότητα των πηγών τροφής. Ακόμη, οι μέλισσες εναποθέτουν φερομόνες, χημικές ουσίες που απελευθερώνονται από το σώμα τους, για να σημειώσουν μονοπάτια και χρησιμοποιούν τις κεραίες τους για τη μεταβίβαση σημάτων και δονήσεων. Ο ABC προσομοιώνει τις αλληλεπιδράσεις αυτές μεταξύ των **εργατριών**, των **θεατών** και των **ανιχνευτών** μελισσών για να εξερευνήσει τον χώρο λύσεων με σκοπό την ανακάλυψη περιοχών λύσεων (πηγών τροφής) υψηλής ποιότητας.



Εικόνα 3.19. *Αναπαράσταση των τριών ειδών μελισσών του αλγορίθμου ABC*

Κάθε επανάληψη (ή κύκλος) του αλγορίθμου αποτελείται από τρεις φάσεις που λαμβάνουν χώρα η μία μετά την άλλη:

1. **Φάση εργατριών μελισσών (Employed bees phase):** Κάθε εργάτρια μέλισσα πηγαίνει σε μία πηγή τροφής (λύση) του πληθυσμού και πραγματοποιεί μία αναζήτηση γύρω από αυτήν. Στην πράξη γίνεται δημιουργία μίας εφικτής γειτονικής λύσης για κάθε λύση του πληθυσμού με τη χρήση κάποιου ευρετικού αλγορίθμου τοπικής αναζήτησης και αν αυτή είναι καλύτερη, τότε αντικαθιστά την προηγούμενή της στον πληθυσμό.
2. **Φάση θεατών μελισσών (Onlooker bees phase):** Οι θεατές μέλισσες της αποικίας παρατηρώντας τον «χορό» των εργατριών μελισσών που γύρισαν από τις πηγές τροφής, επιλέγουν για το αν θα ακολουθήσουν τις εργάτριες μέλισσες σε κάποια πηγή, ώστε να την εξερευνήσουν ακόμη περισσότερο, ανάλογα με την ένταση του χορού της εργάτριας μέλισσας, πρακτικά ανάλογα με την ποιότητα της λύσης σύμφωνα με την αντικειμενική συνάρτηση. Έτσι, αφού αξιολογηθούν όλες οι λύσεις (ποιότητα πηγών τροφής) στον πληθυσμό, οι θεατές μέλισσες επιλέγουν μία λύση από τον πληθυσμό με πιθανότητα επιλογής ανάλογη του fitness score της λύσης. Η διαδικασία της επιλογής αυτής γίνεται για φορές όσες και το μέγεθος του πληθυσμού και ακολουθείται αλγοριθμικά η ίδια διαδικασία σύμφωνα με την «τεχνική της ρουλέτας» που είδαμε στο κεφάλαιο των γενετικών αλγορίθμων ως τεχνική επιλογής γονέων.

3. **Φάση ανιχνευτών μελισσών (Scout bees phase):** Παρακολουθώντας για κάθε κύκλο την βελτίωση ή μη της ποιότητας κάθε λύσης, οι ανιχνευτές μέλισσες αναλαμβάνουν να εξερευνήσουν εντελώς ξένες πηγές τροφής, εγκαταλείποντας λύσεις που δεν έχουν βελτιωθεί για κάποιον αριθμό κύκλων. Λύσεις που θεωρούνται πλέον μη ελπιδοφόρες διαγράφονται από τον πληθυσμό και αντικαθίστανται από νέες λύσεις που θα δημιουργηθούν με τυχαιότητα. Η φάση αυτή είναι που δίνει την δυνατότητα στον αλγόριθμο να διαφεύγει από περιοχές μη ποιοτικών λύσεων ή τοπικά βέλτιστα και να εξερευνά ξένες περιοχές του χώρου αναζήτησης.

Αναλυτικά, τα βήματα που ακολουθούνται τυπικά σε έναν αλγόριθμο ABC είναι:

1. **Ορισμός παραμέτρων**: Μέγεθος πληθυσμού πηγών τροφής (λύσεων), όριο κύκλων μη βελτίωσης των λύσεων (για την φάση των ανιχνευτών μελισσών).
2. **Ορισμός του κριτηρίου τερματισμού**, που μπορεί να είναι ο αριθμός των επαναλήψεων ή κύκλων, η εύρεση μίας λύσης συγκεκριμένης ποιότητας ή κριτήρια υπολογιστικών πόρων και χρόνου.
3. **Δημιουργία του αρχικού πληθυσμού εφικτών λύσεων**, με κάποια ευρετική κατασκευαστική μέθοδο ή με τυχαιότητα.
4. **Αξιολόγηση όλων των λύσεων** σύμφωνα με την αντικειμενική συνάρτηση.
5. **Φάση εργατριών μελισσών**.
6. **Φάση θεατών μελισσών**.
7. **Φάση ανιχνευτών μελισσών**, μόνο για τις λύσεις που δεν έχουν βελτιωθεί για αριθμό κύκλων ίσο με το όριο μη βελτίωσης.
8. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 10.
9. Επανάληψη των βημάτων 4-8 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
10. **Επιστροφή της καλύτερης λύσης που βρέθηκε** κατά τη διαδικασία της αναζήτησης.

3.3.8. Διαδικασία άπληστης τυχαιοποιημένης προσαρμοστικής αναζήτησης (Greedy randomized adaptive search procedure, GRASP)

Η διαδικασία άπληστης τυχαιοποιημένης προσαρμοστικής αναζήτησης (GRASP) προτάθηκε αρχικά από τους Feo και Resende (1989) βασιζόμενοι στην λογική της άπληστης τυχαιοποιημένης ευρετικής μεθόδου (semi-greedy heuristics) των Hart και Shogan (1987). Αποτελεί επαναληπτική διαδικασία για την εύρεση προσεγγιστικών λύσεων σε προβλήματα συνδυαστικής βελτιστοποίησης συνδυάζοντας στοιχεία άπληστων κατασκευαστικών αλγορίθμων, τυχαιότητας και τοπικής αναζήτησης. Κάθε επανάληψη του αλγορίθμου

αποτελείται από δύο φάσεις, μία φάση κατασκευής μίας λύσης και μία φάση τοπικής αναζήτησης με σκοπό τη βελτίωση αυτής της λύσης.

- **Φάση κατασκευής (Construction phase):** Η φάση αυτή επιτυγχάνεται μέσω της στρατηγικής τυχαιοποιημένης απληστίας (randomized greedy strategy). Η στρατηγική επιλογής κάθε επόμενου στοιχείου (πελάτη στην περίπτωση των ΠΔΟ) βασίζεται στην τυχαία επιλογή από μία λίστα υποψηφίων, που **ονομάζεται λίστα περιορισμού των υποψηφίων (restricted candidate list, RCL)**. Η λίστα RCL δημιουργείται εισάγοντας σε αυτήν έναν περιορισμένο αριθμό υποψήφιων στοιχείων, τα οποία κατατάσσονται σε σειρά ανάλογα με το πως θα αλλάξει σύμφωνα με την αντικειμενική συνάρτηση η ποιότητα της λύσης ή του μέρους της (διαδρομή ενός οχήματος). Το μέγεθος ή τα όριά της καθορίζονται από τον χρήστη και μπορεί να είναι σταθερά ή και μεταβλητά κατά την επαναληπτική πρόσθεση στοιχείων. Αποτελεί πολύ σημαντική παράμετρο καθώς ένα μεγάλο μέγεθος θα οδηγήσει σε αρκετά τυχαίες λύσεις (εστίαση στην στρατηγική εξερεύνησης) ενώ ένα μικρό μέγεθος θα οδηγήσει σε περιορισμένη αναζήτηση του χώρου των λύσεων (εστίαση στην στρατηγική εκμετάλλευσης) και παρόμοια ίσως κάθε φορά αποτελέσματα.

  Στην πράξη, για την επίλυση ΠΔΟ, αν έχουμε θέσει ως L το μέγεθος της λίστας RCL και χρησιμοποιούμε τον πλησιέστερο γείτονα ως κατασκευαστική μέθοδο, τότε αυτή θα περιέχει τους L κοντινότερους πελάτες (ταξινομημένους) στον τελευταίο πελάτη (ή την αποθήκη) της έως τώρα διαδρομής. Η επιλογή του πελάτη που θα προστεθεί σε κάθε επανάληψη της κατασκευής μπορεί να γίνει είτε εντελώς τυχαία, είτε λαμβάνοντας επίσης υπόψη και την θέση του στην ταξινομημένη λίστα, δίνοντας μεγαλύτερη πιθανότητα επιλογής στις καλύτερες βάσει απόστασης επιλογές. Η βαρύτητα που θα δοθεί στην κάθε επιλογή καθορίζεται με την εισαγωγή μίας ακόμη παραμέτρου, του **παράγοντα απληστίας α (greedy factor)**. Η «προσαρμοστική» φύση του αλγορίθμου δικαιολογείται από το γεγονός ότι υπάρχει η δυνατότητα δυναμικής αλλαγής των τιμών των παραμέτρων (μέγεθος λίστας RCL, τιμή παράγοντα απληστίας, κατασκευαστική ευρετική μέθοδος) κατά τη διάρκεια της φάσης κατασκευής της αρχικής λύσης.
- **Φάση τοπικής αναζήτησης (Local search phase):** Αφού δημιουργηθεί η αρχική λύση για την τρέχουσα επανάληψη, επιλέγεται μία ή περισσότερες μέθοδοι τοπικής αναζήτησης για την προσπάθεια βελτίωσής της. Για την επίλυση ΠΔΟ, αυτές μπορεί να είναι ευρετικές μέθοδοι βελτίωσής των διαδρομών (1-1 exchange, 1-0 exchange,

2-opt, 3-opt, …) ή ακόμη και πιο ισχυροί μεθευρετικοί αλγόριθμοι που απαιτούν μία αρχική λύση για να εξερευνήσουν περαιτέρω τον χώρο των λύσεων, όπως η περιορισμένη αναζήτηση (TS) και η προσομοιωμένη ανόπτηση (SA).

Σε γενικές γραμμές, τα βήματα που ακολουθούνται κατά την εφαρμογή του GRASP είναι:

1. **Ορισμός παραμέτρων**: Τιμή ή όρια τιμών (σε περίπτωση δυναμικών μεταβολών) του παράγοντα απληστίας, μέγεθος ή όρια μεγεθών της λίστας RCL, κατασκευαστική ευρετική μέθοδος (πλησιέστερος γείτονας, αλγόριθμος εξοικονομήσεων, …) για την φάση κατασκευής της λύσης, ευρετικοί αλγόριθμοι τοπικής αναζήτησης ή/και μεθευρετικοί αλγόριθμοι για την φάση τοπικής αναζήτησης.
2. **Ορισμός του κριτηρίου τερματισμού**, που μπορεί να είναι ο αριθμός των επαναλήψεων, ή η εύρεση μίας λύσης συγκεκριμένης ποιότητας ή κριτήρια υπολογιστικών πόρων και χρόνου.
3. **Φάση κατασκευής μίας λύσης**: Επαναληπτική δημιουργία λίστας RCL και προσθήκη στοιχείων με χρήση της επιλεγμένης κατασκευαστικής ευρετικής μεθόδου έως ότου δημιουργηθεί μία πλήρης λύση (λίστα με διαδρομές οχημάτων).
4. **Φάση τοπικής αναζήτησης** με χρήση των επιλεγμένων ευρετικών ή/και μεθευρετικών μεθόδων έως ότου δεν επιτυγχάνεται περαιτέρω βελτίωση.
5. **Αξιολόγηση της νέας λύσης** και σύγκριση με την έως τώρα καλύτερη λύση. Αν είναι καλύτερη, αντικατάσταση με την λύση της τελευταίας επανάληψης.
6. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 8.
7. Επανάληψη των βημάτων 3-6 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
8. **Επιστροφή της καλύτερης λύσης που βρέθηκε** κατά την επαναληπτική διαδικασία.

3.4. Μεθευρετικοί αλγόριθμοι πολλαπλών στόχων (Multi-objective metaheuristics)

Οι μεθευρετικοί αλγόριθμοι πολλαπλών στόχων είναι αλγόριθμοι βελτιστοποίησης που έχουν σχεδιαστεί για την αντιμετώπιση προβλημάτων με πολλαπλούς αντικρουόμενους (ή μη) στόχους ή αντικειμενικές συναρτήσεις (multiple objectives). Σε πολλά σενάρια του πραγματικού κόσμου, η βελτιστοποίηση ενός μεμονωμένου στόχου μπορεί να μην είναι επαρκής, καθώς συχνά υπάρχουν πολλά κριτήρια αξιολόγησης που πρέπει να ληφθούν υπόψη ταυτόχρονα. Γενικά, τα προβλήματα βελτιστοποίησης πολλαπλών στόχων μπορούν να διατυπωθούν μαθηματικά ως εξής:

$$\textbf{\textit{min/max}}\ y = F(x) = (f_1(x), f_2(x), \ldots, f_n(x))$$

υπό τους περιορισμούς:

$$g_i(X) \leq 0,\ i = 1, 2, \ldots, k_1$$

$$g_i(X) = 0,\ i = k_1+1, \ldots, k_2$$

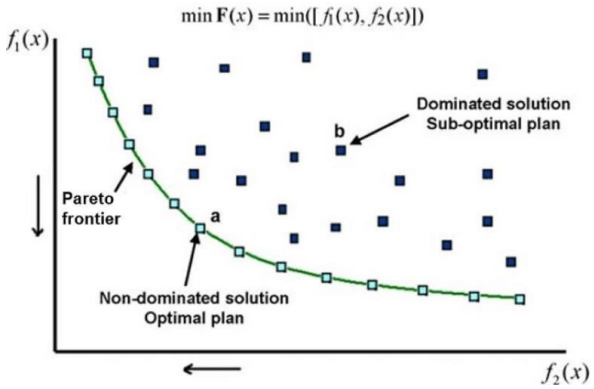
$$\textit{όπου}\ x = (x_1, x_2, \ldots, x_j) \in X,\ y = (y_1, y_2, \ldots, y_t) \in Y$$

όπου $\boldsymbol{x}$ είναι το διάνυσμα των μεταβλητών απόφασης, $\boldsymbol{X}$ ο χώρος όλων των επιτρεπτών τιμών των μεταβλητών απόφασης, $\boldsymbol{y}$ το διάνυσμα των τιμών των στόχων για κάθε συνδυασμό και $\boldsymbol{Y}$ ο χώρος των επιτρεπτών τιμών στόχων, $\boldsymbol{n}$ το πλήθος των στόχων (αντικειμενικών συναρτήσεων), $\boldsymbol{k_1}$ το πλήθος των περιορισμών ανισότητας, $\boldsymbol{k_2}$ το συνολικό πλήθος περιορισμών (ανισοτήτων και ισοτήτων).

Η πρόκληση κατά την αντιμετώπιση των προβλημάτων αυτών είναι ότι οι στόχοι σχετίζονται, περιορίζονται, ή ακόμη έρχονται και σε σύγκρουση μεταξύ τους. Σκοπός είναι η εύρεση ενός συνόλου λύσεων που αντιπροσωπεύουν καλύτερα τις αντισταθμίσεις (trade-offs) μεταξύ αυτών των αντικρουόμενων (ή μη) στόχων, γνωστών ως βέλτιστες κατά Pareto λύσεις (Pareto optimal solutions) ή **μέτωπο Pareto (Pareto frontier)**.

Ο όρος «Pareto βέλτιστες λύσεις» προέρχεται από τη θεωρία παιγνίων και την οικονομική επιστήμη (Vilfredo Pareto, 1906). Το σύνολο αυτό αναπαριστά μια κατάσταση όπου καμία λύση δεν μπορεί να βελτιωθεί σε έναν στόχο χωρίς να θυσιαστεί κλάσμα της απόδοσής της σε κάποιον άλλο (τουλάχιστον έναν) στόχο. Αυτό σημαίνει ότι κάθε λύση στο μέτωπο Pareto είναι μια βέλτιστη λύση για τα δεδομένα στο πλαίσιο των συγκεκριμένων στόχων, και καμία δεν μπορεί να θεωρηθεί καλύτερη από κάποια άλλη.

Οι λύσεις που δεν ανήκουν στο μέτωπο Pareto θεωρούνται υπο-βέλτιστες και καλούνται «κυριαρχούμενες» (dominated). Ο όρος «κυριαρχία» (domination) προέρχεται επίσης από την θεωρία παιγνίων και χαρακτηρίζει μία λύση (ή στρατηγική) ως «κυρίαρχη» έναντι κάποιας άλλης αν η πρώτη είναι τουλάχιστον όσο καλή είναι και η δεύτερη σε όλους τους στόχους, και υπερτερεί σε τουλάχιστον έναν. Σε περίπτωση που η πρώτη λύση υπερτερεί σε όλους τους στόχους, τότε καλείται **«αυστηρά κυρίαρχη» (strictly dominant)**, ενώ σε άλλη περίπτωση που είναι ισάξια σε τουλάχιστον έναν στόχο, καλείται **«ασθενώς κυρίαρχη» (weakly dominant)** έναντι της δεύτερης.



Εικόνα 3.20. *Απεικόνιση του μετώπου Pareto κατά τη βελτιστοποίηση με δύο αντικειμενικές συναρτήσεις προς ελαχιστοποίηση*

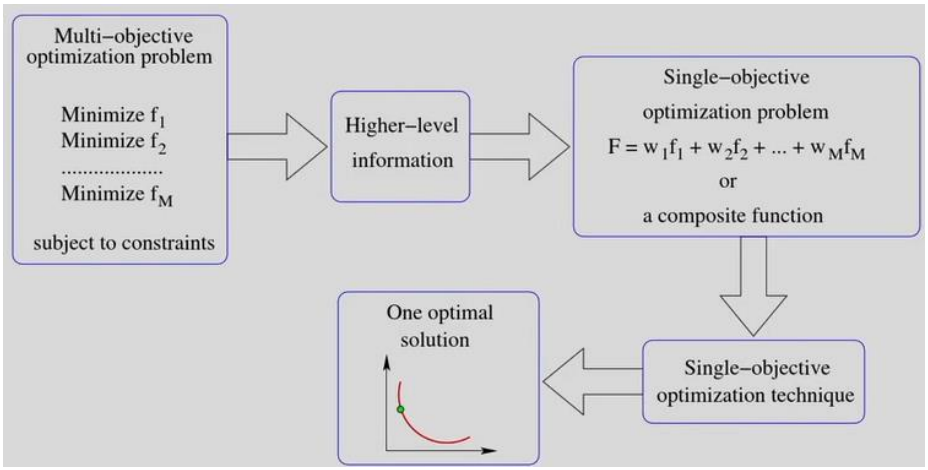
Μαθηματικά, τα παραπάνω θα μπορούσαν να εκφραστούν ως:

Έστω X το σύνολο των πιθανών λύσεων ενός προβλήματος βελτιστοποίησης. Αυτό το πρόβλημα μπορεί να έχει πολλαπλούς στόχους, n σε αριθμό, και για κάθε λύση $x \in X$ υπάρχουν πολλαπλές μετρήσεις απόδοσης (έστω ότι όλες είναι συναρτήσεις κόστους προς ελαχιστοποίηση). Αυτές συμβολίζονται ως $f_1(x)$, $f_2(x)$, ..., $f_n(x)$ όπου ο δείκτης $i=\{1, 2, ..., n\}$ αναφέρεται στον κάθε ξεχωριστό στόχο.

- ***Αυστηρή κυριαρχία (Strict dominance):*** *Κάθε λύση $y \in X$ κυριαρχείται αυστηρά από κάθε άλλη λύση $x \in X$ εάν $f_i(x) < f_i(y)$, $\forall i \in \{1,2,...,n\}$.*
- ***Ασθενής κυριαρχία (Weak dominance):*** *Κάθε λύση $y \in X$ κυριαρχείται ασθενώς από κάθε άλλη λύση $x \in X$ εάν $f_i(x) \leq f_i(y)$, $\forall i \in \{1,2,...,n\}$* ***και*** *$f_i(x) < f_i(y)$ για κάποιο $i \in \{1,2,...,n\}$.*
- ***Μέτωπο Pareto (Pareto frontier)****: Το σύνολο $P \subseteq X$ ονομάζεται μέτωπο Pareto εάν για κάθε λύση x στο σύνολο P, δεν υπάρχει καμία άλλη λύση y στο σύνολο X που να κυριαρχεί (ασθενώς) έναντι της x.*
  $P=\{x \in X : \nexists y \in X$ τέτοια ώστε $f_i(y) \leq f_i(x)$ $\forall i \in \{1,2,...,n\}$, και $f_j(y) < f_j(x)$ για κάποιο $j\}$.

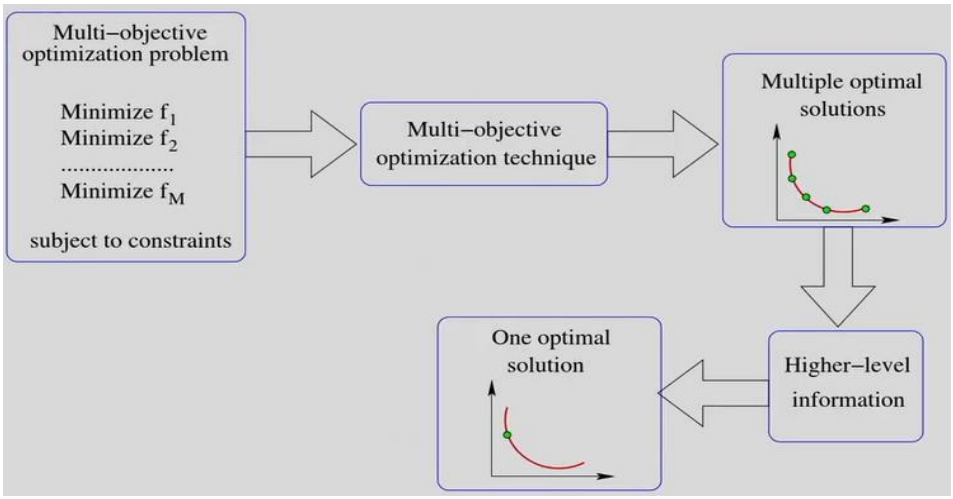
Αφού υπολογιστούν όλες οι βέλτιστες κατά Pareto λύσεις σε ένα πρόβλημα πολλαπλών στόχων, η τελική απόφαση ιδανικά θα παρθεί βάσει άλλων πληροφοριών υψηλότερου επιπέδου (higher level information) που δεν ήταν εξ αρχής γνωστές στον αλγόριθμο. Παράδειγμα τέτοιας πληροφορίας θα μπορούσε να είναι κάποιο άνω όριο που έχει θέσει μία εταιρεία μεταφορών για το συνολικό κόστος μετακίνησης (μεταξύ και άλλων στόχων προς ταυτόχρονη βελτιστοποίηση) του στόλου μεταφορών. Αν όμως υπολογιστούν όλες οι Pareto βέλτιστες λύσεις, μπορεί να παρατηρηθεί ένα πολύ μεγάλο όφελος συνολικά (μέσω των υπολοίπων αντικειμενικών συναρτήσεων) με την επιλογή λύσεων που παραβιάζουν κατά

πολύ λίγο το άνω όριο που είχε προηγουμένως τεθεί στο συγκεκριμένο στόχο, και μία μικρή υποχώρηση ίσως οδηγήσει σε πολλαπλά οφέλη (π.χ. αρκετά λιγότερες βλάβες στα οχήματα με χρήση ποιοτικότερου και λίγο ακριβότερου καυσίμου). Μέσω αυτού του απλού παραδείγματος μας γίνεται σαφές ότι η προσέγγιση της βελτιστοποίησης πολλαπλών στόχων με εξ αρχής προτιμήσεις (δίνοντας συγκεκριμένη βαρύτητα ή περιορίζοντας κάποια κριτήρια) δεν είναι ο ιδανικός τρόπος επίλυσης αυτών των προβλημάτων.



Εικόνα 3.21. *Λανθασμένη χρήση πληροφοριών και στόχευση σε μία μόνο Pareto βέλτιστη λύση*

Ο στόχος είναι να γνωρίζει ο λήπτης της απόφασης λεπτομερώς τα οφέλη ή τη ζημία που θα επιφέρει η κάθε του επιλογή συγκριτικά με τις υπόλοιπες. Με την παρουσίαση μερικών μεθευρετικών αλγορίθμων πολλαπλών στόχων θα γίνει σαφές παρακάτω ότι ένας από τους κύριους στόχους των μεθοδολογιών είναι η διατήρηση της διαφορετικότητας ώστε το μέτωπο Pareto να είναι όσο το δυνατόν πιο διευρυμένο γίνεται στον χώρο αναζήτησης.



Εικόνα 3.22. *Ορθή χρήση των πληροφοριών για μελέτη των αντισταθμίσεων μεταξύ πολλαπλών βέλτιστων λύσεων*

3.4.1. Περιορισμένη αναζήτηση πολλαπλών στόχων (Multi-objective tabu search, MOTS)

Η περιορισμένη αναζήτηση πολλαπλών στόχων (MOTS) αποτελεί επέκταση του κλασικού αλγορίθμου TS με σκοπό τον αποτελεσματικό χειρισμό πολλαπλών συναρτήσεων ενσωματώνοντας μηχανισμούς για την διαφοροποίηση της εξερεύνησης και την αναζήτηση ενός συνόλου Pareto βέλτιστων λύσεων (A. Baykasoglu et al., 2007). Προσθετικά στη λίστα Tabu του κλασικού αλγορίθμου TS, δημιουργείται μία λίστα Pareto (Pareto list) και μία λίστα υποψηφίων (ως μελλοντικές τρέχουσες) λύσεων (candidate list). Η λίστα Pareto χρησιμοποιείται για να αποθηκεύονται οι καλύτερες λύσεις που έχουν εντοπιστεί μέχρι στιγμής. Η λίστα υποψηφίων λειτουργεί ως δεξαμενή πολλά υποσχόμενων λύσεων που δεν περιέχονται (ακόμη) στη λίστα Pareto, αλλά δείχνουν δυνατότητες για μελλοντική εξερεύνηση των γειτονιών τους.

Αναλυτικά, τα βήματα του αλγορίθμου MOTS είναι:

1. **Ορισμός του μεγέθους *T* της λίστας Tabu και του κριτηρίου τερματισμού**. Ο αλγόριθμος σταματάει όταν επιτευχθεί προ-καθορισμένος αριθμός επαναλήψεων ή όταν η candidate list παραμένει ίδια (δεν μπορούν να βρεθούν άλλες μη κυριαρχούμενες λύσεις).
2. **Εκκίνηση με μία αρχική εφικτή λύση (seed)** η οποία μπορεί να δημιουργηθεί είτε τυχαία είτε χρησιμοποιώντας κάποια ευρετική κατασκευαστική μέθοδο.
3. **Ορισμός της γειτονιάς γύρω από την τρέχουσα λύση**. Αυτή η γειτονιά αντιπροσωπεύει όλες τις πιθανές εφικτές λύσεις που μπορούν να προκύψουν από την τρέχουσα λύση κάνοντας μικρές αλλαγές (εναλλαγή στοιχείων, αφαίρεση ή προσθήκη στοιχείων ή οποιαδήποτε άλλη σχετική τροποποίηση ανάλογα με τη φύση του προβλήματος). Όπως αναφέρθηκε και στον κλασικό αλγόριθμο TS, για μεγάλης κλίμακας ΠΔΟ είναι αδύνατον να δοκιμαστούν όλες οι εναλλαγές ή αφαιρέσεις/προσθήκες στοιχείων (πελατών) μεταξύ των διαδρομών λόγω των αμέτρητων πολλές φορές συνδυασμών, οπότε επιλέγεται ένας αριθμός γειτονικών λύσεων που θα δημιουργούνται με τυχαιότητα σε κάθε επανάληψη.
4. **Οι λύσεις της γειτονιάς που περιέχονται στη λίστα Tabu διαγράφονται** (ώστε να μην μπορούν να επιλεγούν ως candidate seed).
5. **Αξιολόγηση των γειτονικών λύσεων** σε όλους τους στόχους με χρήση των αντικειμενικών συναρτήσεων.

6. **Εύρεση των candidate seed λύσεων**, οι οποίες είναι οι λύσεις της γειτονιάς που δεν κυριαρχούνται από άλλες λύσεις της γειτονιάς, της Pareto list και της candidate list.
7. Όσες λύσεις εντός των Pareto και candidate list κυριαρχούνται από οποιαδήποτε γειτονική λύση διαγράφονται από τις λίστες αυτές.
8. Προσθήκη της τρέχουσας λύσης στην Pareto list.
9. Προσθήκη όλων των candidate seed λύσεων στην candidate list.
10. Προσθήκη της τρέχουσας λύσης στην λίστα Tabu. Αν η προσθήκη αυτή υπερβαίνει το μέγεθος *T* της λίστας, τότε αντικαθιστά την παλαιότερη λύση (first in, first out).
11. **Ορισμός μίας τυχαίας λύσης από τις candidate seed λύσεις ως τρέχουσα**. Αν δεν υπάρχει καμία candidate seed λύση, επιλογή της παλαιότερης λύσης από την candidate list ως τρέχουσα.
12. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 14.
13. Επανάληψη των βημάτων 3-12 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
14. **Επιστροφή του συνόλου των λύσεων της Pareto list.**

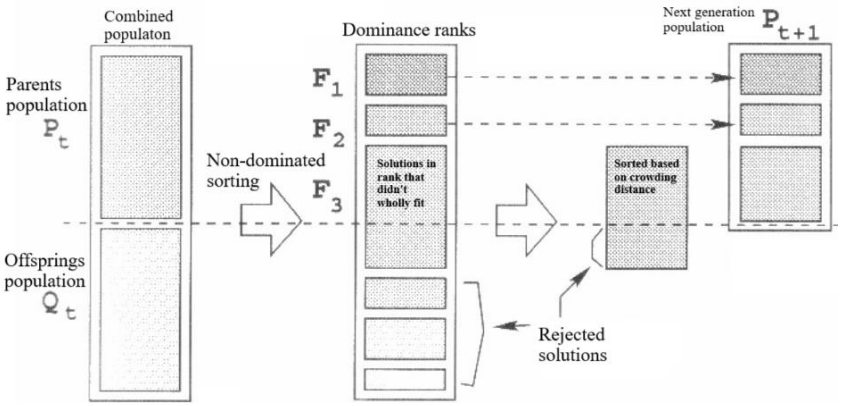
3.4.2. Γενετικός αλγόριθμος μη κυριαρχούμενης ταξινόμησης II (Non-dominated sorting genetic algorithm II, NSGA-II)

Ο αλγόριθμος NSGA-II ανήκει στην ευρύτερη κατηγορία των γενετικών αλγορίθμων πολλαπλών αντικειμενικών συναρτήσεων (multi-objective genetic algorithms, MOGA), είναι από τους πιο δημοφιλείς της κατηγορίας και χαρακτηρίζεται ως σημείο κατατεθέν όσον αφορά την ιστορική εξέλιξη των γενετικών αλγορίθμων από την εποχή της έμπνευσής τους. Οι δημιουργοί του (K. Deb et al., 2002) εμπνεύστηκαν έναν απλό και παράλληλα πρωτοποριακό αλγόριθμο που συνδυάζει υπάρχουσες μέχρι τότε στρατηγικές με ξεχωριστό και αποτελεσματικό τρόπο. Ο NSGA-II χρησιμοποιεί μια εξελιγμένη προσέγγιση γρήγορης μη κυριαρχούμενης ταξινόμησης (fast non-dominated sorting) για τη διατήρηση της διαφορετικότητας και την καθοδήγηση της εξελικτικής διαδικασίας προς το βέλτιστο μέτωπο Pareto. Συνεπώς, η επιλογή των ατόμων που θα αποτελέσουν τους γονείς της επόμενης γενιάς πραγματοποιείται κυρίως με βάση την Pareto κυριαρχία, ενώ οι τεχνικές διασταύρωσης δεν διαφέρουν από τον κλασικό GA. Ακόμη, σημαντικά διαφέρουν οι τεχνικές επιλογής γονέων και η διαδικασία δημιουργίας του πληθυσμού κάθε επόμενης γενιάς. Τέλος, προκειμένου να κατευθυνθεί η αναζήτηση των λύσεων προς περιοχές που δεν έχουν εξερευνηθεί, η ποιότητα που ανατίθεται σε κάθε άτομο του πληθυσμού επηρεάζεται και από το πόσο συνωστισμένη είναι η γειτονιά του συγκεκριμένου ατόμου.

Μία επισκόπηση, πριν την αναλυτικότερη παρουσίαση, των βημάτων του αλγορίθμου NSGA-II είναι:

1. **Ορισμός των παραμέτρων**: Μέγεθος πληθυσμού, τεχνική επιλογής γονέων, μηχανισμός διασταύρωσης, μηχανισμός μετάλλαξης, πιθανότητα μετάλλαξης.
2. **Ορισμός του κριτηρίου τερματισμού**, που μπορεί να είναι ο αριθμός γενεών (επαναλήψεων), η ποιότητα και η διαφορετικότητα (αριθμός μοναδικών λύσεων) των λύσεων στο μέτωπο Pareto ή κριτήρια υπολογιστικών πόρων και χρόνου.
3. **Δημιουργία του αρχικού πληθυσμού** εφικτών λύσεων με χρήση κάποιας ευρετικής μεθόδου ή με τυχαιότητα.
4. **Αξιολόγηση των λύσεων του πληθυσμού** σε όλους τους στόχους.
5. **Μη κυριαρχούμενη ταξινόμηση** του πληθυσμού, που οδηγεί σε πολλαπλά επίπεδα κυριαρχίας (dominance ranks), σε ένα από τα οποία θα καταταγεί κάθε λύση του πληθυσμού.
6. **Υπολογισμός της απόστασης συνωστισμού** της κάθε λύσης.
7. **Επιλογή** γονέων βάσει της επιλεγμένης τεχνικής ή με συνδυασμό αυτών.
8. **Διασταύρωση** γονέων ανά ζεύγη για δημιουργία του πληθυσμού των απογόνων.
9. **Μετάλλαξη** ενός ποσοστού των απογόνων με χρήση της επιλεγμένης τεχνικής μετάλλαξης και βάσει της πιθανότητας μετάλλαξης.
10. **Αξιολόγηση των λύσεων του πληθυσμού των απογόνων** σε όλους τους στόχους**.**
11. **Συνένωση των πληθυσμών γονέων και απογόνων** για τη δημιουργία του συνδυασμένου πληθυσμού (combined population), που αποτελείται από 2*(μέγεθος πληθυσμού) λύσεις.
12. **Μη κυριαρχούμενη ταξινόμηση** του συνδυασμένου πληθυσμού και υπολογισμός της απόστασης συνωστισμού της κάθε λύσης.
13. **Δημιουργία του πληθυσμού της επόμενης γενιάς**, με επιλογή των λύσεων από τον συνδυασμένο πληθυσμό ανά επίπεδο κυριαρχίας, από τα πρώτα προς τα τελευταία επίπεδα. Προστίθενται όλες οι λύσεις των επιπέδων κυριαρχίας των οποίων το πλήθος των λύσεων «χωράει» ολόκληρο χωρίς να γίνει υπέρβαση του μεγέθους του πληθυσμού. Οι υπόλοιπες λύσεις θα προέλθουν από τις λύσεις του τελευταίου επιπέδου κυριαρχίας που δεν μπόρεσε να προστεθεί ολόκληρο, αφού ταξινομηθούν κατά φθίνουσα σειρά βάσει της απόστασης συνωστισμού, έως ότου συμπληρωθεί ο απαιτούμενος αριθμός λύσεων του πληθυσμού της επόμενης γενιάς.
14. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 16.

15. Επανάληψη των βημάτων 4-14 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
16. **Μη κυριαρχούμενη ταξινόμηση** του τελευταίου πληθυσμού μετά την επαναληπτική διαδικασία.
17. **Επιστροφή των λύσεων του πρώτου επιπέδου κυριαρχίας**, που θα είναι και το βέλτιστο μέτωπο Pareto.



Εικόνα 3.23. *Δημιουργία της επόμενης γενιάς από τον συνδυασμένο πληθυσμό*

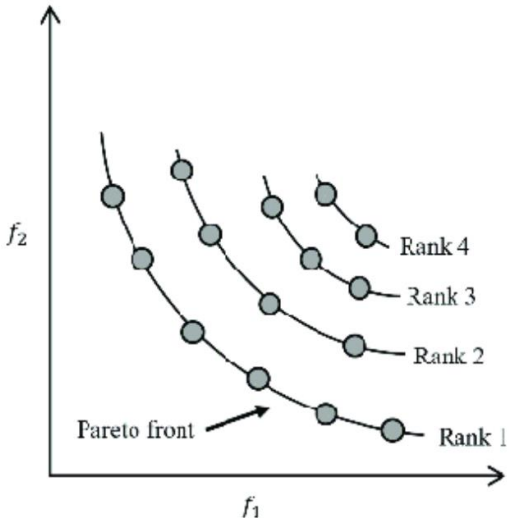
3.4.2.1. Μη κυριαρχούμενη ταξινόμηση (Non-dominated sorting)

Η μη κυριαρχούμενη ταξινόμηση κατηγοριοποιεί τα άτομα ενός πληθυσμού με βάση τις σχέσεις κυριαρχίας τους, σχηματίζοντας τελικά σύνολα ή μέτωπα (ranks) λύσεων που δεν κυριαρχούνται από καμία λύση του μετώπου όπου ανήκουν αλλά και των χειρότερων ποιοτικά μετώπων.

Όπως είδαμε και παραπάνω, *σε ένα πρόβλημα n αντικειμενικών συναρτήσεων (έστω όλες προς ελαχιστοποίηση), μία λύση A κυριαρχεί (ασθενώς) έναντι μίας λύσης B εάν* $f_i(A) \leq f_i(B)$, $\forall\, i \in \{1,2,...,n\}$ *και* $f_i(A) < f_i(B)$ *για κάποιο* $i \in \{1,2,...,n\}$. Αντίστοιχα, μία λύση *A* δεν μπορεί να θεωρηθεί χειρότερη από κάποια άλλη λύση *B* αν δεν ισχύει ο παραπάνω ορισμός, δηλαδή εάν δεν κυριαρχείται, είτε ασθενώς είτε αυστηρά, από την *B*.

Τα μη κυριαρχούμενα μέτωπα που θα προκύψουν μας εξασφαλίζουν ότι:

- Οι λύσεις στο πρώτο μέτωπο (rank 1) δεν κυριαρχούνται από καμία άλλη λύση στον πληθυσμό.
- Τα άτομα στο δεύτερο μέτωπο (rank 2) κυριαρχούνται μόνο από τα άτομα (τουλάχιστον ένα) του πρώτου μετώπου.
- Τα άτομα στο τρίτο μέτωπο (rank 3) κυριαρχούνται μόνο από τα άτομα (τουλάχιστον ένα) του πρώτου και δεύτερου μετώπου.
- Και ούτω καθεξής.



Εικόνα 3.24. *Απεικόνιση των μετώπων μετά από μη κυριαρχούμενη ταξινόμηση πληθυσμού*

Αλγοριθμικά, η διαδικασία που ακολουθείται είναι:

1. **Είσοδος**: Πληθυσμός P αποτελούμενος από N λύσεις, με κάθε μία να αξιολογείται από M αντικειμενικές συναρτήσεις.
2. **Αρχικοποίηση**: Δημιουργία μίας λίστας από κενά σύνολα (μέτωπα) $F_1, F_2, \ldots, F_k$.
3. **Έλεγχος κυριαρχίας**:
   - Για κάθε λύση p_i στον πληθυσμό:
     - Δημιουργία δύο μετρητών: S_i (μετρητής των λύσεων που κυριαρχούνται από την p_i) και D_i (σύνολο των λύσεων που κυριαρχούνται από την p_i).
     - Επαναληπτική σύγκριση όλων των υπόλοιπων λύσεων p_j με την p_i:
       - Αν η p_i κυριαρχεί έναντι της p_j, προσθήκη της p_j στο σύνολο D_i και $S_i = S_i + 1$.
   - Εύρεση των λύσεων που ανήκουν στο μέτωπο F_1:
     - Αν $S_i = 0$, τότε η p_i ανήκει στο πρώτο μέτωπο (δηλαδή δεν κυριαρχείται από καμία άλλη λύση του πληθυσμού). Προσθήκη στο F_1.
4. **Ανάθεση λύσεων στα υπόλοιπα μέτωπα**:
   - Αρχικοποίηση ενός μετρητή $i = 1$.
   - Για όσο $|F_i| \neq 0$:
     - Δημιουργία μίας άδειας λίστας Q (για αποθήκευση των λύσεων του επόμενου μετώπου).
     - Για κάθε λύση p_i στο σύνολο F_i:
       - Για κάθε λύση p_j στο σύνολο D_i:
         a. $S_j = S_j - 1$.
         b. Αν $S_j = 0$, προσθήκη της λύσης p_j στο Q.
     - $i = i + 1$
     - Αν $|Q| \neq 0$, προσθήκη στο σύνολο F_{i+1}.
5. **Έξοδος**: Επιστροφή της λίστας με τα συμπληρωμένα σύνολα $F_1, F_2, \ldots, F_k$.

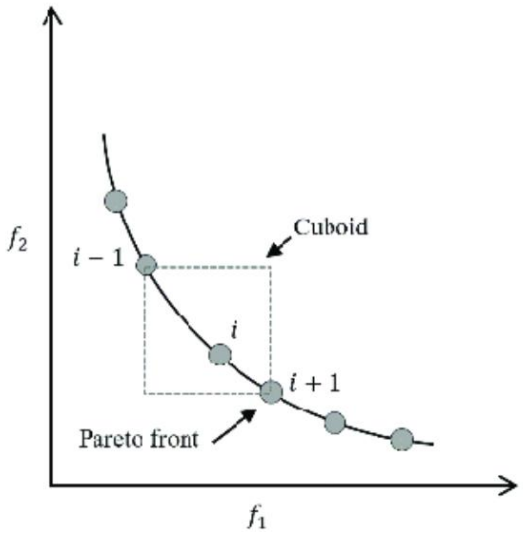
3.4.2.2. *Υπολογισμός απόστασης συνωστισμού*

Η απόσταση συνωστισμού (crowding distance) είναι ένα μέτρο που χρησιμοποιείται για την αξιολόγηση της ποικιλομορφίας των λύσεων σε έναν πληθυσμό **κατά μήκος του κάθε μη κυριαρχούμενου μετώπου**. Ποσοτικοποιεί την πυκνότητα των λύσεων στον χώρο λαμβάνοντας υπόψη τις αποστάσεις μεταξύ γειτονικών λύσεων βάσει της κάθε αντικειμενικής συνάρτησης. Μια λύση με μεγαλύτερη απόσταση συνωστισμού υποδηλώνει ότι βρίσκεται σε μια λιγότερο πυκνή περιοχή του χώρου αναζήτησης, η διατήρηση της οποίας πιθανόν να προάγει την ποικιλομορφία και εν τέλει την καλύτερη εξερεύνηση των σχέσεων μεταξύ των αντικρουόμενων στόχων για το πρόβλημα βελτιστοποίησης.

Αλγοριθμικά, η διαδικασία που ακολουθείται είναι:

1. **Είσοδος**: Λίστα με τα σύνολα F_1, F_2, …, F_k.
2. Αρχικοποίηση της απόστασης συνωστισμού κάθε λύσης p_i: $CD_{p_i} = 0$.
3. Για κάθε σύνολο F_i:
    - Για κάθε αντικειμενική συνάρτηση m:
        - Ταξινόμηση των λύσεων του συνόλου F_i με βάση την τιμή τους για την αντικειμενική συνάρτηση m.
        - Για την απόσταση συνωστισμού της πρώτης (p_1) και της τελευταίας ($p_{|Fi|}$) λύσης θέτουμε: $CD_{p_1} = CD_{p_{|Fi|}} = \infty$ (ή μία πολύ μεγάλη τιμή).
        - Για κάθε ενδιάμεση λύση p_j στην ταξινομημένη λίστα:
            - Προστίθεται στην απόσταση συνωστισμού CD_{p_j} η διαφορά των τιμών των δύο γειτονικών σε αυτήν λύσεων, αφού κανονικοποιηθεί βάσει των μέγιστων και ελάχιστων τιμών των λύσεων σε ολόκληρο τον πληθυσμό για τη συγκεκριμένη αντικειμενική συνάρτηση m: $CD_{p_j} = CD_{p_j} + \frac{f^m_{j+1} - f^m_{j-1}}{f^m_{max} - f^m_{min}}$.
4. **Έξοδος**: Αποστάσεις συνωστισμού για κάθε λύση των συνόλων εισόδου.

Το μέτρο του μεγέθους της απόστασης συνωστισμού επί της ουσίας ισούται με το μισό της περιμέτρου του ορθογωνίου παραλληλεπιπέδου που εσωκλείει τη λύση i, με τις δύο γειτονικές λύσεις να αποτελούν τις κορυφές του ορθογωνίου. Θα πρέπει επίσης να σημειωθεί ότι για κάθε λύση δεν είναι απαραίτητο οι ίδιες λύσεις $(i-1)$ και $(i+1)$ να είναι γειτονικές για όλες τις αντικειμενικές συναρτήσεις.



Εικόνα 3.25. *Υπολογισμός της απόστασης συνωστισμού*

3.4.2.3. Τεχνικές επιλογής γονέων (Parent selection)

Συνήθεις τεχνικές επιλογής γονέων του NSGA-II παρουσιάζονται παρακάτω:

- **Τεχνική της ρουλέτας (Roulette wheel selection):** Καθώς δεν είναι εύκολο σε προβλήματα πολλαπλών στόχων να θεωρηθεί μία λύση καλύτερη από κάποια άλλη, μπορεί να οριστεί από τον χρήστη μία συνάρτηση υπολογισμού του fitness score που λαμβάνει υπόψη το επίπεδο κυριαρχίας και την απόσταση συνωστισμού της κάθε λύσης. Έτσι, με συνδυασμό αυτών των δύο παραγόντων ορίζεται η ποιότητα της κάθε λύσης i ως $Fitness(i)=f(rank_i,-CD_i)$, όπου $rank_i$ το επίπεδο κυριαρχίας όπου ανήκει και CD_i η απόσταση συνωστισμού. Έτσι, η πιθανότητα επιλογής της ως γονέα θα είναι $P(i) = \frac{Fitness(i)}{\sum_{j=1}^{N} Fitness(j)}$, όπου N το μέγεθος του πληθυσμού. Ένας εναλλακτικός και πιο απλός τρόπος είναι η χρήση μόνο των αποστάσεων συνωστισμού, οπότε η πιθανότητα επιλογής της κάθε λύσης θα είναι $P(i)=\frac{CD(i)}{\sum_{j=1_{(CD\neq\infty)}}^{N} CD(j)}$. Αλγοριθμικά, η διαδικασία που ακολουθείται είναι η ίδια με την κλασική τεχνική της ρουλέτας που παρουσιάστηκε στους γενετικούς αλγορίθμους.
- **Δυαδική επιλογή διαγωνισμού (Binary tournament selection):** Η τεχνική BTS είναι αρκετά απλή αλλά αποτελεσματική, καθώς δίνει άμεσα προτεραιότητα σε λύσεις με βάση την κυριαρχία. Αλγοριθμικά, η διαδικασία που ακολουθείται είναι:

  1. Επιλογή 2 τυχαίων λύσεων από τον πληθυσμό.

  2. Επιλογή ως γονέα της λύσης που κυριαρχεί έναντι της άλλης. Αν δεν υπάρχει καθαρή κυριαρχία, επιλογή της λύσης με τη μεγαλύτερη απόσταση συνωστισμού.

3. Επανάληψη των βημάτων 1-2 έως ότου ληφθεί αριθμός γονέων ίσος με το μέγεθος του πληθυσμού.

- **Ελιτισμός (Elitism):** Η έννοια μας είναι ήδη γνωστή από τους γενετικούς αλγορίθμους. Μπορεί αντίστοιχα να εφαρμοστεί και στον αλγόριθμο NSGA-II, λαμβάνοντας υπόψη το fitness score των λύσεων όπως αυτό ορίζεται από τη συνάρτηση $Fitness(i)=f(rank_i,-CD_i)$ που είδαμε στην τεχνική της ρουλέτας. Επιλέγεται ένα ποσοστό (έστω 10%) του πληθυσμού ως «ελιτ» και αφορά τις λύσεις με τα υψηλότερα fitness scores. Οι λύσεις αυτές απομονώνονται και περνούν στην επόμενη γενιά ανέπαφες, χωρίς να γίνει διασταύρωση με κάποια άλλη λύση.

3.4.3. Εξελικτικός αλγόριθμος μετώπου Pareto με ανάθεση δύναμης II (Strength Pareto evolutionary algorithm II, SPEA-II)

Ο αλγόριθμος SPEA-II είναι ένας προηγμένος αλγόριθμος βελτιστοποίησης πολλαπλών στόχων που εισήχθη από τους Eckart Zitzler, Marco Laumanns και Lothar Thiele το 2001 ως μία βελτιωμένη έκδοση του προκατόχου του, SPEA. Χρησιμοποιεί έναν συνδυασμό ανάθεσης τιμών καταλληλότητας, εκτίμησης πυκνότητας (αντίστοιχη έννοια με τον συνωστισμό στον αλγόριθμο NSGA-II) και **διαδικασιών περικοπής (truncation)** ενός **συνόλου αρχειοθετημένων λύσεων (archive)** με σκοπό τη διατήρηση και εξέλιξη ενός μη κυριαρχούμενου συνόλου. Στα άτομα του πληθυσμού αποδίδονται τιμές **δύναμης (strength)** με βάση τις σχέσεις κυριαρχίας τους με άλλα άτομα και η ποιότητα των λύσεων αξιολογείται συνδυάζοντας τις **«καθαρές» τιμές καταλληλότητας (raw fitness)** με ένα **μέτρο πυκνότητας (density)** για την ενθάρρυνση τόσο της σύγκλισης όσο και της ποικιλομορφίας στο χώρο αναζήτησης. Μέσω της **περιβαλλοντικής επιλογής (environmental selection)**, ο αλγόριθμος διατηρεί μη κυριαρχούμενες λύσεις και διαχειρίζεται το μέγεθος του αρχείου για να εξισορροπήσει τις στρατηγικές εξερεύνησης και εκμετάλλευσης.

Μία επισκόπηση, πριν την αναλυτικότερη παρουσίαση, των βημάτων του αλγορίθμου SPEA-II, είναι:

1. **Ορισμός παραμέτρων:** Μέγεθος πληθυσμού, μηχανισμός διασταύρωσης, μηχανισμός μετάλλαξης, πιθανότητα μετάλλαξης, μέγεθος αρχείου.
2. **Ορισμός του κριτηρίου τερματισμού**, που μπορεί να είναι ο αριθμός γενεών (επαναλήψεων), η ποιότητα και η διαφορετικότητα (αριθμός μοναδικών λύσεων) των λύσεων στο μέτωπο Pareto ή κριτήρια υπολογιστικών πόρων και χρόνου.

3. **Δημιουργία του αρχικού πληθυσμού** εφικτών λύσεων με χρήση κάποιας ευρετικής μεθόδου ή με τυχαιότητα.
4. **Δημιουργία του συνόλου αρχειοθετημένων λύσεων**, που αρχικά είναι κενό.
5. **Ανάθεση τιμής καταλληλότητας** στις λύσεις του πληθυσμού και του αρχείου.
6. **Περιβαλλοντική επιλογή:** Δημιουργία του αρχείου της επόμενης γενιάς, που θα αποτελείται λύσεις του πληθυσμού και του αρχείου της τρέχουσας γενιάς.
7. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 13.
8. **Επιλογή γονέων από το νέο αρχείο** με χρήση δυαδικής επιλογής διαγωνισμού (BTS) για τη δημιουργία της δεξαμενής ζευγαρώματος (mating pool).
9. **Διασταύρωση** γονέων ανά ζεύγη για δημιουργία του πληθυσμού των απογόνων.
10. **Μετάλλαξη** ενός ποσοστού των απογόνων με χρήση της επιλεγμένης τεχνικής μετάλλαξης και βάσει της πιθανότητας μετάλλαξης.
11. **Αντικατάσταση του πληθυσμού με τον πληθυσμό των απογόνων.**
12. Επανάληψη των βημάτων 5-7-11 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
13. **Επιστροφή των μη κυριαρχούμενων λύσεων του αρχείου (βέλτιστο μέτωπο Pareto).**

3.4.3.1. Ανάθεση τιμής καταλληλότητας (Fitness assignment)

Η διαδικασία ανάθεσης τιμής καταλληλότητας περιλαμβάνει βήματα για τον υπολογισμό της δύναμης, της «καθαρής» φυσικής κατάστασης και της πυκνότητας της κάθε λύσης.

Αλγοριθμικά, η διαδικασία που ακολουθείται είναι:

1. **Είσοδος:** Πληθυσμός P αποτελούμενος από N λύσεις και αρχείο $\overline{P}$ με μέγεθος $\overline{N}$, και κάθε λύση αξιολογείται από M αντικειμενικές συναρτήσεις.
2. Για κάθε λύση i στον πληθυσμό και στο αρχείο:
   - Υπολογισμός της δύναμης $S_i = |\{j \mid j \in P \cup \overline{P} \text{ και } j \text{ κυρ. από } i\}|$, που αντιπροσωπεύει τον αριθμό των λύσεων που κυριαρχούνται από την i.
   - Υπολογισμός «καθαρής» τιμής καταλληλότητας $R_i = \sum_{j \in P \cup \overline{P},\ i \text{ κυρ. από } j} S_j$.
   - Αξιολόγηση της i βάσει όλων των αντικειμενικών συναρτήσεων και υπολογισμός της Ευκλείδειας απόστασης (στον M-διάστατο χώρο) $d\{i, j\} = \sqrt{\sum_{k=1}^{M} [obj(i,k) - obj(j,k)]^2}$ της λύσης i με κάθε άλλη λύση j.
   - Αύξουσα ταξινόμηση των αποστάσεων $d\{i, j\}$.
   - Υπολογισμός της πυκνότητας $D_i = \frac{1}{\sigma_i^k + 2}$, όπου σ_i^k η απόσταση της λύσης i από το k στοιχείο της ταξινομημένης λίστας των αποστάσεων με τις υπόλοιπες λύσεις (ο k-πλησιέστερος γείτονας), και συνήθως επιλέγεται $k = \sqrt{N + \overline{N}}$.

3. Ανάθεση τιμής καταλληλότητας σε κάθε λύση ως $F_i = R_i + D_i$.
4. **Έξοδος:** Τιμές καταλληλότητας για κάθε λύση του πληθυσμού και του αρχείου.

3.4.3.2. Περιβαλλοντική επιλογή (Environmental selection)

Η διαδικασία που ακολουθείται για τη δημιουργία του αρχείου της επόμενης γενιάς είναι:

1. **Είσοδος:** Πληθυσμός P_t αποτελούμενος από N λύσεις και αρχείο $\overline{P}_t$ με μέγεθος $\overline{N}$, και κάθε λύση έχει ήδη αξιολογηθεί από M αντικειμενικές συναρτήσεις.
2. Εύρεση και αντιγραφή όλων των μη κυριαρχούμενων λύσεων στο αρχείο της επόμενης γενιάς: $\overline{P_{t+1}} = \{i \mid i \in P_t \cup \overline{P}_t \text{ και } F_i < 1\}$.
3. Έλεγχος του μεγέθους του αρχείου:
   - Αν $|\overline{P_{t+1}}| = \overline{N}$, τέλος της διαδικασίας επιλογής, επιστροφή του αρχείου $\overline{P_{t+1}}$.
   - Αν $|\overline{P_{t+1}}| < \overline{N}$, επιλογή των καλύτερων (βάσει των τιμών καταλληλότητας) $\overline{N} - |\overline{P_{t+1}}|$ κυριαρχούμενων λύσεων από το προηγούμενο αρχείο $\overline{P}_t$ και τον πληθυσμό P_t και προσθήκη στο $\overline{P_{t+1}}$. Έπειτα επιστροφή του αρχείου $\overline{P_{t+1}}$.
   - Αν $|\overline{P_{t+1}}| > \overline{N}$, εφαρμογή της διαδικασίας περικοπής (truncation):
     - Επαναληπτική εύρεση της λύσης i που απέχει την μικρότερη Ευκλείδεια απόσταση από κάποια άλλη λύση j (ουσιαστική η λύση που προάγει λιγότερο την ποικιλομορφία) και διαγραφή της, έως ότου $|\overline{P_{t+1}}| = \overline{N}$.
     - Επιστροφή του αρχείου $\overline{P_{t+1}}$.
4. **Έξοδος:** Αρχείο επόμενης γενιάς που περιέχει έναν συνδυασμό μη κυριαρχούμενων λύσεων και δυνητικά κυριαρχούμενων, με μέγεθος $\overline{N}$.

3.5. Τεχνικές μέτρησης απόδοσης (Performance metrics)

Στη βελτιστοποίηση πολλαπλών στόχων, η αξιολόγηση της απόδοσης απαιτεί τη χρήση κατάλληλων μετρήσεων (performance indicators) για την ποσοτικοποίηση διαφόρων πτυχών της ποιότητας των λύσεων. Οι μετρήσεις αυτές παρέχουν πληροφορίες για τη σύγκλιση (convergence), την ποικιλομορφία (diversity) και την κάλυψη (coverage) του χώρου αναζήτησης που επιτυγχάνεται από έναν αλγόριθμο βελτιστοποίησης. Θα αναφερθούμε σε δύο μόνο από τις πολλές τεχνικές που έχουν προταθεί (C. Audet et al., 2020), στον υπερόγκο και στην κατανομή, που βοηθούν στην αξιολόγηση της σύγκλισης, ποικιλομορφίας και κατανομής ενός μη κυριαρχούμενου συνόλου λύσεων. Η χρήση των τεχνικών αυτών επιτρέπει στους χρήστες να συγκρίνουν διαφορετικούς αλγόριθμους, να επιλέγουν τους καταλληλότερους και να ρυθμίζουν τις παραμέτρους τους για τη καλύτερη δυνατή απόδοση.

3.5.1. Υπερόγκος μετώπου (Hypervolume indicator)

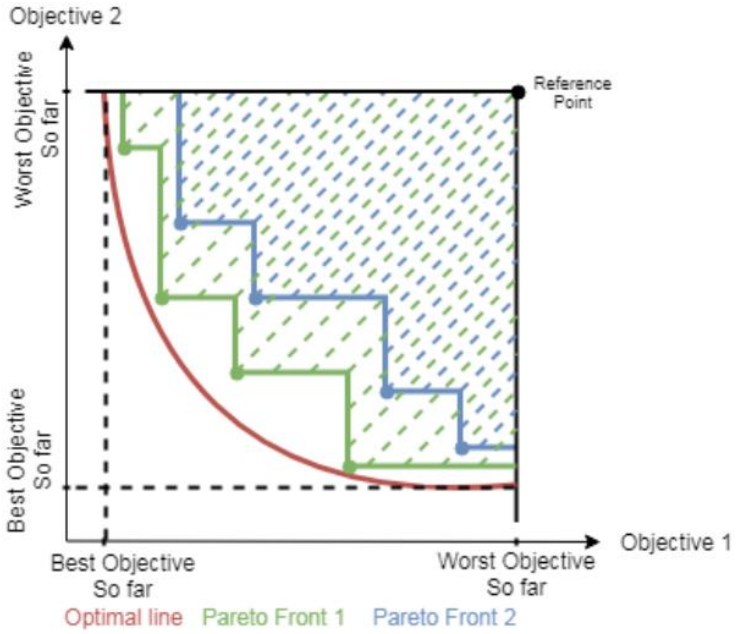
Ο υπερόγκος αποτελεί τεχνική μέτρησης της ποιότητας του μετώπου Pareto σε έναν n-διάστατο χώρο, καθώς μας ενδιαφέρει τόσο η μορφή του όσο και η κατανομή του στο χώρο των λύσεων. Ο λόγος που η τεχνική αυτή έγινε δημοφιλής τα τελευταία χρόνια είναι ότι ο

υπερόγκος αξιολογεί ταυτόχρονα την εγγύτητα στην επιθυμητή μορφή αλλά και την ποικιλομορφία των λύσεων του μετώπου. Αυτό πρακτικά σημαίνει ότι όταν ένα σύνολο λύσεων κυριαρχεί σε ένα άλλο, τότε αυτομάτως και ο υπερόγκος του πρώτου θα είναι μεγαλύτερος από αυτόν του δεύτερου. Για τη μέτρηση του υπερόγκου πρέπει να οριστεί ένα σημείο αναφοράς (reference point) με βάση το οποίο υπολογίζεται η τιμή του και συγκρίνονται τα διαφορετικά μέτωπα.

Ο υπερόγκος ενός μετώπου Pareto, όπως ορίστηκε από τους Zitzler και Thiene (1998), για ένα σύνολο μη κυριαρχούμενων λύσεων $PF \in R^C$, συμβολίζεται με $I_H(PF)$, εξαρτάται από το σημείο αναφοράς $r = (r_1, r_2, ..., r_C)' \in R^C$ και δίνεται από τον τύπο:

$$I_H(PF, r) = \lambda \left(\bigcup_{s \in PF} space(s, r) \right)$$

όπου το $space(s, r) = \{v \in R^C \mid r \prec v \preccurlyeq s\}$ αποτελεί τον αντικειμενικό χώρο, που παίρνει την μορφή παραλληλογράμμου αν οι αντικειμενικές συναρτήσεις είναι δύο, αλλιώς μιλάμε για χώρο μεγαλύτερων διαστάσεων. Ο αντικειμενικός χώρος αποτελεί την περιοχή στην οποία ανήκουν όλα τα διανύσματα του προβλήματος $v \in R^C$, τα οποία κυριαρχούνται ασθενώς από τα στοιχεία $s \in PF$ και όλα αυτά κυριαρχούν έναντι του σημείου αναφοράς r. Η μέτρηση Lebesgue (Lebesgue measure λ) εκφράζει τον όγκο ενός n-διάστατου Ευκλείδειου χώρου.



Εικόνα 3.26. *Απεικόνιση υπερόγκου μετώπων για δύο αντικειμενικές συναρτήσεις προς ελαχιστοποίηση*

Ο υπερόγκος του μετώπου Pareto εξαρτάται σε πολύ μεγάλο βαθμό από το σημείο αναφοράς και όταν το βέλτιστο μέτωπο είναι άγνωστο, τότε δεν μπορεί να οριστεί ένα τέτοιο αντικειμενικό σημείο. Αυτό που προτείνεται από τους Zitler at al. (2007) και Lu et al. (2012) είναι να ορίζεται ως σημείο αναφοράς το σημείο «ναδίρ» του εξεταζόμενου μετώπου. Σημείο ναδίρ σε αυτές τις περιπτώσεις ορίζεται ως το διάνυσμα με τις χειρότερες τιμές σε όλες τις αντικειμενικές συναρτήσεις προς βελτιστοποίηση. Ακόμη, για να διασφαλιστεί ότι το σημείο αναφοράς κυριαρχεί σε όλα τα υπόλοιπα σημεία συνήθως προστίθεται μία μικρή τιμή (έστω +1) σε κάθε διάσταση των αντικειμενικών συναρτήσεων.

3.5.2. Κατανομή μετώπου (Spacing metric)

Η μέτρηση της κατανομής (Audet et. al, 2020) χρησιμεύει για την αξιολόγηση της ποικιλομορφίας και της κατανομής των λύσεων στο μέτωπο Pareto. Παρέχει χρήσιμες πληροφορίες για τον βαθμό στον οποίο οι λύσεις κατανέμονται στον χώρο αναζήτησης. Με τον υπολογισμό της μέσης Ευκλείδειας απόστασης ανά ζεύγη μεταξύ των μη κυριαρχούμενων λύσεων, προσφέρεται μία εικόνα για την ποικιλομορφία του συνόλου αυτού. Μια χαμηλότερη τιμή σημαίνει και ένα πιο ομοιόμορφα κατανεμημένο σύνολο λύσεων, υποδεικνύοντας την καλύτερη εξερεύνηση του χώρου αναζήτησης.

Η κατανομή ενός μετώπου Pareto, για ένα σύνολο μη κυριαρχούμενων λύσεων $PF \in R^C$, συμβολίζεται με *SM* και δίνεται από τον τύπο:

$$SM = \frac{1}{\frac{N(N-1)}{2}} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \left\| x_i - x_j \right\|$$

όπου $N=|PF|$ το πλήθος των μη κυριαρχούμενων λύσεων, x_i και x_j τα διανύσματα των τιμών των λύσεων *i και j* στους στόχους του προβλήματος, αντίστοιχα.

3.6. Προτεινόμενοι υβριδικοί αλγόριθμοι

Οι υβριδικοί μεθευρετικοί αλγόριθμοι δημιουργούνται από την συγχώνευση πολλαπλών μεθοδολογιών βελτιστοποίησης, συνδυάζοντας έτσι τα δυνατά σημεία διαφορετικών τεχνικών για την αποτελεσματικότερη αντιμετώπιση πολύπλοκων προβλημάτων βελτιστοποίησης. Έτσι, μπορούν να προσφέρουν ένα ευέλικτο και προσαρμόσιμο πλαίσιο ικανό να αντιμετωπίσει ένα ευρύ φάσμα χαρακτηριστικών και περιορισμών του κάθε προβλήματος, επιφέροντας δυνητικά καλύτερα αποτελέσματα ή γρηγορότερη σύγκλιση συγκριτικά με το αν εφαρμοζόταν ένας μόνο μεθευρετικός αλγόριθμος.

Παρακάτω θα ακολουθήσουν συγκεκριμένες προτάσεις υβριδικών προσεγγίσεων για την επίλυση προβλημάτων πολλαπλών αντικειμενικών συναρτήσεων. Σκοπός είναι η αξιολόγηση της επίδρασης του συνδυασμού των τεχνικών αυτών, κάτι που θα αναλυθεί λεπτομερώς στο τελευταίο κεφάλαιο, αυτό των εφαρμογών.

3.6.1. 1ος Υβριδικός αλγόριθμος: Συνδυασμός αλγορίθμων SPEA-II, Tabu Search

Η πρόταση αφορά τη χρήση οποιουδήποτε κατασκευαστικού αλγορίθμου για την δημιουργία του αρχικού πληθυσμού εφικτών λύσεων. Ο πληθυσμός αυτός στη συνέχεια θα αποτελέσει την πρώτη γενιά και θα βελτιωθεί με χρήση του SPEA-II, εφαρμόζοντας επιπρόσθετα περιορισμένη αναζήτηση σε ένα μέρος των λύσεων της κάθε γενιάς του εξελικτικού αλγορίθμου. Έτσι, ένα μέρος του πληθυσμού κάθε γενιάς θα ακολουθεί την κλασική λογική της επιλογής γονέων, διασταυρώσεων και μεταλλάξεων και στα υπόλοιπα άτομα του πληθυσμού θα εφαρμόζεται περιορισμένη αναζήτηση.

Τα βήματα του προτεινόμενου αλγορίθμου περιλαμβάνουν:

1. **Ορισμός παραμέτρων:** Μέγεθος πληθυσμού, μηχανισμός διασταύρωσης, μηχανισμός μετάλλαξης, πιθανότητα μετάλλαξης, μέγεθος αρχείου, κριτήριο τερματισμού (για SPEA-II), μέγεθος της λίστας tabu, κριτήριο τερματισμού (για TS).
2. **Δημιουργία του αρχικού πληθυσμού** εφικτών λύσεων.
3. **Δημιουργία του συνόλου αρχειοθετημένων λύσεων**, που αρχικά είναι κενό.
4. **Ανάθεση τιμής καταλληλότητας** στις λύσεις του πληθυσμού και του αρχείου.
5. **Περιβαλλοντική επιλογή:** Δημιουργία του αρχείου της επόμενης γενιάς, που θα αποτελείται λύσεις του πληθυσμού και του αρχείου της τρέχουσας γενιάς.
6. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού (SPEA-II)**. Αν ναι, βήμα 12.
7. **Χωρισμός του αρχείου της επόμενης γενιάς** σε δύο (έστω ίσα) μέρη.
8. Στο πρώτο μέρος του αρχείου**:**
    a. **Επιλογή γονέων** με χρήση δυαδικής επιλογής διαγωνισμού (BTS) για τη δημιουργία της δεξαμενής ζευγαρώματος.
    b. **Διασταύρωση** γονέων ανά ζεύγη για δημιουργία του μισού πληθυσμού των απογόνων.
    c. **Μετάλλαξη** ενός ποσοστού των απογόνων με χρήση της επιλεγμένης τεχνικής μετάλλαξης και βάσει της πιθανότητας μετάλλαξης.
9. Στο δεύτερο μέρος του αρχείου**:**

a. **Δημιουργία λίστας tabu** για κάθε λύση.
b. **Εφαρμογή περιορισμένης αναζήτησης σε κάθε λύση** έως ότου ικανοποιηθεί το κριτήριο τερματισμού (για TS).
c. **Αποθήκευση των τελικών λύσεων** (είτε βελτιωμένων βάσει του fitness είτε όχι) στον πληθυσμό των απογόνων.

10. **Αντικατάσταση του πληθυσμού με τον πληθυσμό των απογόνων.**
11. Επανάληψη των βημάτων 4-6-10 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
12. **Επιστροφή του αρχείου (βέλτιστο μέτωπο Pareto),** που θα περιέχει τις λύσεις που δεν κυριαρχούνται μεταξύ τους και από καμία άλλη λύση του τελευταίου πληθυσμού.

3.6.2. 2ος Υβριδικός αλγόριθμος: Συνδυασμός αλγορίθμων NSGA-II, Simulated annealing

Η πρόταση αφορά την χρήση οποιουδήποτε κατασκευαστικού αλγορίθμου για την δημιουργία του αρχικού πληθυσμού εφικτών λύσεων. Ο πληθυσμός αυτός στη συνέχεια θα αποτελέσει την πρώτη γενιά και θα βελτιωθεί με χρήση του NSGA-II, εφαρμόζοντας επιπρόσθετα προσομοιωμένη ανόπτηση σε ένα μέρος των λύσεων της κάθε γενιάς του γενετικού αλγορίθμου. Έτσι, ένα μέρος του πληθυσμού κάθε γενιάς θα ακολουθεί την κλασική λογική της επιλογής γονέων, διασταυρώσεων και μεταλλάξεων και στα υπόλοιπα άτομα του πληθυσμού θα εφαρμόζεται προσομοιωμένη ανόπτηση.

Τα βήματα του προτεινόμενου αλγορίθμου περιλαμβάνουν:

1. **Ορισμός των παραμέτρων**: Μέγεθος πληθυσμού, τεχνική επιλογής γονέων, μηχανισμός διασταύρωσης, μηχανισμός μετάλλαξης, πιθανότητα μετάλλαξης, κριτήριο τερματισμού (για NSGA-II), αρχική θερμοκρασία, διαδικασία «ψύξης», κριτήριο τερματισμού (για SA).
2. **Δημιουργία του αρχικού πληθυσμού** εφικτών λύσεων.
3. **Αξιολόγηση των λύσεων του πληθυσμού** σε όλους τους στόχους.
4. **Χωρισμός του πληθυσμού** σε δύο (έστω ίσα) μέρη**.**
5. Στο πρώτο μέρος του πληθυσμού**:**
    a. **Μη κυριαρχούμενη ταξινόμηση**, που οδηγεί σε πολλαπλά επίπεδα κυριαρχίας (dominance ranks), σε ένα από τα οποία θα καταταγεί κάθε λύση.
    b. **Υπολογισμός της απόστασης συνωστισμού** της κάθε λύσης.
    c. **Επιλογή** γονέων βάσει της επιλεγμένης τεχνικής ή με συνδυασμό αυτών.

d. **Διασταύρωση** γονέων ανά ζεύγη για δημιουργία του πληθυσμού των απογόνων.
e. **Μετάλλαξη** ενός ποσοστού των απογόνων με χρήση της επιλεγμένης τεχνικής μετάλλαξης και βάσει της πιθανότητας μετάλλαξης.

6. Στο δεύτερο μέρος του πληθυσμού:
   a. **Αρχικοποίηση θερμοκρασίας** για κάθε λύση.
   b. **Εφαρμογή προσομοιωμένης ανόπτησης σε κάθε λύση έως** ότου ικανοποιηθεί το κριτήριο τερματισμού (για SA).
   c. **Αποθήκευση των τελικών λύσεων** (βελτιωμένων ή μη) στον πληθυσμό των απογόνων. Μία λύση θεωρείται βελτιωμένη και γίνεται αποδεκτή όταν κυριαρχεί (ασθενώς ή αυστηρά) έναντι της αρχικής.
7. **Αξιολόγηση των λύσεων του πληθυσμού των απογόνων** σε όλους τους στόχους**.**
8. **Συνένωση των πληθυσμών γονέων και απογόνων** για τη δημιουργία του συνδυασμένου πληθυσμού (combined population), που αποτελείται από 2*(μέγεθος πληθυσμού) λύσεις.
9. **Μη κυριαρχούμενη ταξινόμηση** του συνδυασμένου πληθυσμού και υπολογισμός απόστασης συνωστισμού της κάθε λύσης.
10. **Δημιουργία του πληθυσμού της επόμενης γενιάς** (βλέπε κεφ. NSGA-II)
11. **Έλεγχος ικανοποίησης του κριτηρίου τερματισμού**. Αν ναι, βήμα 12.
12. Επανάληψη των βημάτων 3-10 μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.
13. **Μη κυριαρχούμενη ταξινόμηση** του τελευταίου πληθυσμού μετά την επαναληπτική διαδικασία.
14. **Επιστροφή των λύσεων του πρώτου επιπέδου κυριαρχίας**, που θα είναι και το βέλτιστο μέτωπο Pareto.

Κεφάλαιο 4. Εφαρμογές

Στην παρούσα εργασία θα ασχοληθούμε με την επίλυση, με χρήση της γλώσσας προγραμματισμού Python σε περιβάλλον PyCharm, των δύο παρακάτω προβλημάτων, κάθε ένα από τα οποία εμπλέκει πάνω από μία παραλλαγές των απλών ΠΔΟ:

- **Εφαρμογή 1η:** ΠΔΟ με χρονικά παράθυρα, περιορισμούς χωρητικότητας οχημάτων, χρόνων μετάβασης και πολλαπλές αντικειμενικές συναρτήσεις (Multi-Objective Capacity-Time-Constrained VRP with time window) όπου οι στόχοι είναι

(ακολουθώντας και τις ήδη ορισμένες παραμέτρους και μεταβλητές της μαθηματικής μοντελοποίησης στο Κεφ. 2.1.1.) :

a. Ελαχιστοποίηση της συνολικής διανυόμενης απόστασης:

   $Min \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} \cdot x_{ij}^{k})$

b. Ελαχιστοποίηση της ανισορροπίας φορτίων μεταξύ των οχημάτων:

   $Min \sum_{k=1}^{m} | \sum_{i=1}^{n} q_i \cdot y_i^k - \frac{1}{m} \sum_{i=1}^{m} \sum_{i=1}^{n} q_i \cdot y_i^k |$

c. Ελαχιστοποίηση του συνολικού χρόνου αναμονής σε τοποθεσίες πελατών:

   $Min \sum_{i=1}^{m} \sum_{1}^{n} [max(0, e_i - \alpha_i^k) \cdot y_i^k]$ όπου α_i^k ο χρόνος άφιξης του οχήματος k στον πελάτη i και e_i η χρονική στιγμή που ανοίγει το χρονικό παράθυρο εξυπηρέτησης του πελάτη i.

- **Εφαρμογή 2η:** ΠΔΟ με περιορισμό στον χρόνο μετάβασης, περιορισμούς χωρητικότητας και πολλαπλές αντικειμενικές συναρτήσεις (Multi-Objective Capacity-Time-Constrained VRP) όπου οι στόχοι είναι:

a. Ελαχιστοποίηση της συνολικής διανυόμενης απόστασης.

b. Ελαχιστοποίηση του συνολικού χρόνου ταξιδιού.

   $Min \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} (\tau_{ij} \cdot x_{ij}^{k})$

c. Ελαχιστοποίηση του αριθμού των οχημάτων.

Οι πρόσθετοι κοινοί περιορισμοί που διέπουν και τα δύο παραπάνω προβλήματα είναι:

1. Κάθε πελάτης πρέπει να εξυπηρετείται μία φορά, από ένα και μόνο όχημα.
2. Μία λύση θεωρείται αποδεκτή μόνο αν έχουν εξυπηρετηθεί όλοι οι πελάτες.
3. Όλα τα οχήματα του στόλου είναι ομογενή, έχουν δηλαδή την ίδια χωρητικότητα.
4. Η διαδρομή του κάθε οχήματος ξεκινάει και τελειώνει στην αποθήκη.

Βάσει της παραπάνω περιγραφής τα δεδομένα που χρειαζόμαστε για την επίλυση των προβλημάτων είναι η χωρητικότητα των οχημάτων, το πλήθος των πελατών, η γεωγραφική θέση, η ζήτηση και ο χρόνος εξυπηρέτησης του καθενός. Για το πρώτο πρόβλημα θα χρειαστούμε επίσης και το χρονικό παράθυρο εξυπηρέτησης του κάθε πελάτη. Και στα δύο προβλήματα ο μέγιστος χρόνος ταξιδιού (max route time) για το κάθε όχημα θα είναι η χρονική στιγμή που «κλείνει» το χρονικό παράθυρο της αποθήκης.

Στην ιστοσελίδα https://www.sintef.no/projectweb/top/vrptw/ παρέχονται δεδομένα για διάφορες κατανομές πελατών στον χώρο. Πιο συγκεκριμένα, επιλέχθηκαν τα δεδομένα για τα προβλήματα R1_2_1 και C1_2_1 για αριθμό πελατών = 200.

Τα δύο αυτά προβλήματα θα επιλυθούν ως ακολούθως:

Εφαρμογή 1η: Επίλυση R1_2_1 με χρήση του αλγορίθμου SPEA-II και στη συνέχεια με χρήση του 1ου προτεινόμενου υβριδικού αλγορίθμου για σύγκριση των αποτελεσμάτων.

Εφαρμογή 2η: Επίλυση C1_2_1 (χωρίς χρονικά παράθυρα) με χρήση του αλγορίθμου NSGA-II και στη συνέχεια με χρήση του 2ου προτεινόμενου υβριδικού αλγορίθμου για σύγκριση των αποτελεσμάτων.

Στο κάθε αρχείο περιέχονται οι παράμετροι του προβλήματος και ο πελάτης με δείκτη 0 αντιστοιχεί στην αποθήκη.

Αναλυτικό παράδειγμα (15 πρώτοι πελάτες) για ένα αρχείο παρουσιάζεται παρακάτω:

C1_2_1.TXT

Αρχείο Επεξεργασία Προβολή

c1_2_1

Μέγιστος αριθμός οχημάτων

VEHICLE NUMBER — Χωρητικότητα: CAPACITY

50	200					

CUSTOMER	Γεωγραφική θέση		Ζήτηση	Χρονικό παράθυρο		Χρόνος εξυπηρέτησης
CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	70	70	0	0	1351	0
1	33	78	20	750	809	90
2	59	52	20	553	602	90
3	10	137	30	147	219	90
4	4	28	10	616	661	90
5	25	26	20	128	179	90
6	86	37	10	478	531	90
7	1	109	10	616	680	90
8	6	135	40	351	386	90
9	32	79	20	655	721	90
10	24	26	20	219	271	90
11	86	36	20	384	443	90
12	95	35	10	196	252	90
13	63	50	10	364	416	90
14	100	106	10	567	626	90
15	99	112	20	846	911	90

Εικόνα 4.1. *Παράδειγμα δεδομένων σε ένα αρχείο*

4.1. Εφαρμογή 1η: Επίλυση προβλήματος R1_2_1

4.1.1. Κώδικας εφαρμογής αλγορίθμου SPEA-II

4.1.1.1. Εισαγωγή δεδομένων

```
import copy
import pandas as pd
import numpy as np
import random
import math
from scipy.spatial import distance_matrix
from functions import *

file path = 'R1 2 1.txt'
df1 = pd.read csv(file path, delim whitespace=True, header=None, skiprows=4, nrows=1)
 , vehicle capacity = df1.iloc[0]

customers_df = pd.read_csv(file_path, delim_whitespace=True, header=None, names=['CUST NO.',
'XCOORD.', 'YCOORD.', 'DEMAND', 'READY TIME', 'DUE DATE', 'SERVICE TIME'], skiprows=8)
indexes = customers_df['CUST NO.'].tolist()  # Λίστα με τους δείκτες των πελατών
```

```
demands = customers_df['DEMAND'].tolist()  # Λίστα τη ζήτηση του κάθε πελάτη
service_times = customers_df['SERVICE TIME'].tolist()  # Λίστα με το χρόνο εξυπηρέτησης του
κάθε πελάτη

x_coordinates = customers_df['XCOORD.'].tolist()
y_coordinates = customers_df['YCOORD.'].tolist()
coordinates = [(a, b) for a, b in zip(x_coordinates, y_coordinates)]  # Λίστα με ζεύγη
συντεταγμένων (X, Y) πελατών
distances = distance_matrix(coordinates, coordinates)  # Πίνακας αποστάσεως μεταξύ των πελατών
travel_times = distances  # Πίνακας χρόνων μετάβασης μεταξύ των πελατών (ταυτίζεται με τον
πίνακα αποστάσεων)

time_windows_A = customers_df['READY TIME'].tolist()
time_windows_B = customers_df['DUE DATE'].tolist()
time_windows = [(a, b) for a, b in zip(time_windows_A, time_windows_B)]  # Λίστα με χρονικά
παράθυρα (α, β) πελατών

max_route_time = max(time_windows_B)  # Μέγιστος χρόνος διαδρομής για κάθε όχημα

sorted_indexes = [index for _, index in sorted(zip(time_windows_B[1:], indexes[1:]))]  #
Ταξινόμηση των πελατών με βάση το β χρονικό παράθυρο
intervals_and_indexes = []
num_intervals = 3  # Δημιουργία num_intervals διαστημάτων
intervals, customer_indexes = split_windows_B_to_intervals_and_find_indexes(sorted_indexes,
time_windows_B, num_intervals)
intervals_and_indexes.append(customer_indexes)

customer_indexes_lists = []
number_of_lists_to_make = 300 # Από κάθε λίστα θα δημιουργηθεί μία αρχική λύση
while len(customer_indexes_lists) < number_of_lists_to_make:
    new_list = sorted_indexes[:]  # Αντιγραφή της ταξινομημένης λίστας
    for i in range(len(customer_indexes) - 1):  # Για κάθε interval
        first_customer_in_interval = customer_indexes[i]  # Ο πρώτος πελάτης στη λίστα
sorted_indexes με παράθυρο β εντός του i interval
        first_customer_in_next_interval = customer_indexes[i + 1]  # Ο πρώτος πελάτης στη
λίστα sorted_indexes με παράθυρο β εντός του i + 1 interval

        if random.random() < 0.4:  # Με πιθανότητα 40%, εναλλαγή (swap) των θέσεων δύο τυχαίων
πελατών
            index1, index2 = random.sample(range(first_customer_in_interval,
first_customer_in_next_interval), 2)
            new_list[index1], new_list[index2] = new_list[index2], new_list[index1]

    if new_list != sorted_indexes and new_list not in customer_indexes_lists:
        customer_indexes_lists.append(new_list)
```

Μέσω του παραπάνω κώδικα γίνεται ανάγνωση των δεδομένων από txt σε pandas dataframes και αποθήκευση αυτών σε μεταβλητές και λίστες. Στη συνέχεια γίνεται ταξινόμηση των πελατών με βάση το β χρονικό τους παράθυρο. Τα χρονικά παράθυρα β χωρίζονται σε διαστήματα (intervals) ώστε να καταταγεί κάθε πελάτης της ταξινομημένης λίστας σε ένα από αυτά. Βάσει αυτού, γίνεται αναδιάταξη της ταξινομημένης λίστας με πολλές (πιθανές για κάθε διάστημα με πιθανότητα 40%) 1-1 εναλλαγές μεταξύ πελατών που ανήκουν στο ίδιο διάστημα (άρα έχουν κοντινά β χρονικά παράθυρα), με σκοπό τη δημιουργία πολλών, λίγο τροποποιημένων λιστών, από την ταξινομημένη λίστα sorted_indexes, που στη συνέχεια θα χρησιμοποιηθούν για τη δημιουργία του αρχικού πληθυσμού (μεγέθους 300 λύσεων).

4.1.1.2. Συνάρτηση split_windows_B_to_intervals_and_find_indexes

```
def split_windows_B_to_intervals_and_find_indexes(sorted_indexes, time_windows_B,
num_intervals):
    interval_width = max(time_windows_B) / num_intervals  # Πλάτος κάθε ίσου διαστήματος
    intervals = []  # Αρχικοποίηση λίστας με διαστήματα
    indexes_where_intervals_change = [0]  # Ο πρώτος πελάτης στη sorted_list

    for i in range(num_intervals):
```

```
        interval_start = interval_width * i  # Αρχή i διαστήματος
        interval_end = interval_width * (i + 1)  # Τέλος i διαστήματος

        interval_end = min(interval_end,
                           max(time_windows_B))  # Για αποφυγή δημιουργίας λανθασμένου
τελευταίου διαστήματος
        intervals.append((interval_start, interval_end))  # Προσθήκη του διαστήματος στη λίστα
intervals

        for position_in_sorted_list, customer_index in enumerate(sorted_indexes):
            # Όταν βρεθεί ο (πρώτος) πελάτης του οποίου το B παράθυρο > interval_end ()
            if time_windows_B[customer_index] >= interval_end:
                # ! Μας ενδιαφέρει η θέση του πελάτη αυτού στη sorted_list !
                indexes_where_intervals_change.append(position_in_sorted_list)
                break
    return intervals, indexes_where_intervals_change
```

4.1.1.3. Δημιουργία αρχικού πληθυσμού

```
initial_population = []  # Αρχικοποίηση πληθυσμού λύσεων
for customer_indexes_list in customer_indexes_lists:
    routes, total_demands, total_waiting_times, total_travel_times, total_distances =
generate_new_feasible_solution(customer_indexes_list[:], distances, travel_times,
service_times, time_windows, demands, max_route_time, vehicle_capacity)
    initial_population.append(routes)  # Αποθήκευση κάθε λύσης στον πληθυσμό

print(f"Solutions in initial population: {len(initial_population)}")
```

Κάθε λύση του αρχικού πληθυσμού δημιουργείται με χρήση της συνάρτησης generate_new_solution για κάθε διαφορετική λίστα customer_indexes_list.

4.1.1.4. Συνάρτηση δημιουργίας (εφικτής) αρχικής λύσης generate_new_solution

```
def generate_new_solution(customer_indexes, distances, travel_times, service_times,
time_windows, demands, max_route_time, vehicle_capacity):
    start = 0  # Ο δείκτης της αποθήκης
    routes = []  # Λίστα όπου θα αποθηκεύονται οι διαδρομές των οχημάτων
    total_demands = []  # Συνολική ζήτηση που κάλυψε το κάθε όχημα
    total_waiting_times = []  # Συνολικός χρόνος αναμονής του κάθε οχήματος
    total_travel_times = []  # Συνολικός χρόνος ταξιδιού του κάθε οχήματος
    total_distances = []  # Συνολική απόσταση που διένυσε το κάθε όχημα

    while customer_indexes:  # Όσο η λίστα δεν είναι άδεια, άρα υπάρχει πελάτης που δεν έχει
εξυπηρετηθεί
        vehicle_route = [start]  # Δημιουργία νέας διαδρομής οχήματος, που ξεκινάει από την
αποθήκη
        current_time = 0  # Αρχικοποίηση χρόνου ταξιδιού
        current_load = 0  # Αρχικοποίηση του φορτίου του οχήματος
        vehicle_waiting_time = 0  # Αρχικοποίηση χρόνου αναμονής
        vehicle_travel_time = 0  # Αρχικοποίηση χρόνου ταξιδιού
        vehicle_distance_traveled = 0  # Αρχικοποίηση διανυόμενης απόστασης

        current_customer = start
        for next_customer in customer_indexes[:]:  # Επαναληπτικό πέρασμα των πελατών της
λίστας customer_indexes
            if demands[next_customer] + current_load > vehicle_capacity:
                continue  # Παραβίαση χωρητικότητας οχήματος, ο πελάτης παραλείπεται

            distance_to_travel = distances[current_customer][next_customer]  # Απόσταση
(τρέχων πελάτης, επόμενος)
            time_to_travel = travel_times[current_customer][
                next_customer]  # Χρόνος μετάβασης (τρέχων πελάτης, επόμενος)
            arrival_time = current_time + time_to_travel  # Υπολογισμός χρονικής στιγμής
άφιξης
            earliest_start_time, latest_start_time = time_windows[next_customer]

            waiting_time = max(0, earliest_start_time - arrival_time)  # Υπολογισμός χρόνου
αναμονής
            time_at_location = waiting_time + service_times[next_customer]  # Συνολικός χρόνος
στην τοποθεσία του πελάτη

            # Έλεγχος ικανοποίησης χρονικού παραθύρου και μη παραβίασης του μέγιστου χρόνου
ταξιδιού
            if arrival_time <= latest_start_time and (current_time + time_to_travel +
time_at_location + travel_times[next_customer][start] <= max_route_time):
```

```
                vehicle waiting time += waiting time
                vehicle travel time += time to travel + time at location
                vehicle distance traveled += distance to travel
                current time = arrival time + time at location
                current_load += demands[next_customer]  # Αύξηση του φορτίου του οχήματος κατά τη ζήτηση του πελάτη

                vehicle route.append(next customer)  # Πρόσθεση του πελάτη που εξυπηρετήθηκε στη διαδρομή
                current customer = next customer  # Ο πελάτης που μόλις εξυπηρετήθηκε γίνεται ο "τρέχων" πελάτης
                customer_indexes.remove(next_customer)  # Διαγραφή του πελάτη που εξυπηρετήθηκε
            else:  # Παραβίαση χρονικού παραθύρου ή max route time, ο πελάτης παραλείπεται
                continue

        vehicle_route.append(start)  # Προσθήκη της αποθήκης στο τέλος της διαδρομής
        vehicle_distance_traveled += distances[current_customer][
            start]  # Πρόσθεση απόστασης (τελευταίος πελάτης, αποθήκη)
        vehicle travel time += travel times[current customer][start]  # Πρόσθεση χρόνου μετάβασης (τελευταίος πελάτης, αποθήκη)

        # Προσθήκη όλων των χαρακτηριστικών της διαδρομής του τρέχοντος οχήματος στις λίστες προς επιστροφή
        routes.append(vehicle route)
        total demands.append(current load)
        total_waiting_times.append(vehicle_waiting_time)
        total travel times.append(vehicle travel time)
        total distances.append(vehicle distance traveled)

        if not customer indexes:  # Δεν υπάρχει μη-εξυπηρετημένος πελάτης
            break
    # Επιστροφή όλων των διαδρομών των οχημάτων, που αποτελούν μια ολοκληρωμένη λύση
    return routes, total_demands, total_waiting_times, total_travel_times, total_distances
```

4.1.1.5. Κώδικας αλγορίθμου SPEA-II

```
population = copy.deepcopy(initial population)  # Αρχικοποίηση πληθυσμού
max generations = 100 # Ορισμός αριθμού γενεών
mutation_rate = 0.4 # Πιθανότητα μετάλλαξης
archive_size = len(population) # Ορισμός μεγέθους αρχείου
archive = []  # Δημιουργία του αρχικά κενού αρχείου

for generation in range(max generations):
    print(f"Generation {generation + 1}")
    solutions_objectives_values = []
    for solution in (population + archive):  # Υπολογισμός των τιμών στις 3 αντικειμενικές συναρτήσεις για κάθε λύση
        result = evaluate solution(solution, distances, travel times, service times, time windows, demands, vehicle capacity, max route time)
        objective1, objective2, objective3 = result['Total Distance Travelled'], result['Deviation from Mean Load'], result['Total Waiting Time']
        solutions_objectives_values.append((objective1, objective2, objective3))  # Δημιουργία tuple με τις τιμές των objectives

    strengths = calculate solutions strengths(solutions objectives values)  # Υπολογισμός S(i)
    raw_fitness_values = calculate_solutions_raw_fitness(strengths, solutions_objectives_values)  # Υπολογισμός R(i)
    densities = calculate_solutions_density(solutions_objectives_values)  # Υπολογισμός D(i)
    fitness values = calculate solutions total fitness(raw fitness values, densities) # Υπολογισμός F(i) = R(i) + D(i)

    terminate_algorithm = False
    if generation == max generations - 1:
        terminate algorithm = True

    next archive, terminate algorithm = environmental selection(population + archive, solutions_objectives_values, fitness_values, archive_size, terminate_algorithm)

    if terminate algorithm:  # Έλεγχος ικανοποίησης κριτηρίου τερματισμού
        archive = [solution for solution,   in next archive]
        break

    # Δημιουργία δεξαμενής ζευγαρώματος
    mating pool size = len(population)
    mating pool = binary tournament selection(next archive, fitness values, mating pool size)
    offsprings_population = []  # Αρχικοποίηση της λίστας με λύσεις-απογόνους
```

```
    for i in range(0, mating pool size, 2):  # Επαναληπτικό πέρασμα των λύσεων στη δεξαμενή
ζευγαρώματος ανά 2 λύσεις
        parent1, parent1 index = mating pool[i]  # Γονέας 1
        parent2, parent2 index = mating pool[i + 1]  # Γονέας 2

        # Διασταύρωση γονέων
        flag = False
        both offspring objectives = []
        offsprings = []
        tries = 0  # Προσπάθειες διασταύρωσης των γονέων για τη δημιουργία εφικτών (feasible)
λύσεων-απογόνων
        while not flag:  # Επανάληψη διασταύρωσης των γονέων μέχρι να δημιουργηθούν εφικτές
λύσεις-απόγονοι
            offsprings = customer position alignment crossover(parent1, parent2)  # Δημιουργία
δύο απογόνων
            both_offspring_objectives = []
            result1 = evaluate_solution(offsprings[0], distances, travel_times, service_times,
            time_windows, demands, vehicle_capacity, max_route_time) # Αξιολόγηση απογόνου 1
            result2 = evaluate solution(offsprings[1], distances, travel times, service times,
            time windows, demands, vehicle capacity, max route time) # Αξιολόγηση απογόνου 2
            if result1['feasible'] and result2['feasible']:  # Αν και δύο απόγονοι είναι
εφικτές λύσεις
                objective11, objective12, objective13 = result1['Total Distance Travelled'],
result1['Deviation from Mean Load'], result1['Total Waiting Time']
                both offspring objectives.append((objective11, objective12, objective13))
                objective21, objective22, objective23 = result2['Total Distance Travelled'],
result2['Deviation from Mean Load'], result2['Total Waiting Time']
                both offspring objectives.append((objective21, objective22, objective23))
                flag = True
            elif tries == 10:
                flag = True  # Δε βρέθηκαν εφικτές λύσεις, διακοπή της επανάληψης
        if tries != 10: # Βρέθηκαν εφικτές λύσεις
            for idx, objectives in enumerate(both_offspring_objectives):
                # Αν ο απόγονος κυριαρχείται από κάποιον γονέα, κρατάμε τον γονέα
                if dominates(solutions objectives values[parent1 index], objectives) and
parent1 not in next archive:
                    offsprings population.append(parent1)
                elif dominates(solutions objectives values[parent2 index], objectives) and
parent2 not in next_archive:
                    offsprings population.append(parent2)
                else:
                    offsprings population.append(offsprings[idx])# Αν ο απόγονος δεν
κυριαρχείται κανέναν από τους δύο γονείς, κρατάμε τον απόγονο
         else:
            offsprings_population.extend([parent1, parent2]) # Αν γίνουν tries σε αριθμό
προσπάθειες και δε βρεθούν δύο εφικτές λύσεις-απόγονοι, διατήρηση των γονέων
        if random.random() < mutation rate:  # Μετάλλαξη των δύο τελευταίων λύσεων-απογόνων με
πιθανότητα mutation rate
            for last_solutions_index in ([-1, -2]):
                flag = False
                while not flag:  # Επανάληψη μέχρι να προκύψει εφικτή μεταλλαγμένη λύση
                    mutated solution = mutate(offsprings population[last solutions index])  #
Μετάλλαξη της τελευταίας λύσης που προστέθηκε στον πληθυσμό
                    mutated solution eval = evaluate solution(mutated solution, distances,
travel_times, service_times, time_windows, demands, vehicle_capacity, max_route_time)
                    if mutated_solution_eval['feasible']:
                        flag = True
                        offsprings population[last solutions index] = mutated solution

    population = offsprings population # Πληθυσμός επόμενης γενιάς = πληθυσμός απογόνων
    archive = [solution for solution,  in next archive] # Ανανέωση αρχείου επόμενης γενιάς
```

Ως κριτήριο τερματισμού έχει οριστεί ο μέγιστος αριθμός των γενεών ή η επίτευξη αρχείου με archive_size σε αριθμό μη κυριαρχούμενες λύσεις. Η ικανοποίηση του δεύτερου κριτηρίου ελέγχεται εντός της συνάρτησης environmental_selection. Αφού τερματιστεί ο αλγόριθμος, επιστρέφεται το αρχείο ως το βέλτιστο μέτωπο Pareto. Ακολουθούν οι συναρτήσεις που χρησιμοποιούνται εντός του κώδικα του SPEA-II.

4.1.1.6. Συναρτήσεις αξιολόγησης λύσεων evaluate_route, evaluate_solution

```
def evaluate solution(solution, distances, travel times, service times, time windows, demands,
vehicle_capacity, max_route_time):
    total distance = 0
    total travel time = 0
    total waiting time = 0
    total demand fulfilled = 0
    feasible = True
    vehicle_loads = []
    for route in solution:  # Για κάθε διαδρομή στη λίστα solution
        route evaluation = evaluate route(route, distances, travel times, service times,
time windows, demands, vehicle capacity, max route time)
        if not route evaluation['feasible']:  # Αν έστω και μία διαδρομή της λύσης είναι
ανέφικτη, η λύση είναι ανέφικτη
            feasible = False
            break
        total distance += route evaluation['Distance travelled']
        total travel time += route evaluation['Travel time']
        total waiting time += route evaluation['Waiting time']
        total_demand_fulfilled += route_evaluation['Demand fulfilled']
        vehicle_loads.append(route_evaluation['Demand fulfilled'])
    # Υπολογισμός την ανισορροπίας φορτίων ως το άθροισμα των αποκλίσεων από το μέσο φορτίο
    num vehicles = len(solution)
    mean load = sum(vehicle loads) / num vehicles
    total deviation = sum([abs(load - mean load) for load in vehicle loads])
    # Επιστροφή όλων των υπολογισμένων παραμέτρων της λύσης
    return {
        'feasible': feasible,
        'Total Distance Travelled': total distance,
        'Total Travel Time': total travel time,
        'Total Waiting Time': total_waiting_time,
        'Number of Vehicles': len(solution),
        'Total Demand Fulfilled': total demand fulfilled,
        'Deviation from Mean Load': total deviation
    }
```

Για αξιολόγηση κάθε λύσης με χρήση της evaluate_solution χρησιμοποιείται επίσης η συνάρτηση evaluate_route για την αξιολόγηση κάθε διαδρομής εντός της λύσης μία προς μία.

```
def evaluate route(route, distances, travel times, service times, time windows, demands,
vehicle capacity, max route time):
    # Αρχικοποίηση όλων των παραμέτρων της διαδρομής που θα αξιολογηθεί
    total_distance = 0
    total travel time = 0
    total waiting time = 0
    total demand = 0
    current time = 0

    for i in range(len(route) - 1):  # Επαναληπτικό πέρασμα όλων των πελατών της διαδρομής
(route)
        from customer = route[i]
        to customer = route[i + 1]

        total_distance += distances[from_customer][to_customer]
        total demand += demands[to customer]
        travel time = travel times[from customer][to customer]
        arrival time = current time + travel time
        earliest_start, latest_start = time_windows[to_customer]
        if arrival_time > latest_start:  # Έλεγχος παραβίασης του χρονικού παραθύρου του
πελάτη i
            return {'feasible': False}  # Ανέφικτη διαδρομή

        waiting time = max(0, earliest start - arrival time)
        total_waiting_time += waiting_time
        total travel time += travel time + service times[to customer] + waiting time
        current time = total travel time

    if current time > max route time or total demand > vehicle capacity:  # Παραβίαση
χωρητικότητας ή μέγιστου χρόνου ταξιδιού
        return {'feasible': False}  # Ανέφικτη διαδρομή
    # Εφικτή διαδρομή, επιστροφή όλων των υπολογισμένων παραμέτρων της διαδρομής
    return {
        'feasible': True,
        'Distance travelled': total_distance,
```

```
        'Travel time': total travel time,
        'Waiting time': total waiting time,
        'Demand fulfilled': total demand
    }
```

4.1.1.7. Συναρτήσεις υπολογισμού strength, raw fitness, density, total fitness

```
def calculate_solutions_strengths(solutions_objectives_values):
    solutions strenghts = [0] * len(solutions objectives values)  # Αρχικοποίηση S(i)=0 για
κάθε λύση (πληθυσμός + αρχείο)
    # Σύγκριση κάθε λύσης με κάθε άλλη λύση για έλεγχο κυριαρχίας
    for i, solution_1 in enumerate(solutions_objectives_values):
        for j, solution_2 in enumerate(solutions_objectives_values):
            if dominates(solution 1, solution 2):  # Έλεγχος αν η λύση 2 κυριαρχείται από τη
λύση 1
                solutions strenghts[i] += 1
    return solutions_strenghts
def normalize list(list of numbers):
    normalized list = (list of numbers - np.min(list of numbers)) / (np.max(list of numbers) -
np.min(list of numbers) + 1)

    return normalized_list


def calculate solutions raw fitness(strengths, solutions objectives values):
    solutions raw fitness list = [0] * len(strengths)  # Αρχικοποίηση R(i)=0 για κάθε λύση
(πληθυσμός + αρχείο)

    for i in range(len(strengths)):  # R(i) = Άθροισμα των δυνάμεων S(j) όλων των λύσεων j που
κυριαρχούν έναντι της λύσης i
        solutions raw fitness list[i] = sum(strengths[j] for j in range(len(strengths)) if
                dominates(solutions objectives values[j], solutions objectives values[i]))
    # Επιστροφή των raw fitness των λύσεων

    return solutions_raw_fitness_list


def calculate solutions density(solutions objectives values):
    densities_list = []

    for solution index, solution i objectives in enumerate(solutions objectives values):
        distances to other solutions = []
        # Υπολογισμός όλων των αποστάσεων (στις 3 διαστάσεις των objectives) της λύσης
solution index με κάθε άλλη λύση j
        for j, other_solution_objectives in enumerate(solutions_objectives_values):
            if j != solution_index:  # Αποφυγή υπολογισμού της απόστασης της λύσης
solution index με τον εαυτό της (= 0)
                # για να μην επηρεαστεί η ταξινομημένη λίστα sorted distances
                distance = math.sqrt(
                    sum((x - y) ** 2 for x, y in zip(solution i objectives,
other_solution_objectives)))
                distances_to_other_solutions.append(distance)

        sorted distances = sorted(distances to other solutions)  # Αύξουσα ταξινόμηση των
αποστάσεων

        # Από τη θεωρία του SPEA-II γνωρίζουμε ότι k=sqrt(αριθμός λύσεων πληθυσμού + αρχείου)
        k_nearest_neighbor_index = int(math.sqrt(len(sorted_distances)))

        # Υπολογισμός πυκνότητας της λύσης με δείκτη solution index
        density = 1 / (sorted_distances[k_nearest_neighbor_index] + 2)
        densities_list.append(density)

    return densities list


def calculate_solutions_total_fitness(raw_fitness_values, densities):
    solutions total fitness list = [0] * len(raw fitness values)

    for i in range(len(raw fitness values)):
        solutions total fitness list[i] = raw fitness values[i] + densities[i]

    return solutions_total_fitness_list
```

4.1.1.8. Συνάρτηση περιβαλλοντικής επιλογής environmental_selection

```
def environmental selection(combined population, objectives values, fitness values,
archive_size, terminate_algorithm):

    non dominated solutions indexes = []  # Λύσεις που δεν κυριαρχούνται από καμία άλλη λύση
    dominated solutions indexes = []  # Λύσεις που κυριαρχούνται έστω και από μία λύση

    # Εύρεση όλων των λύσεων στον συνδυασμένο πληθυσμό (πληθυσμός + αρχείο) που δεν
κυριαρχούνται από καμία άλλη λύση
    for solution index, solution objectives in enumerate(objectives values):
        if is nondominated(solution objectives, objectives values):
            non dominated solutions indexes.append(solution index)  # Η λύση solution index
είναι μη-κυριαρχούμενη
        else:
            dominated_solutions_indexes.append(solution_index)  # Η λύση solution_index είναι
κυριαρχούμενη

    # Φθίνουσα ταξινόμηση των λιστών με τους δείκτες των λύσεων βάσει των fitness scores
    non dominated solutions indexes.sort(key=lambda index: fitness values[index],
reverse=True)
    dominated_solutions_indexes.sort(key=lambda index: fitness_values[index], reverse=True)

    # Δημιουργία του αρχείου της επόμενης γενιάς
    next archive = []
    # Προσθέτουμε τις μη-κυριαρχούμενες λύσεις (ξεκινώντας από τις καλύτερες βάσει fitness
score) στο επόμενο αρχείο
    for index in non dominated solutions indexes:
        is duplicate = False
        if len(next archive) < archive size:
            for solution,   in next archive:
                if combined_population[index] == solution:
                    is_duplicate = True
            if not is duplicate:
                next archive.append((combined population[index], index))
        else:  # Αν το επόμενο αρχείο συμπληρωθεί με archive size λύσεις, σταματάμε για να μη
χρειαστεί να γίνει περικοπή (truncation)
            break

    print(f"next archive unique NON-DOMINATED SOLUTIONS: {len(next archive)}")
    if len(next archive) == archive size:  # Αν συγκεντρωθούν archive size μη κυριαρχούμενες
λύσεις, τέλος αλγορίθμου
        return next_archive, True  # Επιστροφή των μετώπου μη κυριαρχούμενων λύσεων, και True

    # Αν όχι, συμπληρώνεται με τις καλύτερες κυριαρχούμενες λύσεις (έχει γίνει ήδη ταξινόμηση
των dominated solutions indexes)
    for index in dominated solutions indexes:  # Αν το επόμενο αρχείο δεν έχει συμπληρωθεί,
συμπλήρωση με κυριαρχούμενες από τρέχων πληθυσμό και αρχείο
        is_duplicate = False
        if len(next_archive) < archive_size:
            for solution,   in next archive:
                if combined population[index] == solution:
                    is duplicate = True
            if not is_duplicate:
                next archive.append((combined population[index], index))
        else:  # Αν το αρχείο συμπληρωθεί με archive size λύσεις, σταματάμε για να μη
χρειαστεί να γίνει περικοπή (truncation)
            break
    return next archive, terminate algorithm  # Επιστροφή επόμενου αρχείου (tuple με λύσεις,
δείκτες λύσεων) με μη κυριαρχούμενες και κυριαρχούμενες λύσεις, και False
```

4.1.1.9. Συναρτήσεις σύγκρισης λύσεων βάσει κυριαρχίας

```
def dominates(sol1 objectives, sol2 objectives):
    solution1 dominates solution2 = False

    for obj_index in range(len(sol1_objectives)):
        if sol1_objectives[obj_index] > sol2_objectives[obj_index]:
            return False
        elif sol1 objectives[obj index] < sol2 objectives[obj index]:
            solution1 dominates solution2 = True

    return solution1_dominates_solution2

def is nondominated(solution objectives, combined population objectives):
    for other solution objectives in combined population objectives:  # Σύγκριση της λύσης
solution με κάθε άλλη λύση
```

```
        if dominates(other solution objectives, solution objectives):
            return False  # Κυριαρχείται έστω και από μία λύση
    return True  # Δεν κυριαρχείται από καμία λύση, άρα είναι non-dominated
```

4.1.1.10. Συνάρτηση δυαδικής επιλογής διαγωνισμού: binary_tournament_selection

```
def binary_tournament_selection(next_archive, fitness_values, mating_pool_size):
    mating_pool = []  # Αρχικοποίηση δεξαμενής ζευγαρώματος

    for   in range(mating pool size):  # Επαναληπτικά επιλογή δύο τυχαίων λύσεων από το
next archive
        random_solution_1, random_index_1 = random.choice(next_archive)
        random_solution_2, random_index_2 = random.choice(next_archive)

        if fitness values[random index 1] < fitness values[random index 2]:
            mating pool.append((random solution 1, random index 1))
        else:
            mating_pool.append((random_solution_2, random_index_2))
    return mating_pool  # Λίστα με tuples (λύση, index της λύσης στο επόμενο αρχείο)
```

4.1.1.11. Συνάρτηση διασταύρωσης customer_position_alignment_crossover

```
def customer position alignment crossover(parent1, parent2):
    # Συνάρτηση για εύρεση της θέσης (όχημα και δείκτης) του πελάτη customer στη λύση parent
    def find customer position(parent, customer):
        for i, route in enumerate(parent):
            if customer in route:
                return i, route.index(customer)
    # Συνάρτηση για αφαίρεση ενός τυχαίου πελάτη (customer to align) από τον γονέα parent1 από
οπουδήποτε βρίσκεται στις διαδρομές, και προσθήκη ξανά, αλλά στη θέση και στο όχημα που ο
ίδιος πελάτης βρίσκεται στην λύση parent2. Επιστρέφεται η τροποποιημένη λύση parent1 ως
απόγονος.
    def align customer position(parent1, parent2):
        offspring = copy.deepcopy(parent1)

        while True:
            customer_to_align = random.choice([c for route in parent1 for c in route if c!=0])
            target_route_index, target_position = find_customer_position(parent2,
customer to align)

            if target route index < len(offspring):
                if target_position < len(offspring[target_route_index]):
                    for route in offspring:
                        if customer_to_align in route:
                            route.remove(customer to align)
                            break
                    offspring[target route index].insert(target position, customer to align)
                    break
                else:
                    continue
            else:
                continue
        return offspring
    # Διασταύρωση των γονέων για τη δημιουργία offspring1, offspring2
    offspring1 = align_customer_position(parent1, parent2)
    offspring2 = align customer position(parent2, parent1)

    return offspring1, offspring2
```

4.1.1.12. Συνάρτηση μετάλλαξης mutate_solution

```
def mutate(solution):
    mutated solution = copy.deepcopy(solution)

    # Τυχαία επιλογή μεταξύ 1-1 ανταλλαγής, 2-2 ανταλλαγής, Τυχαίας επανατοποθέτησης
    mutation_type = random.choice(["1-1_exchange", "2-2_exchange", "Random relocate"])

    if mutation type == "1-1 exchange":
        route index = random.randint(0, len(mutated solution) - 1)
        route = mutated solution[route index]
        if len(route) < 4: # Έλεγχος ότι υπάρχουν 2 τουλάχιστον πελάτες
            return mutated_solution
        customer1_index, customer2_index = random.sample(range(1, len(route) - 1), 2)
        route[customer1 index], route[customer2 index] = route[customer2 index],
route[customer1 index]

    elif mutation type == "2-2 exchange":
        route_index = random.randint(0, len(mutated_solution) - 1)
```

```
        route = mutated_solution[route_index]
        if len(route) < 6: # Έλεγχος ότι υπάρχουν 4 τουλάχιστον πελάτες
            return mutated_solution
        customers = random.sample(range(1, len(route) - 1), 4)
        customers.sort()
        route[customers[0]], route[customers[2]] = route[customers[2]], route[customers[0]]
        route[customers[1]], route[customers[3]] = route[customers[3]], route[customers[1]]

    elif mutation_type == "Random_relocate":
        if len(mutated_solution) < 2: # Έλεγχος ότι υπάρχουν 2 τουλάχιστον διαδρομές
            return mutated_solution
        route1_index, route2_index = random.sample(range(len(mutated_solution)), 2)
        route1 = mutated_solution[route1_index]
        route2 = mutated_solution[route2_index]
        if len(route1) < 3 or len(route2) < 3: # Έλεγχος ύπαρξης 1 τουλάχιστον πελάτη
            return mutated_solution
        customer_index = random.randint(1, len(route1) - 2)
        customer = route1.pop(customer_index)  # Αφαίρεση από route1
        insertion_index = random.randint(1, len(route2) - 2)
        route2.insert(insertion_index, customer)  # Επανατοποθέτηση στη διαδρομή route2

    return mutated_solution
```

4.1.1.13. Συναρτήσεις μέτρησης υπερόγκου και κατανομής

Αφού τερματιστεί ο αλγόριθμος SPEA-II θα έχουμε μία λίστα με μη κυριαρχούμενες λύσεις archive. Αυτές θα αξιολογηθούν σε όλα τα objectives, θα κανονικοποιηθούν οι τιμές αυτών και θα χρησιμοποιηθούν για τον υπολογισμό των μετρήσεων υπερόγκου και κατανομής του συνόλου των λύσεων.

Για τον υπολογισμό της κατανομής του μετώπου:

```
def normalize_objectives(objectives_tuples):
    objective1_values = [obj[0] for obj in objectives_tuples]
    objective2_values = [obj[1] for obj in objectives_tuples]
    objective3_values = [obj[2] for obj in objectives_tuples]

    normalized_obj1 = normalize_list(objective1_values)
    normalized_obj2 = normalize_list(objective2_values)
    normalized_obj3 = normalize_list(objective3_values)

    normalized_objectives = list(zip(normalized_obj1, normalized_obj2, normalized_obj3))
    return normalized_objectives

def calculate_spacing_metric(normalized_objectives):
    num_solutions = len(normalized_objectives)
    sum_distances = 0.0

    for i in range(num_solutions):
        for j in range(i + 1, num_solutions):
            distance = np.linalg.norm(np.array(normalized_objectives[i]) -
np.array(normalized_objectives[j])) # Ευκλείδεια νόρμα των 3-tuples
            sum_distances += distance
    spacing_metric = sum_distances/(num_solutions * (num_solutions - 1)/2)

    return spacing_metric

data = []
for solution in archive:  # Υπολογισμός των τιμών στις 3 αντικειμενικές συναρτήσεις για κάθε
λύση
    result = evaluate_solution(solution, distances, travel_times, service_times, time_windows,
demands, vehicle_capacity, max_route_time)
    objective1, objective2, objective3 = result['Total Distance Travelled'], result['Deviation
from Mean Load'], result['Total Waiting Time']
    data.append((objective1, objective2, objective3))  # Δημιουργία tuple με τις τιμές των
objectives

normalized_objectives = normalize_objectives(data)
spacing_metric = calculate_spacing_metric(normalized_objectives)

print("Spacing metric: ", spacing_metric)
```

Στη συνέχεια για τον υπολογισμό του υπερόγκου θα χρησιμοποιηθεί η βιβλιοθήκη pymoo και ως reference_point το διάνυσμα με τις χειρότερες τιμές (+1) σε όλες τις αντικειμενικές συναρτήσεις για τις λύσεις του μετώπου.

```
from pymoo.indicators.hv import HV
non dominated solutions = np.array(normalized objectives)
worst values = np.max(non dominated solutions, axis=0)
reference_point = worst_values + 1 # + 1 σε κάθε διάσταση

ind = HV(ref point=reference point)
print("Hypervolume indicator: ", ind(non_dominated_solutions))
```

Τέλος, γίνεται μία 3D οπτικοποίηση του μετώπου Pareto με χρήση των βιβλιοθηκών matplotlib, numpy και scipy.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata

x = [t[0] for t in data]
y = [t[1] for t in data]
z = [t[2] for t in data]
points = np.array(list(zip(x, y, z)))

xi, yi = np.meshgrid(np.linspace(min(x), max(x), 100), np.linspace(min(y), max(y), 100))
zi = griddata(points[:, :2], points[:, 2], (xi, yi), method='linear')

fig = plt.figure(figsize=(8, 6))
ax = fig.add subplot(111, projection='3d')
ax.plot_surface(xi, yi, zi, cmap='viridis', edgecolor='none', alpha=0.7, label='Pareto Front
Surface')

ax.scatter3D(x, y, z, s=30, c='blue')
ax.set xlabel('Total Distance Travelled')
ax.set ylabel('Deviation from Mean Load')
ax.set_zlabel('Total Waiting Time')
plt.show()
```

4.1.2. Αποτελέσματα εφαρμογής αλγορίθμου SPEA-II

Τα αποτελέσματα παρουσιάζονται όπως ορίστηκε στον κώδικα παραπάνω για μέγιστο αριθμό γενεών = 100 (πρώτο κριτήριο τερματισμού), πιθανότητα μετάλλαξης 40% (μεγάλη πιθανότητα καθώς αποτελεί έναν έμμεσο τρόπο τοπικής αναζήτησης), μέγεθος πληθυσμού = μέγεθος αρχείου = μέγεθος δεξαμενής ζευγαρώματος = 300 λύσεις. Τονίζεται επίσης ότι σε περίπτωση που ληφθεί αρχείο next_archive με πλήθος μη κυριαρχούμενων λύσεων ίσο με το μέγεθος του αρχείου, η διαδικασία τερματίζεται (δεύτερο κριτήριο τερματισμού). Τα αποτελέσματα φαίνονται παρακάτω (ανά 5 γενιές) για ένα τρέξιμο του προγράμματος:

Generation 5
next_archive unique NON-DOMINATED SOLUTIONS: 38
Generation 10
next_archive unique NON-DOMINATED SOLUTIONS: 76
Generation 15
next_archive unique NON-DOMINATED SOLUTIONS: 88
Generation 20
next_archive unique NON-DOMINATED SOLUTIONS: 153
Generation 25
next_archive unique NON-DOMINATED SOLUTIONS: 97
Generation 30

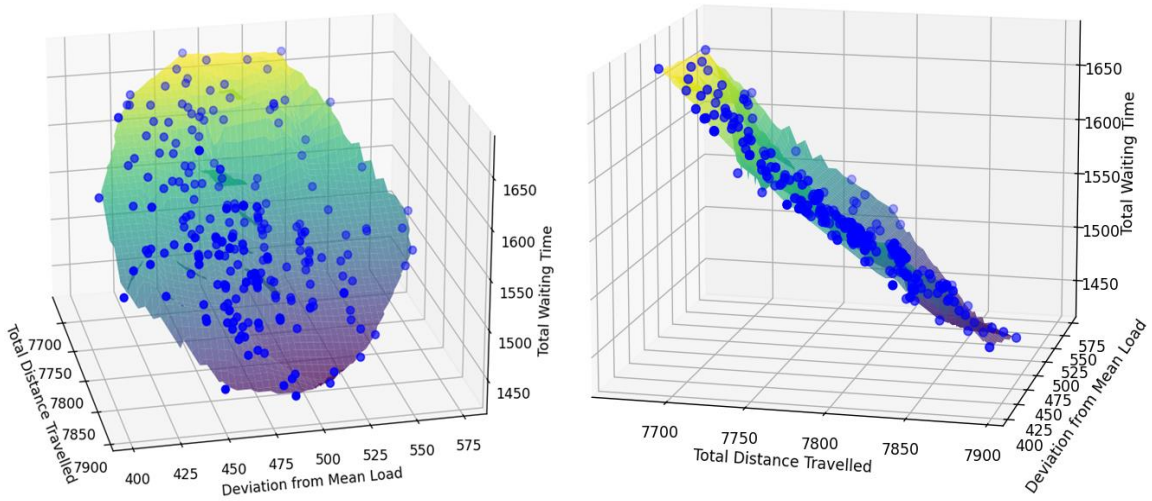
next_archive unique NON-DOMINATED SOLUTIONS: 89
Generation 35
next_archive unique NON-DOMINATED SOLUTIONS: 112
Generation 40
next_archive unique NON-DOMINATED SOLUTIONS: 76
Generation 45
next_archive unique NON-DOMINATED SOLUTIONS: 38
Generation 50
next_archive unique NON-DOMINATED SOLUTIONS: 69
Generation 55
next_archive unique NON-DOMINATED SOLUTIONS: 116
Generation 60
next_archive unique NON-DOMINATED SOLUTIONS: 199
Generation 65
next_archive unique NON-DOMINATED SOLUTIONS: 192
Generation 70
next_archive unique NON-DOMINATED SOLUTIONS: 198
Generation 75
next_archive unique NON-DOMINATED SOLUTIONS: 227
Generation 80
next_archive unique NON-DOMINATED SOLUTIONS: 207
Generation 85
next_archive unique NON-DOMINATED SOLUTIONS: 187
Generation 90
next_archive unique NON-DOMINATED SOLUTIONS: 299
Generation 91
next_archive unique NON-DOMINATED SOLUTIONS: 300

Το πρόγραμμα τερματίστηκε με ικανοποίηση του δεύτερου κριτηρίου τερματισμού μετά από 91 γενιές, με τις εξής μετρήσεις απόδοσης:

Spacing metric: 0.49339544803549

Hypervolume indicator: 6.831663292911731

Τέλος, η απεικόνιση του μετώπου Pareto (από δύο διαφορετικές γωνίες):



Εικόνα 4.1. *Το μέτωπο Pareto από την εφαρμογή SPEA-II στο πρόβλημα R1_2_1*

Για να αξιολογηθεί η αποτελεσματικότητα αλλά και η ευστάθεια του παραπάνω αλγορίθμου απαιτείται **στατιστική ανάλυση** των αποτελεσμάτων μετά από πολλά τρεξίματα. Παρακάτω παρουσιάζεται η μέση τιμή και η τυπική απόκλιση των γενεών που πραγματοποιήθηκαν πριν ικανοποιηθεί το κριτήριο τερματισμού, της κατανομής αλλά και του υπερόγκου του μετώπου Pareto, για 20 τρεξίματα του αλγορίθμου με τις παραμέτρους που ορίστηκαν παραπάνω:

Run #	Generations	Spacing metric	Hypervolume indicator
1	91	0,49339544803549	6,831663292911731
2	81	0,47554282989169	6,786086654074115
3	87	0,51539091199020	6,8808609643830305
4	82	0,42386790346822	7,217274096228489
5	71	0,46376965744800	6,92361458116937
6	76	0,52775438634598	6,622994381836919
7	88	0,44288041495822	6,933405230513468
8	64	0,52748080068030	7,167128841004617
9	87	0,20918135758688	7,3542728152585575
10	99	0,14863885910468	7,5889379275295274
11	94	0,48893247350245	6,718897993262209
12	67	0,44084299031742	6,807737547867917
13	89	0,34474030936237	7,677301524185894
14	77	0,42248700006374	6,7160629874446425
15	66	0,45295984261160	7,08098212135792
16	90	0,38342633929933	6,684899133493037
17	80	0,31238443398340	7,635433241899607
18	83	0,47121244749111	6,743896418551538
19	95	0,50400582765289	6,81690056296627
20	68	0,45179616763350	6,738494520300352
Μέση τιμή	**81,75**	**0,43**	**6,99**
Τυπική απόκλιση	**10,17**	**0,10**	**0,33**

4.1.3. Κώδικας εφαρμογής 1ου προτεινόμενου υβριδικού αλγορίθμου

Ο κώδικας του αλγορίθμου SPEA-II απαιτεί ελάχιστες τροποποιήσεις για προσθήκη της περιορισμένης αναζήτησης ενώ οι υπόλοιπες συναρτήσεις παραμένουν οι ίδιες. Απαιτείται να προστεθεί ο χωρισμός του επόμενου αρχείου σε δύο ίσα μέρη και η εφαρμογή του μεθευρετικού αλγορίθμου της περιορισμένης αναζήτησης σε κάθε λύση του δεύτερου μισού του επόμενου αρχείου, ενώ για το πρώτο μισό ακολουθείται ξανά η διαδικασία διασταυρώσεων και μεταλλάξεων του γενετικού αλγορίθμου.

4.1.3.1. Κώδικας SPEA-II + Tabu search

```
population = copy.deepcopy(initial population)  # Αρχικοποίηση πληθυσμού
max generations = 100
mutation rate = 0.4
archive_size = 300
archive = []  # Δημιουργία του αρχικά κενού αρχείου
```

```
for generation in range(max generations):
    print(f"Generation {generation + 1}")
    solutions objectives values = []
    for solution in (population + archive):  # Υπολογισμός των τιμών στις 3 αντικειμενικές συναρτήσεις για κάθε λύση
        result = evaluate_solution(solution, distances, travel_times, service_times, time windows, demands, vehicle capacity, max route time)
        objective1, objective2, objective3 = result['Total Distance Travelled'], result['Deviation from Mean Load'], result['Total Waiting Time']
        solutions objectives values.append((objective1, objective2, objective3))  # Δημιουργία tuple με τις τιμές των objectives

    strengths = calculate solutions strengths(solutions objectives values)  # Υπολογισμός S(i) των λύσεων
    raw fitness values = calculate solutions raw fitness(strengths, solutions_objectives_values)  # Υπολογισμός R(i) των λύσεων
    densities = calculate_solutions_density(solutions_objectives_values)  # Υπολογισμός D(i) των λύσεων
    fitness values = calculate solutions total fitness(raw fitness values, densities)  # Υπολογισμός F(i) = R(i) + D(i)

    terminate_algorithm = False
    if generation == max generations - 1:
        terminate algorithm = True

    next archive, terminate algorithm = environmental selection(population + archive, solutions_objectives_values, fitness_values, archive_size, terminate_algorithm)

    if terminate algorithm:  # Έλεγχος ικανοποίησης κριτηρίου τερματισμού
        archive = [solution for solution,   in next archive]
        break

    next_archive_first_half = next_archive[:len(next_archive) // 2] # Πρώτο μέρος αρχείου
    next archive second half = next archive[len(next archive) // 2:] # Δεύτερο μέρος αρχείου

    # Δημιουργία δεξαμενής ζευγαρώματος μεγέθους archive size/2
    mating pool size = archive size//2
    mating pool = binary tournament selection(next archive first half, fitness values, mating_pool_size)

    offsprings population = []  # Αρχικοποίηση της λίστας με λύσεις-απογόνους
    for i in range(0, mating pool size, 2):  # Επαναληπτικό πέρασμα των λύσεων στη δεξαμενή ζευγαρώματος ανά 2 λύσεις
        parent1, parent1_index = mating_pool[i]  # Γονέας 1
        parent2, parent2_index = mating_pool[i + 1]  # Γονέας 2

        # Διασταύρωση γονέων
        flag = False
        both_offspring_objectives = []
        offsprings = []
        tries = 0  # Προσπάθειες διασταύρωσης των γονέων για τη δημιουργία εφικτών (feasible) λύσεων-απογόνων
        while not flag:  # Επανάληψη διασταύρωσης των γονέων μέχρι να δημιουργηθούν εφικτές λύσεις-απόγονοι
            offsprings = customer_position_alignment_crossover(parent1, parent2)  # Δημιουργία δύο απογόνων
            both offspring objectives = []
            result1 = evaluate solution(offsprings[0], distances, travel times, service times, time windows, demands, vehicle capacity, max route time)  # Αξιολόγηση απογόνου 1

            result2 = evaluate_solution(offsprings[1], distances, travel_times, service_times, time_windows, demands, vehicle_capacity, max_route_time)  # Αξιολόγηση απογόνου 2

            if result1['feasible'] and result2['feasible']:  # Αν και δύο απόγονοι είναι εφικτές λύσεις
                objective11, objective12, objective13 = result1['Total Distance Travelled'], result1['Deviation from Mean Load'], result1['Total Waiting Time']
                both_offspring_objectives.append((objective11, objective12, objective13))
                objective21, objective22, objective23 = result2['Total Distance Travelled'], result2['Deviation from Mean Load'], result2['Total Waiting Time']
                both_offspring_objectives.append((objective21, objective22, objective23))
                flag = True
            elif tries == 10:
                flag = True  # Δε βρέθηκαν εφικτές λύσεις, διακοπή της επανάληψης
        if tries != 10:
            for idx, objectives in enumerate(both offspring objectives):
                # Αν ο απόγονος κυριαρχείται από κάποιον γονέα, κρατάμε τον γονέα
```

```
                if dominates(solutions objectives values[parent1 index], objectives) and
parent1 not in next archive:
                    offsprings population.append(parent1)
                elif dominates(solutions objectives values[parent2 index], objectives) and
parent2 not in next_archive:
                    offsprings_population.append(parent2)
                else:  # Αν ο απόγονος δεν κυριαρχείται κανέναν από τους δύο γονείς, κρατάμε
τον απόγονο
                    offsprings population.append(offsprings[idx])
        else:  # Αν γίνουν tries σε αριθμό προσπάθειες και δε βρεθούν δύο εφικτές λύσεις-
απόγονοι, διατήρηση των γονέων
            offsprings_population.extend([parent1, parent2])

        if random.random() < mutation rate:  # Μετάλλαξη των δύο τελευταίων λύσεων-απογόνων με
πιθανότητα mutation rate
            for last_solutions_index in ([-1, -2]):
                flag = False
                while not flag:  # Επανάληψη μέχρι να προκύψει εφικτή μεταλλαγμένη λύση
                    mutated solution = mutate(offsprings population[last solutions index])  #
Μετάλλαξη της τελευταίας λύσης που προστέθηκε στον πληθυσμό
                    mutated solution eval = evaluate solution(mutated solution, distances,
travel times, service times, time windows, demands, vehicle capacity, max route time)
                    if mutated solution eval['feasible']:
                        flag = True
                        offsprings population[last solutions index] = mutated solution

    max_iterations = 10 # Αριθμός επαναλήψεων περιορισμένης αναζήτησης για κάθε λύση
    tabu list size= 5 # Μέγεθος λίστας tabu
    for solution in next archive second half:
        solution fitness value = fitness values[solution[1]]
        solution = tabu search(solution[0], solution fitness value, max iterations,
tabu list size, distances, travel times, service times, time windows, demands,
vehicle_capacity, max_route_time)
        offsprings population.append(solution)
    population = offsprings population
    archive = [solution for solution, _ in next archive]
```

4.1.3.2. Συνάρτηση δημιουργίας γειτονιάς λύσης generate_neighbourhood

```
def generate_neighbourhood(solution, tabu_list):
    neighbors = [] # Αρχικοποίηση λίστας γειτονικών λύσεων
    # Δημιουργία όλων των πιθανών γειτονικών λύσεων με 1-1 exchange πελατών μεταξύ διαδρομών
    for route idx, route in enumerate(solution):
        for i in range(1, len(route) - 2):
            for j in range(i + 1, len(route) - 1):
                neighbor = copy.deepcopy(solution)
                neighbor[route_idx][i], neighbor[route_idx][j] = neighbor[route_idx][j],
neighbor[route idx][i]
                 if neighbor not in tabu list:
                    neighbors.append(neighbor)
    return neighbors
```

4.1.3.3. Συνάρτηση περιορισμένης αναζήτησης tabu_search

```
def tabu_search(initial_solution, initial_solution_fitness_value, max_iterations,
tabu list size, distances, travel times, service times, time windows, demands,
vehicle capacity, max route time):
    current solution = initial solution  # Ορισμός της αρχικής λύσης ως τρέχουσα
    best_solution = initial_solution  # Η καλύτερη έως τώρα λύση
    best solution fitness = initial solution fitness value  # Fitness αρχικής λύσης

    tabu list = []

    for _ in range(max_iterations):
        neighbors = generate_neighbourhood(current_solution, tabu_list)  # Δημιουργία
γειτονιάς τρέχουσας λύσης
        neighbours objective values = []
        for neighbor in neighbors:
            evaluation = evaluate solution(neighbor, distances, travel times, service times,
time_windows, demands, vehicle_capacity, max_route_time)
            if evaluation['feasible']:
                obj1, obj2, obj3 = evaluation['Total Distance Travelled'],
evaluation['Deviation from Mean Load'], evaluation['Total Waiting Time']
                neighbours objective values.append((obj1, obj2, obj3))

        if len(neighbours_objective_values) > 2:
            # Υπολογισμός S(i), R(i), D(i) και F(i) = R(i) + D(i) των γειτονικών λύσεων
```

```
            strengths = calculate_solutions_strengths(neighbours_objective_values)
            raw_fitness_values = calculate_solutions_raw_fitness(strengths,
neighbours_objective_values)
            densities = calculate_solutions_density(neighbours_objective_values)
            neighbourhood_fitness_values =
calculate_solutions_total_fitness(raw_fitness_values, densities)
            # Επιλογή της καλύτερης γειτονικής λύσης βάσει fitness
            best_neighbor_index =
neighbourhood_fitness_values.index(min(neighbourhood_fitness_values))
            best_neighbor = neighbors[best_neighbor_index]

            current_solution = best_neighbor  # Ορισμός της καλύτερης γειτονικής λύσης ως
τρέχουσα
            tabu_list.append(best_neighbor)  # Προσθήκη της καλύτερης λύσης στη λίστα tabu
            if len(tabu_list) > tabu_list_size:  # Αφαίρεση της πρώτης λύσης της λίστας tabu
αν δε χωράει η επόμενη
                tabu_list.pop(0)
            # Έλεγχος αν η καλύτερη γειτονική λύση είναι καλύτερη από την best_solution
            if min(neighbourhood_fitness_values) < best_solution_fitness:
                best_solution = best_neighbor
                best_solution_fitness = min(neighbourhood_fitness_values)
    return best_solution
```

4.1.4. Αποτελέσματα εφαρμογής 1ου προτεινόμενου υβριδικού αλγορίθμου

Τα αποτελέσματα παρουσιάζονται όπως ορίστηκε στον κώδικα παραπάνω για μέγιστο αριθμό γενεών = 100 (πρώτο κριτήριο τερματισμού), πιθανότητα μετάλλαξης = 40%, μέγεθος πληθυσμού = μέγεθος αρχείου = 300, μέγεθος δεξαμενής ζευγαρώματος = 150, μέγεθος λίστας tabu = 5, επαναλήψεις περιορισμένης αναζήτησης = 10. Κι εδώ σε περίπτωση που ληφθεί αρχείο next_archive με πλήθος μη κυριαρχούμενων λύσεων ίσο με το μέγεθος του αρχείου, η διαδικασία τερματίζεται (δεύτερο κριτήριο τερματισμού). Τα αποτελέσματα φαίνονται παρακάτω (ανά 5 γενιές) για ένα τρέξιμο του προγράμματος:

Generation 5
next_archive unique NON-DOMINATED SOLUTIONS: 26
Generation 10
next_archive unique NON-DOMINATED SOLUTIONS: 53
Generation 15
next_archive unique NON-DOMINATED SOLUTIONS: 58
Generation 20
next_archive unique NON-DOMINATED SOLUTIONS: 62
Generation 25
next_archive unique NON-DOMINATED SOLUTIONS: 93
Generation 30
next_archive unique NON-DOMINATED SOLUTIONS: 73
Generation 35
next_archive unique NON-DOMINATED SOLUTIONS: 124
Generation 40
next_archive unique NON-DOMINATED SOLUTIONS: 130
Generation 45
next_archive unique NON-DOMINATED SOLUTIONS: 121
Generation 50
next_archive unique NON-DOMINATED SOLUTIONS: 137
Generation 55
next_archive unique NON-DOMINATED SOLUTIONS: 205

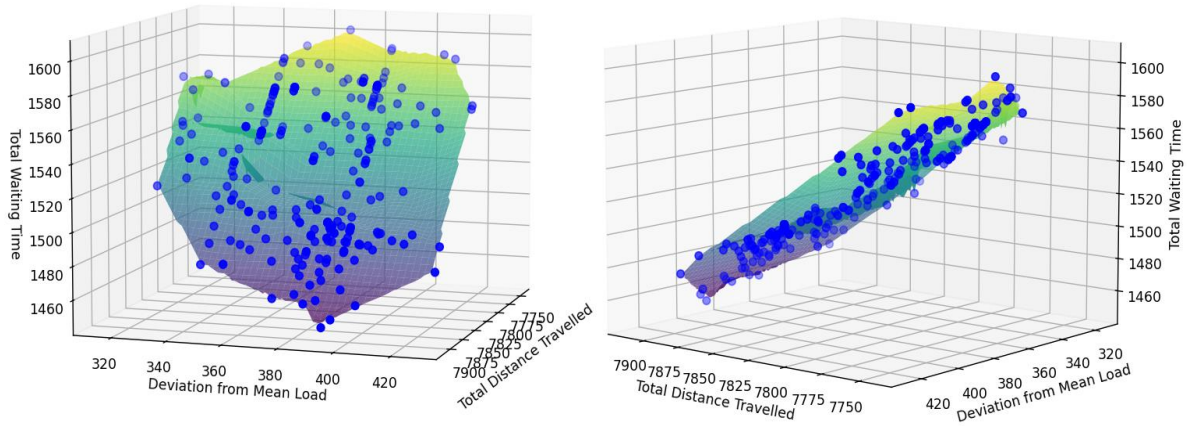
Generation 60
next_archive unique NON-DOMINATED SOLUTIONS: 271
Generation 65
next_archive unique NON-DOMINATED SOLUTIONS: 159
Generation 66
next_archive unique NON-DOMINATED SOLUTIONS: 198
Generation 67
next_archive unique NON-DOMINATED SOLUTIONS: 244
Generation 68
next_archive unique NON-DOMINATED SOLUTIONS: 291
Generation 69
next_archive unique NON-DOMINATED SOLUTIONS: 300

Το πρόγραμμα τερματίστηκε με ικανοποίηση του δεύτερου κριτηρίου τερματισμού μετά από 69 γενιές, με τις εξής μετρήσεις απόδοσης:

Spacing metric: 0.40842558843446584

Hypervolume indicator: 7.116241391699080

Τέλος, η απεικόνιση του μετώπου Pareto (από δύο διαφορετικές γωνίες):



Εικόνα 4.2. *Το μέτωπο Pareto από την εφαρμογή συνδυασμού SPEA-II και Tabu search στο πρόβλημα R1_2_1*

Για να αξιολογηθεί η αποτελεσματικότητα αλλά και η ευστάθεια του παραπάνω αλγορίθμου απαιτείται κι εδώ **στατιστική ανάλυση** των αποτελεσμάτων μετά από 20 τρεξίματα του προγράμματος με τις παραμέτρους που ορίστηκαν παραπάνω, η οποία παρουσιάζεται παρακάτω:

Run #	Generations	Spacing metric	Hypervolume indicator
1	69	0,40842558843446584	7,116241391699080
2	48	0,45907241105663044	6,989669253696330
3	92	0,5011843979691207	7,087286793314520
4	69	0,20963078945444694	7,072928614303910
5	56	0,4502425896933588	6,785142289545980
6	61	0,5132179419164766	6,821684213292020

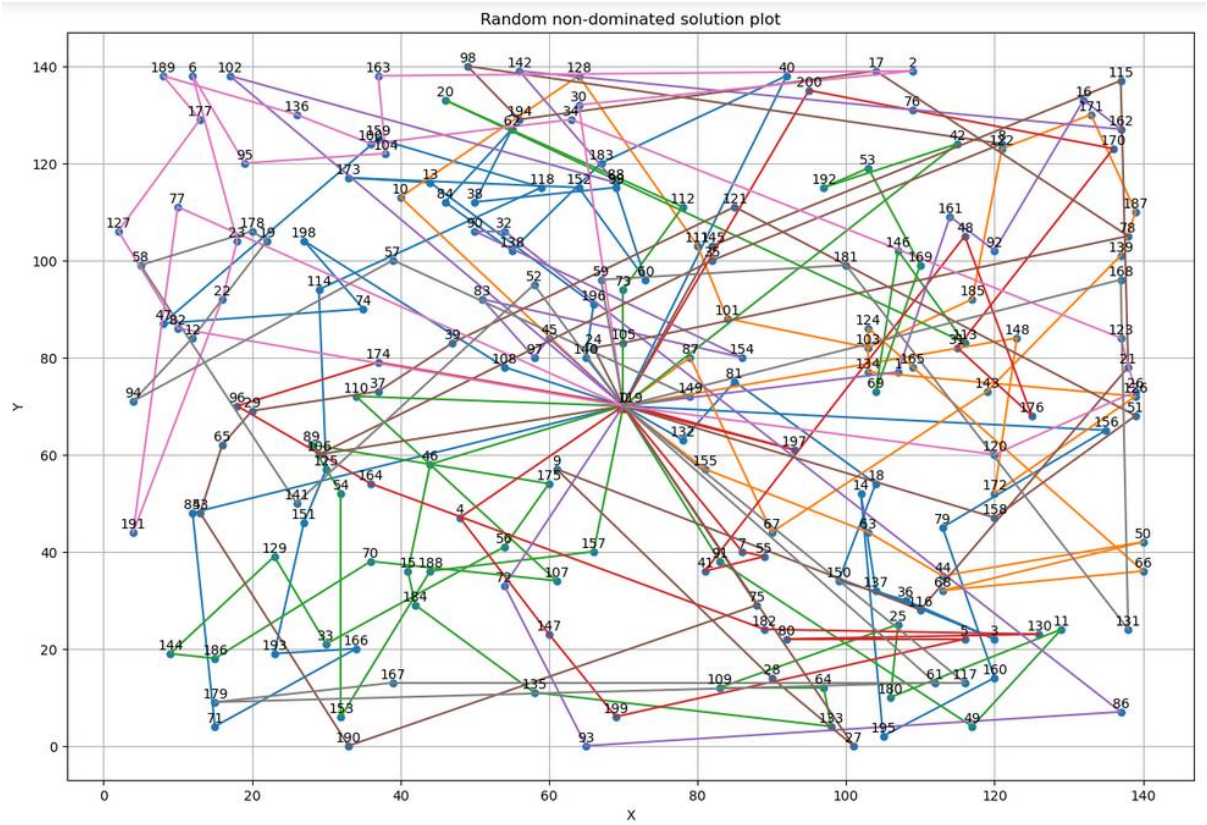
7	53	0,4277377308577704	7,141407387428860
8	49	0,21476790917199778	7,023786264184520
9	72	0,20501725861792763	7,207187358953380
10	74	0,1447602379734173	7,437159168978930
11	79	0,4756331157770095	7,054842892925310
12	52	0,4317689982709556	7,148124425261310
13	74	0,1336576988275908	8,061166600395180
14	62	0,40919357786186067	7,051866136816870
15	51	0,43610792601358117	7,010172300144340
16	74	0,36931670224835983	6,618050142158100
17	65	0,10934835700292944	8,093559236413580
18	68	0,4571785710007973	7,148530203664620
19	80	0,4856594450474678	8,225914596744250
20	53	0,43681564861575384	7,142804191518370
Μέση τιμή	**65,05**	**0,36**	**7,21**
Τυπική απόκλιση	**11,81**	**0,13**	**0,42**

4.1.5. Σχολιασμός αποτελεσμάτων

Όπως γίνεται αντιληπτό από την σύγκριση των αποτελεσμάτων των δύο αλγορίθμων, η εφαρμογή του 1ου προτεινόμενου υβριδικού αλγορίθμου (συνδυασμός SPEA-II και Tabu search) σε σύγκριση με τον SPEA-II οδήγησε σε καλύτερα αποτελέσματα, καθώς:

1. Η σύγκλιση στον ίδιο αριθμό (300) μη κυριαρχούμενων λύσεων γίνεται σε λιγότερες γενιές (μέση τιμή 65,05 έναντι 81,75), δηλαδή γρηγορότερα.
2. Η κατανομή (spacing metric) του μετώπου Pareto έχει μικρότερη τιμή (μέση τιμή 0,36 έναντι 0,43), υποδεικνύοντας ένα πιο ομοιόμορφα κατανεμημένο και με μεγαλύτερη ποικιλομορφία σύνολο λύσεων.
3. Ο υπερόγκος του μετώπου Pareto έχει μεγαλύτερη τιμή (μέση τιμή 7,21 έναντι 6,99), υποδεικνύοντας ένα σύνολο λύσεων με καλύτερη κατανομή στον χώρο αναζήτησης και με καλύτερης ποιότητας λύσεις.

Ενδεικτικά, η απεικόνιση μίας τυχαία επιλεγμένης λύσης από τις μη κυριαρχούμενες λύσεις του μετώπου Pareto που προέκυψε από τη χρήση του 1ου υβριδικού αλγορίθμου φαίνεται παρακάτω, όπου διακρίνονται οι τοποθεσίες των 200 πελατών και οι διαδρομές των (20 σε αριθμό) οχημάτων:



Εικόνα 4.3. *Απεικόνιση μίας μη κυριαρχούμενης λύσης του βέλτιστου μετώπου Pareto*

Αναλυτικά, τα στοιχεία για κάθε όχημα της παραπάνω λύσης φαίνονται παρακάτω:

Vehicle 1:
Route: [0, 85, 71, 166, 193, 151, 125, 114, 118, 159, 47, 74, 198, 108, 0]
Demand fulfilled: 199
Waiting time: 31.40881156139669
Travel time: 588.9829072995159
Distance travelled: 424.9791797232151

Vehicle 2:
Route: [0, 132, 81, 18, 150, 36, 3, 137, 14, 195, 160, 79, 156, 0]
Demand fulfilled: 196
Waiting time: 23.498275785927117
Travel time: 421.02824823809715
Distance travelled: 332.70044942586867

Vehicle 3:
Route: [0, 140, 196, 13, 173, 152, 60, 99, 38, 62, 84, 138, 183, 40, 0]
Demand fulfilled: 193
Waiting time: 22.844987580707624
Travel time: 489.90058113322374
Distance travelled: 327.98012343638925

Vehicle 4:
Route: [0, 87, 67, 143, 139, 187, 171, 122, 185, 103, 101, 111, 128, 10, 0]
Demand fulfilled: 199

Waiting time: 65.36901040002721
Travel time: 547.8010863942093
Distance travelled: 378.86891314356126

Vehicle 5:
Route: [0, 155, 63, 44, 50, 68, 66, 165, 124, 134, 126, 172, 148, 0]
Demand fulfilled: 192
Waiting time: 25.024012806053776
Travel time: 486.55022783268294
Distance travelled: 362.10809701724446

Vehicle 6:
Route: [0, 73, 112, 20, 113, 146, 69, 169, 53, 192, 42, 0]
Demand fulfilled: 195
Waiting time: 55.080780513973096
Travel time: 503.3312178677751
Distance travelled: 363.3239877524883

Vehicle 7:
Route: [0, 46, 15, 184, 135, 133, 64, 109, 25, 180, 11, 49, 91, 0]
Demand fulfilled: 198
Waiting time: 100.36029541039946
Travel time: 529.6083582564364
Distance travelled: 319.86435048433503

Vehicle 8:
Route: [0, 157, 188, 153, 54, 89, 175, 56, 33, 129, 144, 186, 70, 107, 110, 0]
Demand fulfilled: 192
Waiting time: 53.962779907073866
Travel time: 607.8342000742298
Distance travelled: 407.0491752225658

Vehicle 9:
Route: [0, 4, 147, 199, 5, 80, 130, 182, 164, 96, 174, 119, 197, 0]
Demand fulfilled: 186
Waiting time: 38.92929151984109
Travel time: 574.0545390353205
Distance travelled: 412.220303438765

Vehicle 10:
Route: [0, 7, 55, 41, 48, 176, 31, 170, 200, 0]
Demand fulfilled: 193
Waiting time: 33.44756910071274
Travel time: 489.003502722905
Distance travelled: 336.78617334206183

Vehicle 11:
Route: [0, 1, 161, 92, 16, 162, 76, 142, 88, 102, 0]
Demand fulfilled: 188
Waiting time: 71.1916490601665

Travel time: 542.8799969619054
Distance travelled: 371.6659126571048

Vehicle 12:
Route: [0, 72, 93, 86, 32, 90, 154, 83, 97, 45, 0]
Demand fulfilled: 185
Waiting time: 47.72974659856811
Travel time: 515.899293564647
Distance travelled: 397.6020404476052

Vehicle 13:
Route: [0, 106, 35, 8, 98, 194, 17, 78, 105, 24, 0]
Demand fulfilled: 180
Waiting time: 31.26787935131844
Travel time: 540.8881210547813
Distance travelled: 426.5463235486481

Vehicle 14:
Route: [0, 145, 115, 21, 116, 9, 28, 27, 0]
Demand fulfilled: 180
Waiting time: 105.33027334389297
Travel time: 590.5783279383966
Distance travelled: 419.2548567667375

Vehicle 15:
Route: [0, 158, 51, 121, 39, 37, 29, 65, 43, 190, 75, 0]
Demand fulfilled: 198
Waiting time: 109.71598429066322
Travel time: 632.3327036675859
Distance travelled: 412.52748665998797

Vehicle 16:
Route: [0, 120, 26, 123, 34, 100, 136, 189, 177, 127, 82, 0]
Demand fulfilled: 195
Waiting time: 50.354938389467975
Travel time: 501.7688393811919
Distance travelled: 350.0473431298717

Vehicle 17:
Route: [0, 30, 2, 163, 104, 95, 6, 23, 22, 191, 77, 0]
Demand fulfilled: 137
Waiting time: 40.45673923370709
Travel time: 587.6815644514102
Distance travelled: 470.39393590110114

Vehicle 18:
Route: [0, 149, 57, 94, 12, 19, 178, 58, 141, 52, 0]
Demand fulfilled: 189
Waiting time: 150.9868543477839

Travel time: 528.4742986082191
Distance travelled: 296.764441167032

Vehicle 19:
Route: [0, 168, 131, 181, 59, 0]
Demand fulfilled: 64
Waiting time: 215.91915887652158
Travel time: 536.5445589344146
Distance travelled: 287.26081159650874

Vehicle 20:
Route: [0, 117, 167, 179, 61, 0]
Demand fulfilled: 54
Waiting time: 219.74385275784445
Travel time: 611.0950277189371
Distance travelled: 342.4621919520865

4.2. Εφαρμογή 2η: Επίλυση προβλήματος C1_2_1

4.2.1. Κώδικας εφαρμογής αλγορίθμου NSGA-II

4.2.1.1. Εισαγωγή δεδομένων

```
import pandas as pd
from scipy.spatial import distance matrix
from functions import *
import random
import copy

file path = 'R1 2 1.txt'
customers_df = pd.read_csv(file_path, delim_whitespace=True, header=None, names=['CUST NO.',
'XCOORD.', 'YCOORD.', 'DEMAND', 'READY TIME', 'DUE DATE', 'SERVICE TIME'], skiprows=8)
x coordinates = customers df['XCOORD.'].tolist()
y coordinates = customers df['YCOORD.'].tolist()
coordinates = [(a, b) for a, b in zip(x coordinates, y coordinates)]  # Λίστα με ζεύγη
συντεταγμένων (X, Y) πελατών
travel_times = distance_matrix(coordinates, coordinates) # Πίνακας χρόνων μετάβασης μεταξύ
πελατών

file path = 'C1 2 1.txt'

df1 = pd.read_csv(file_path, delim_whitespace=True, header=None, skiprows=4, nrows=1)
_, vehicle_capacity = df1.iloc[0]
customers df = pd.read csv(file path, delim whitespace=True, header=None, names=['CUST NO.',
'XCOORD.', 'YCOORD.', 'DEMAND',
                                                                                'READY TIME',
'DUE DATE', 'SERVICE TIME'], skiprows=8)
indexes = customers_df['CUST NO.'].tolist()  # Λίστα με τους δείκτες των πελατών
demands = customers_df['DEMAND'].tolist()  # Λίστα τη ζήτηση του κάθε πελάτη
service times = customers df['SERVICE TIME'].tolist()  # Λίστα με το χρόνο εξυπηρέτησης του
κάθε πελάτη

x_coordinates = customers_df['XCOORD.'].tolist()
y_coordinates = customers_df['YCOORD.'].tolist()
coordinates = [(a, b) for a, b in zip(x_coordinates, y_coordinates)]  # Λίστα με ζεύγη
συντεταγμένων (X, Y) πελατών
distances = distance matrix(coordinates, coordinates)  # Πίνακας αποστάσεων μεταξύ πελατών

max_route_time = max(customers_df['DUE DATE'].tolist())  # Μέγιστος χρόνος διαδρομής για κάθε
όχημα
```

Με χρήση του παραπάνω κώδικα γίνεται ανάγνωση των δεδομένων από το αρχείο C1_2_1.txt και η αποθήκευση τους σε μεταβλητές και λίστες. Με σκοπό την διαφοροποίηση των δεδομένων σχετικά με τις αντικειμενικές συναρτήσεις του προβλήματος και την προώθηση της ποικιλομορφίας των λύσεων, ο πίνακας των χρόνων μετάβασης αποκτάται από το αρχείο R1_2_1.txt για τους 200 πελάτες, ώστε να μην ταυτίζεται με τον πίνακα των αποστάσεων. Τα χρονικά παράθυρα που διαβάζονται επίσης δεν λαμβάνονται υπόψη καθώς το πρόβλημα που επιλύεται δεν εμπεριέχει τον περιορισμό αυτό. Ως μέγιστος χρόνος ταξιδιού κάθε οχήματος λαμβάνεται ξανά το β χρονικό παράθυρο της αποθήκης.

4.2.1.2. Δημιουργία αρχικού πληθυσμού

Καθώς δεν υπάρχουν χρονικά παράθυρα που να περιορίζουν τη χρήση ευρετικών αλγορίθμων που βασίζονται στην απόσταση μεταξύ των πελατών, εδώ χρησιμοποιείται ο αλγόριθμος του πλησιέστερου γείτονα για τη δημιουργία των αρχικών λύσεων. Καθώς ο αλγόριθμος αυτός εξάγει πάντα την ίδια αρχική λύση, και λόγω του ότι απαιτείται ο αρχικός πληθυσμός λύσεων να μην είναι ομοιογενής αλλά διαφοροποιημένος, χρησιμοποιείται ο πλησιέστερος γείτονας τροποποιημένος, με την επιλογή κάθε φορά του k πλησιέστερου γείτονα με τυχαιότητα. Έτσι για μία σταθερή τιμή k μπορεί να προκύψει πολύ μεγάλος αριθμός διαφορετικών αρχικών λύσεων. Για περαιτέρω προώθηση της ποικιλομορφίας, το k μεταβάλλεται και επιλέγεται κι αυτό με τυχαιότητα (τιμές 2, 3, 4, 5), εντός κάποιων ορίων ώστε να μην καθίσταται η δημιουργία λύσεων μία τυχαία διαδικασία.

```
initial_population = []
population_size = 500
number_of_customers = len(customers_df)
for _ in range(population_size):
    k = random.choice([2, 3, 4, 5])
    new_solution = nearest_neighbor_random(number_of_customers, distances, travel_times,
service_times, demands, max_route_time, vehicle_capacity, k)
    initial_population.append(new_solution)
```

4.2.1.3. Συνάρτηση δημιουργίας (εφικτής) αρχικής λύσης **nearest_neighbor_random**

```
def nearest_neighbor_random(number_of_customers, distances, travel_times, service_times,
                            demands, max_route_time, vehicle_capacity, k):
    unvisited_customers = set(range(number_of_customers)) - {0}  # Δείκτες των πελατών προς
εξυπηρέτηση
    routes = []  # Λίστα όπου θα αποθηκεύονται οι διαδρομές των οχημάτων
    current_route = [0]  # Αρχικοποίηση της διαδρομής του οχήματος, που ξεκινάει από την
αποθήκη
    current_route_travel_time = 0  # Αρχικοποίηση χρόνου ταξιδιού του πρώτου οχήματος
    current_route_demand = 0  # Αρχικοποίηση φορτίου πρώτου οχήματος
    current_route_distance = 0  # Αρχικοποίηση διανυόμενης απόστασης πρώτου οχήματος

    while unvisited_customers:  # Επανάληψη έως ότου δεν υπάρχει μη εξυπηρετημένος πελάτης
        next_customer = None  # Αρχικά None σε κάθε επανάληψη
        last_customer_in_route = current_route[-1]  # Ο τελευταίος πελάτης στη διαδρομή
        min_distance = float('inf')  # Αρχικοποίηση της απόστασης του κοντινότερου πελάτη
(αρχικά άπειρο)

        # Ταξινόμηση των πελατών βάσει απόστασης από τον τελευταίο πελάτη στη διαδρομή
        neighbors_sorted = sorted(unvisited_customers, key=lambda cust:
distances[last_customer_in_route][cust])
```

```
        k_nearest_neighbors = neighbors_sorted[:k]  # Επιλογή των k κοντινότερων πελατών
        random.shuffle(k_nearest_neighbors)  # Ανακάτεμα των k κοντινότερων

        for customer in k_nearest_neighbors:
            travel_to_customer_distance = distances[last_customer_in_route][customer]
            travel_to_customer_time = travel_times[last_customer_in_route][customer]
            customer_service_time = service_times[customer]
            travel_and_service_time = travel_to_customer_time + customer_service_time

            if travel_to_customer_distance < min_distance and current_route_travel_time +
travel_and_service_time + \
                    travel_times[customer][0] <= max_route_time and current_route_demand +
demands[customer] <= vehicle_capacity:
                next_customer = customer
                min_distance = travel_to_customer_distance
                break  # Διακοπή στον πρώτο εφικτό πελάτη από τους k κοντινότερους

        if next_customer is not None:  # Βρέθηκε εφικτός πελάτης
            current_route.append(next_customer)  # Προσθήκη του πελάτη στη διαδρομή
            unvisited_customers.remove(next_customer)  # Αφαίρεση από τους μη εξυπηρετημένους
            current_route_travel_time += travel_times[last_customer_in_route][next_customer] +
service_times[next_customer]
            current_route_demand += demands[next_customer]
            current_route_distance += distances[last_customer_in_route][next_customer]

        else:
            current_route.append(0)  # Επιστροφή στην αποθήκη
            routes.append(current_route)  # Νέα διαδρομή στη λύση
            current_route = [0]  # Αρχικοποίηση διαδρομής επόμενου οχήματος
            current_route_travel_time = 0  # Αρχικοποίηση χρόνου ταξιδιού επόμενου οχήματος
            current_route_demand = 0  # Αρχικοποίηση φορτίου επόμενου οχήματος
            current_route_distance = 0  # Αρχικοποίηση διανυόμενης απόστασης επόμενου οχήματος

    if current_route != [0]:  # Για προσθήκη της τελευταίας διαδρομής στη λύση όταν αδειάσει
το set unvisited_customers
        current_route.append(0)
        routes.append(current_route)

    return routes
```

4.2.1.4. Κώδικας αλγορίθμου NSGA-II

```
max_generations = 100
mutation_rate = 0.4

population = copy.deepcopy(initial_population)  # Αρχικοποίηση πρώτου πληθυσμού
pareto_front = []
for generation in range(max_generations):
    print(f"Generation {generation + 1}")
    solutions_objectives_values = []
    for solution in population:  # Υπολογισμός των τιμών στις 3 αντικειμενικές συναρτήσεις για
κάθε λύση
        result = evaluate_solution(solution, distances, travel_times, service_times,
                                   demands, vehicle_capacity, max_route_time)

        objective1, objective2, objective3 = result['Number of Vehicles'], result['Total
Distance Travelled'], result['Total Travel Time']

        solutions_objectives_values.append((objective1, objective2, objective3))  # Δημιουργία
tuple με τις τιμές των objectives

    # Μη κυριαρχούμενη ταξινόμηση
    fronts = non_dominated_sorting(solutions_objectives_values)

    # Δημιουργία λίστας με το μέτωπο όπου ανήκει η κάθε λύση
    solutions_front_indexes = [-1] * len(population)
    for front_index, front in enumerate(fronts):
        for solution_index in front:
            solutions_front_indexes[solution_index] = front_index

    # Υπολογισμός αποστάσεων συνωστισμού για κάθε λύση του πληθυσμού
    crowding_distances = [0] * len(solutions_objectives_values)
    for front in fronts:
        calculate_front_crowding_distances(front, solutions_objectives_values,
crowding_distances)
    solution_tuples = []
```

```
    for rank, front in enumerate(fronts):
        for solution index in front:
            solution = population[solution index]
            solution tuples.append((solution index, solution, rank))

    mating_pool_size = population_size
    mating pool = binary tournament selection(solution tuples, crowding distances,
mating pool size)
    offsprings population = []  # Αρχικοποίηση της λίστας με λύσεις-απογόνους
    for i in range(0, mating pool size, 2):  # Επαναληπτικό πέρασμα των λύσεων στη δεξαμενή
ζευγαρώματος ανά 2 λύσεις
        parent1, parent1_index = mating_pool[i]  # Γονέας 1
        parent2, parent2 index = mating pool[i + 1]  # Γονέας 2
        # Διασταύρωση γονέων
        flag = False
        both_offspring_objectives = []
        offsprings = []
        max_tries = 10
        tries = 0  # Προσπάθειες διασταύρωσης των γονέων για τη δημιουργία εφικτών (feasible)
λύσεων-απογόνων
        while not flag:  # Επανάληψη διασταύρωσης των γονέων μέχρι να δημιουργηθούν εφικτές
λύσεις-απόγονοι
            offsprings = customer position alignment crossover(parent1, parent2)  # Δημιουργία
δύο απογόνων
            both offspring objectives = []
            result1 = evaluate solution(offsprings[0], distances, travel times, service times,
demands, vehicle_capacity, max_route_time) # Αξιολόγηση απογόνου 1
            result2 = evaluate solution(offsprings[1], distances, travel times, service times,
demands, vehicle capacity, max route time) # Αξιολόγηση απογόνου 2
            if result1['feasible'] and result2['feasible']:  # Αν και δύο απόγονοι είναι
εφικτές λύσεις
                objective11, objective12, objective13 = result1['Number of Vehicles'],
result1['Total Distance Travelled'], result1['Total Travel Time']
                both offspring objectives.append((objective11, objective12, objective13))
                objective21, objective22, objective23 = result2['Number of Vehicles'],
result2['Total Distance Travelled'], result2['Total Travel Time']
                both offspring objectives.append((objective21, objective22, objective23))
                flag = True
            elif tries == max_tries:
                flag = True  # Δε βρέθηκαν εφικτές λύσεις, διακοπή της επανάληψης

        if tries < max tries:
            for idx, objectives in enumerate(both offspring objectives):
                # Αν ο απόγονος κυριαρχείται από κάποιον γονέα, κρατάμε τον γονέα
                if dominates(solutions_objectives_values[parent1_index], objectives):
                    offsprings population.append(parent1)
                elif dominates(solutions objectives values[parent2 index], objectives):
                    offsprings population.append(parent2)
                else:  # Αν ο απόγονος δεν κυριαρχείται κανέναν από τους δύο γονείς, κρατάμε
τον απόγονο
                    offsprings population.append(offsprings[idx])
        else:  # Αν γίνουν tries σε αριθμό προσπάθειες και δε βρεθούν δύο εφικτές λύσεις-
απόγονοι, διατήρηση των γονέων
            offsprings population.extend([parent1, parent2])

        if random.random() < mutation_rate:  # Μετάλλαξη των δύο τελευταίων λύσεων-απογόνων με
πιθανότητα mutation rate
            for last solutions index in ([-1, -2]):
                flag = False
                while not flag:  # Επανάληψη μέχρι να προκύψει εφικτή μεταλλαγμένη λύση
                    mutated_solution = mutate(offsprings_population[last_solutions_index])  #
Μετάλλαξη της τελευταίας λύσης που προστέθηκε στον πληθυσμό
                    mutated solution eval = evaluate solution(mutated solution, distances,
travel times, service times, demands, vehicle capacity, max route time)
                    if mutated solution eval['feasible']:
                        flag = True
                        offsprings_population[last_solutions_index] = mutated_solution

    # Δημιουργία του συνδυασμένου πληθυσμού γονέων-απογόνων
    parents population = [solution for solution,   in mating pool]
    combined_population = parents_population + offsprings_population

    # Αξιολόγηση του συνδυασμένου πληθυσμού
    solutions objectives values = []
    for solution in combined population:
        result = evaluate solution(solution, distances, travel times, service times,
                                   demands, vehicle_capacity, max_route_time)
```

```
        objective1, objective2, objective3 = result['Number of Vehicles'], result['Total
Distance Travelled'], result['Total Travel Time']

        solutions objectives values.append((objective1, objective2, objective3))  # Δημιουργία
λίστας από tuple με τις τιμές των objectives για κάθε λύση

     # Μη κυριαρχούμενη ταξινόμηση του συνδυασμένου πληθυσμού
    fronts = non dominated sorting(solutions objectives values)
    pareto front = []
    for i in fronts[0]:
        solution = combined_population[i]
        if solution not in pareto_front:
            pareto front.append(solution)
    print(f" First front solutions number: {len(pareto front)}")

    # Υπολογισμός αποστάσεων συνωστισμού για κάθε λύση του συνδυασμένου πληθυσμού
    crowding_distances = [0] * len(combined_population)
    for front in fronts:
        calculate front crowding distances(front, solutions objectives values,
crowding distances)

    # Δημιουργία του πληθυσμού της επόμενης γενιάς
    next population = []
    front index = 0
    while len(next population) < population size: λ# Προσθήκη λύσεων των μετώπων που «χωρούν»
        solutions in front = [combined population[i] for i in fronts[front index]]
        # Αν η προσθήκη των λύσεων του μετώπου front_index δεν ξεπερνάει το population_size
        if len(next population) + len(fronts[front index]) <= population size:
            next population.extend(solutions in front)
            front index += 1
        else: # Συμπλήρωση με λύσεις των τελευταίου μετώπου ταξινομημένες βάσει της απόστασης
συνωστισμού
            last_front = fronts[front_index]
            last front.sort(key=lambda x: crowding distances[x], reverse=True)
            next population.extend(combined population[i] for i in last front[:population size
- len(next population)])
            break
    population = next_population
```

Ως κριτήριο τερματισμού έχει οριστεί ο μέγιστος αριθμός των γενεών. Σε κάθε επανάληψη και μετά την μη κυριαρχούμενη ταξινόμηση του συνδυασμένου πληθυσμού γονέων-απογόνων, το πρώτο επίπεδο κυριαρχίας αντικατοπτρίζει και το μέτωπο Pareto, το οποίο και αποθηκεύεται στη λίστα pareto_front και τυπώνεται σε κάθε επανάληψη. Αφού τερματιστεί ο αλγόριθμος, λαμβάνεται το τελευταίο pareto_front ως το βέλτιστο μέτωπο Pareto.

Οι συναρτήσεις dominates, customer_position_alignment_crossover, mutate, normalize_list, normalize_objectives, calculate_spacing_metric, HV (pymoo) είναι ακριβώς ίδιες με αυτές που χρησιμοποιήθηκαν για τον αλγόριθμο SPEA-II στην 1η εφαρμογή. Ακολουθούν οι διαφοροποιημένες συναρτήσεις που χρησιμοποιούνται εντός του κώδικα του NSGA-II.

4.2.1.5. Συναρτήσεις αξιολόγησης λύσεων evaluate_route, evaluate_solution

```
def evaluate route(route, distances, travel times, service times, demands, vehicle capacity,
max route time):
    # Αρχικοποίηση όλων των παραμέτρων της διαδρομής που θα αξιολογηθεί
    total_distance = 0
    total_travel_time = 0
    total demand = 0
    current time = 0

    for i in range(len(route) - 1):  # Επαναληπτικό πέρασμα όλων των πελατών της διαδρομής
        from_customer = route[i]
        to_customer = route[i + 1]

        total distance += distances[from customer][to customer]
        total_demand += demands[to_customer]
```

```
        travel time = travel times[from customer][to customer]
        total travel time += travel time + service times[to customer]
        current time = total travel time

    if current_time > max_route_time or total_demand > vehicle_capacity:  # Παραβίαση
χωρητικότητας ή μέγιστου χρόνου ταξιδιού
        return {'feasible': False}  # Ανέφικτη διαδρομή
    # Εφικτή διαδρομή, επιστροφή όλων των υπολογισμένων παραμέτρων της διαδρομής
    return {
        'feasible': True,
        'Distance Travelled': total_distance,
        'Travel Time': total_travel_time,
        'Demand fulfilled': total demand
    }

def evaluate_solution(solution, distances, travel_times, service_times, demands,
vehicle_capacity, max_route_time):
    total_distance = 0
    total travel time = 0
    total demand fulfilled = 0
    feasible = True

    for route in solution:  # Αξιολόγηση κάθε διαδρομής στην λίστα solution
        route evaluation = evaluate route(route, distances, travel times, service times,
                                          demands, vehicle capacity, max route time)
        if not route evaluation['feasible']:  # Αν έστω και μία διαδρομή της λύσης είναι
ανέφικτη, η λύση είναι ανέφικτη
            feasible = False
            break
        total distance += route evaluation['Distance Travelled']
        total travel time += route evaluation['Travel Time']
        total demand fulfilled += route evaluation['Demand fulfilled']

    # Επιστροφή όλων των υπολογισμένων παραμέτρων της λύσης
    return {
        'feasible': feasible,
        'Total Distance Travelled': total distance,
        'Total Travel Time': total travel time,
        'Number of Vehicles': len(solution),
        'Total Demand Fulfilled': total demand fulfilled
    }
```

4.2.1.6. Συνάρτηση μη κυριαρχούμενης ταξινόμησης non_dominated_sorting

```
def non dominated sorting(solutions objectives values):
    fronts = []  # Λίστα με λίστες από λύσεις για κάθε επίπεδο κυριαρχίας

    dominated count = [0] * len(solutions objectives values)  # Από πόσες λύσεις κυριαρχείται
η κάθε λύση
    dominated solutions = [[] for   in range(len(solutions objectives values))]  # Σε ποιες
λύσεις κυριαρχεί η κάθε λύση

    for i, sol1 in enumerate(solutions_objectives_values):
        for j, sol2 in enumerate(solutions objectives values):
            if i != j:
                if dominates(sol1, sol2):
                    dominated_count[j] += 1
                    dominated_solutions[i].append(j)

    current front = []
    for i, count in enumerate(dominated count):
        if count == 0:  # Αν δεν κυριαρχούνται από καμία λύση, ανήκουν στο πρώτο μέτωπο
            current_front.append(i)

    while current front:
        next front = []  # Αρχικοποίηση επόμενου μετώπου
        for i in current front:
            for j in dominated_solutions[i]:
                dominated count[j] -= 1
                if dominated count[j] == 0:
                    next front.append(j)
        fronts.append(current front)
        current front = next front

    return fronts
```

4.2.1.7. Συνάρτηση υπολογισμού αποστάσεων συνωστισμού για κάθε μέτωπο calculate_front_crowding_distances

```
def calculate_front_crowding_distances(front, solutions_objectives_values,
crowding_distances):

    num solutions in front = len(front)
    num objectives = len(solutions objectives values[0])
    if num solutions in front == 2: # Αν περιέχει 2 λύσεις, είναι και οι δύο boundary
        return [float('inf'), float('inf')]

    for obj index in range(num objectives):
        sorted front = front[:]
        sorted front.sort(key=lambda x: solutions objectives values[x][obj index])

        # Απόσταση συνωστισμού = άπειρο για τις boundary λύσεις
        crowding distances[sorted front[0]] = float('inf')
        crowding distances[sorted front[-1]] = float('inf')

        min_obj_value = solutions_objectives_values[sorted_front[0]][obj_index]
        max_obj_value = solutions_objectives_values[sorted_front[-1]][obj_index]
        diff = max_obj_value - min_obj_value

        for i in range(1, len(sorted front) - 1):
            if diff > 0:
                solution_index = sorted_front[i]
                crowding_distances[solution_index] +=
(solutions_objectives_values[sorted_front[i + 1]][obj_index] -
solutions objectives values[sorted front[i - 1]][obj index]) / diff

    return crowding distances
```

4.2.1.8. Συνάρτηση δυαδικής επιλογής διαγωνισμού binary_tournament_selection

```
def binary tournament selection(solution tuples, crowding distances, mating pool size):
    mating pool = []  # Αρχικοποίηση δεξαμενής ζευγαρώματος

    for   in range(mating pool size):  # Επαναληπτικά επιλογή δύο τυχαίων λύσεων από τη
δεξαμενή ζευγαρώματος
        index_1, solution_1, solution_1_rank = random.choice(solution_tuples)
        index 2, solution 2, solution 2 rank = random.choice(solution tuples)

        if solution 2 rank < solution 1 rank:
            mating pool.append((solution 2, index 2))
        elif solution_1_rank < solution_2_rank:
            mating_pool.append((solution_1, index_1))
        elif crowding distances[index 1] > crowding distances[index 2]:
            mating pool.append((solution 1, index 1))
        else:
            mating pool.append((solution 2, index 2))

    return mating pool  # Λίστα με tuples (λύση, index της λύσης στον πληθυσμό)
```

4.2.2. Αποτελέσματα εφαρμογής αλγορίθμου NSGA-II

Τα αποτελέσματα παρουσιάζονται όπως ορίστηκε στον κώδικα παραπάνω για μέγιστο αριθμό γενεών = 100 (κριτήριο τερματισμού), πιθανότητα μετάλλαξης 20% (μεγάλη σχετικά πιθανότητα καθώς αποτελεί έναν έμμεσο τρόπο τοπικής αναζήτησης), μέγεθος πληθυσμού = = μέγεθος δεξαμενής ζευγαρώματος = 500 λύσεις. Τα αποτελέσματα φαίνονται παρακάτω (ανά 5 γενιές) για ένα τρέξιμο του προγράμματος:

Generation 5
First front solutions number: 42
Generation 10
First front solutions number: 40

Generation 15
First front solutions number: 59
Generation 20
First front solutions number: 57
Generation 25
First front solutions number: 48
Generation 30
First front solutions number: 63
Generation 35
First front solutions number: 65
Generation 40
First front solutions number: 74
Generation 45
First front solutions number: 51
Generation 50
First front solutions number: 66
Generation 55
First front solutions number: 82
Generation 60
First front solutions number: 85
Generation 65
First front solutions number: 99
Generation 70
First front solutions number: 96
Generation 75
First front solutions number: 82
Generation 80
First front solutions number: 86
Generation 85
First front solutions number: 93
Generation 90
First front solutions number: 92
Generation 95
First front solutions number: 90
Generation 100
First front solutions number: 89

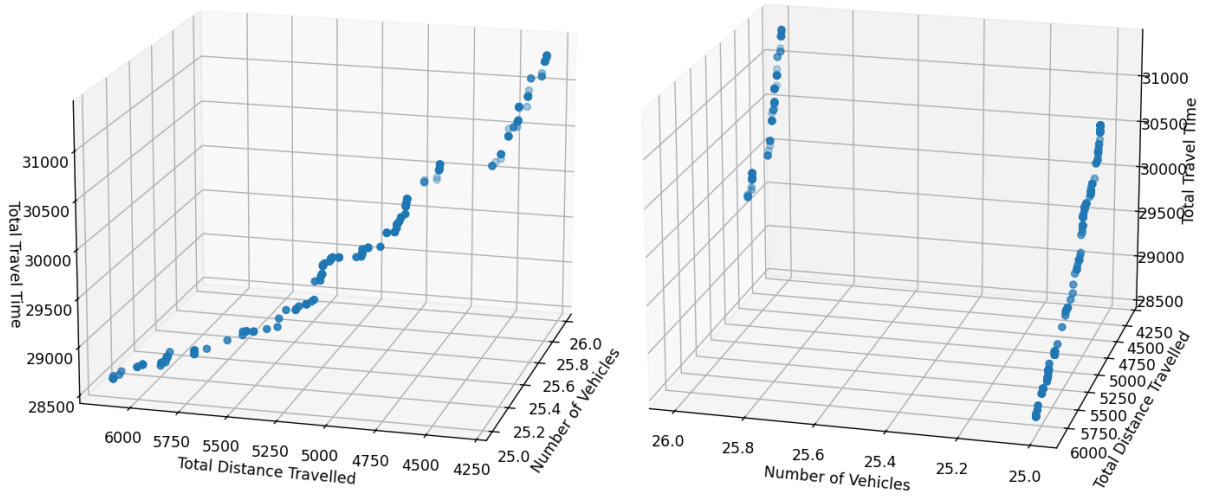
Το πρόγραμμα τερματίστηκε μετά από 100 γενιές, με τις εξής μετρήσεις απόδοσης:

Αριθμός μη κυριαρχούμενων λύσεων: 89

Spacing metric: 0.591573859624484

Hypervolume indicator: 5.62484935707836

Τέλος, η απεικόνιση του μετώπου Pareto (από δύο διαφορετικές γωνίες):



Εικόνα 4.4. *Το μέτωπο Pareto από την εφαρμογή του NSGA-II στο πρόβλημα C_1_2_1*

Σημαντικό είναι να παρατηρηθεί εδώ ότι το μέτωπο αποτελείται από δύο διακριτές περιοχές, λόγω του ότι εμπλέκεται ο στόχος της ελαχιστοποίησης των οχημάτων. Ο λόγος που επικράτησαν και διατηρήθηκαν οι λίγες αυτές λύσεις παρόλο που χρησιμοποιούνται 26 αντί για 25 οχήματα είναι η αρκετά μικρή συνολική διανυόμενη απόσταση. Από την άλλη η περιοχή των λύσεων όπου χρησιμοποιούνται 25 οχήματα είναι πιο ομοιόμορφα και ευρέως κατανεμημένη κάνοντας εμφανείς τις αντισταθμίσεις μεταξύ των στόχων.

Για να αξιολογηθεί η αποτελεσματικότητα αλλά και η ευστάθεια του παραπάνω αλγορίθμου απαιτείται **στατιστική ανάλυση** των αποτελεσμάτων μετά από πολλά τρεξίματα. Παρακάτω παρουσιάζεται η μέση τιμή και η τυπική απόκλιση του αριθμού των (μοναδικών) μη κυριαρχούμενων λύσεων του βέλτιστου μετώπου Pareto πριν ικανοποιηθεί το κριτήριο τερματισμού, της κατανομής (άθροισμα των μετρήσεων για τις δύο διακριτές περιοχές για την αποφυγή παράδοξων αποτελεσμάτων) αλλά και του υπερόγκου του μετώπου, για 20 τρεξίματα του αλγορίθμου με τις παραμέτρους που ορίστηκαν παραπάνω:

Run #	Non-dominated solutions	Spacing metric	Hypervolume indicator
1	89	0,591573859624484	5,62484935707836
2	94	0,555025512774720	3,75046109805197
3	114	0,625047441780836	3,80978299970744
4	90	0,605496327408699	3,77883437353764
5	98	0,558570593819705	3,72525828580378
6	83	0,592083818500558	3,76269127154835
7	77	0,575647253700237	3,71959044974276
8	97	0,483522884429748	3,74678710033555
9	94	0,572030869730192	3,78693705847805
10	113	0,538677761176410	3,75256426543164

11	106	0,545584516426901	3,84090679497587
12	118	0,568895214584117	5,60329830371930
13	95	0,528775809614387	3,70157921487867
14	84	0,590175022683606	3,73290715599157
15	112	0,574196501669496	3,77175179917706
16	77	0,561560418407523	3,65798453126638
17	71	0,617460687427632	3,71207231104389
18	86	0,608482584875425	3,80074270565762
19	90	0,593248391729365	3,78067359222348
20	93	0,579365783601513	3,75821849188364
Μέση τιμή	**93,55**	**0,57**	**3,94**
Τυπική απόκλιση	**12,20**	**0,032**	**0,56**

4.2.3. Κώδικας εφαρμογής 2ου προτεινόμενου υβριδικού αλγορίθμου

Ο κώδικας του αλγορίθμου NSGA-II απαιτεί ελάχιστες τροποποιήσεις για προσθήκη της προσομοιωμένης ανόπτησης ενώ οι υπόλοιπες συναρτήσεις παραμένουν οι ίδιες. Απαιτείται να προστεθεί ο χωρισμός του πληθυσμού σε δύο ίσα μέρη και η εφαρμογή του μεθευρετικού αλγορίθμου της προσομοιωμένης ανόπτησης σε κάθε λύση του δεύτερου μισού του πληθυσμού, ενώ για το πρώτο μισό ακολουθείται ξανά η διαδικασία διασταυρώσεων και μεταλλάξεων του γενετικού αλγορίθμου.

Καθώς πρόκειται για πρόβλημα πολλαπλών αντικειμενικών συναρτήσεων δεν μπορεί να υπολογιστεί κάποιο fitness για κάθε λύση του δεύτερου μισού του πληθυσμού (αφού επιλέγεται να μην γίνει μη κυριαρχούμενη ταξινόμηση ή υπολογισμός αποστάσεων συνωστισμού στο δεύτερο μισό του πληθυσμού). Έτσι, επιλέγεται να θεωρείται καλύτερη μία γειτονική λύση όταν κυριαρχεί έναντι της τρέχουσας από την οποία δημιουργήθηκε. Για τη δημιουργία κάθε γειτονικής λύσης χρησιμοποιείται η συνάρτηση mutate που παρουσιάστηκε και παραπάνω. Το ΔΕ για την πιθανότητα διατήρησης μίας χειρότερης (ή μη κυρίαρχης) λύσης κατά την προσομοιωμένη ανόπτηση τίθεται ως σταθερά = initial temperature/10.

4.2.3.1. Κώδικας NSGA-II + Simulated annealing

```
population size = 500
max generations = 100
mutation_rate = 0.4

population = copy.deepcopy(initial_population)  # Αρχικοποίηση πρώτου πληθυσμού
pareto front = []
for generation in range(max generations):
    print(f"Generation {generation + 1}")
    # Δημιουργία πρώτου και δεύτερου μισού του πληθυσμού
    population_first_half = population[:len(population) // 2]
    population_second_half = population[len(population) // 2:]

    solutions objectives values = []
    for solution in population first half:  # Υπολογισμός των τιμών στις 3 αντικειμενικές
συναρτήσεις για κάθε λύση του μισού πληθυσμού
```

```
        result = evaluate solution(solution, distances, travel times, service times,
                                   demands, vehicle capacity, max route time)

        objective1, objective2, objective3 = result['Number of Vehicles'], result['Total
Distance Travelled'], result['Total Travel Time']

        solutions objectives values.append((objective1, objective2, objective3))  # Δημιουργία
tuple με τις τιμές των objectives

    # Μη κυριαρχούμενη ταξινόμηση
    fronts = non_dominated_sorting(solutions_objectives_values)

    # Δημιουργία λίστας με το front όπου ανήκει η κάθε λύση του μισού
    solutions front indexes = [-1] * len(population first half)
    for front index, front in enumerate(fronts):
        for solution_index in front:
            solutions_front_indexes[solution_index] = front_index

    # Υπολογισμός αποστάσεων συνωστισμού για κάθε λύση του μισού πληθυσμού
    crowding distances = [0] * len(population first half)
    for front in fronts:
        calculate_front_crowding_distances(front, solutions_objectives_values,
crowding distances)

    solution tuples = []
    for rank, front in enumerate(fronts):
        for solution_index in front:
            solution = population[solution index]
            solution tuples.append((solution index, solution, rank))

    mating pool size = population size // 2 # Μείωση του μεγέθους της δεξαμενής στο μισό
    mating pool = binary tournament selection(solution tuples, crowding distances,
mating_pool_size)

    offsprings population = []  # Αρχικοποίηση της λίστας με λύσεις-απογόνους
    for i in range(0, mating pool size, 2):  # Επαναληπτικό πέρασμα των λύσεων στη δεξαμενή
ζευγαρώματος ανά 2 λύσεις
        parent1, parent1 index = mating pool[i]  # Γονέας 1
        parent2, parent2_index = mating_pool[i + 1]  # Γονέας 2
        # Διασταύρωση γονέων
        flag = False
        both offspring objectives = []
        offsprings = []
        tries = 0  # Προσπάθειες διασταύρωσης των γονέων για τη δημιουργία εφικτών (feasible)
λύσεων-απογόνων
        while not flag:  # Επανάληψη διασταύρωσης των γονέων μέχρι να δημιουργηθούν εφικτές
λύσεις-απόγονοι
            offsprings = customer position alignment crossover(parent1, parent2)  # Δημιουργία
δύο απογόνων
            both_offspring_objectives = []
            result1 = evaluate solution(offsprings[0], distances, travel times, service times,
# Αξιολόγηση απογόνου 1
                                        demands, vehicle capacity, max route time)
            result2 = evaluate solution(offsprings[1], distances, travel times, service times,
# Αξιολόγηση απογόνου 2
                                        demands, vehicle_capacity, max_route_time)
            if result1['feasible'] and result2['feasible']:  # Αν και δύο απόγονοι είναι
εφικτές λύσεις
                objective11, objective12, objective13 = result1['Number of Vehicles'],
result1['Total Distance Travelled'], result1['Total Travel Time']
                both_offspring_objectives.append((objective11, objective12, objective13))
                objective21, objective22, objective23 = result2['Number of Vehicles'],
result2['Total Distance Travelled'], result2['Total Travel Time']
                both offspring objectives.append((objective21, objective22, objective23))
                flag = True
            elif tries == 10:
                flag = True  # Δε βρέθηκαν εφικτές λύσεις, διακοπή της επανάληψης
        if tries != 10:
            for idx, objectives in enumerate(both offspring objectives):
                # Αν ο απόγονος κυριαρχείται από κάποιον γονέα, κρατάμε τον γονέα
                if dominates(solutions_objectives_values[parent1_index], objectives):
                    offsprings_population.append(parent1)
                elif dominates(solutions_objectives_values[parent2_index], objectives):
                    offsprings population.append(parent2)
                else:  # Αν ο απόγονος δεν κυριαρχείται κανέναν από τους δύο γονείς, κρατάμε
τον απόγονο
                    offsprings_population.append(offsprings[idx])
```

```
        else:  # Αν γίνουν tries σε αριθμό προσπάθειες και δε βρεθούν δύο εφικτές λύσεις-απόγονοι, διατήρηση των γονέων
            offsprings population.extend([parent1, parent2])
        if random.random() < mutation rate:  # Μετάλλαξη των δύο τελευταίων λύσεων-απογόνων με πιθανότητα mutation_rate
            for last_solutions_index in ([-1, -2]):
                flag = False
                while not flag:  # Επανάληψη μέχρι να προκύψει εφικτή μεταλλαγμένη λύση
                    mutated solution = mutate(offsprings population[last solutions index])  # Μετάλλαξη της τελευταίας λύσης που προστέθηκε στον πληθυσμό
                    mutated_solution_eval = evaluate_solution(mutated_solution, distances, travel_times, service_times, demands, vehicle_capacity, max_route_time)
                    if mutated solution eval['feasible']:
                        flag = True
                        offsprings population[last solutions index] = mutated solution
    # Εφαρμογή simulated annealing σε κάθε λύση του δεύτερου μισού του πληθυσμού και προσθήκη των αποτελεσμάτων στον πληθυσμό των απογόνων
    for solution in population_second_half:
        initial temperature = 100  # Αρχικοποίηση θερμοκρασίας
        cooling rate = 0.99  # Ρυθμός ψύξης εκθετικός
        max iterations = 50  # Αριθμός επαναλήψεων
        new_solution = simulated_annealing(solution, initial_temperature, cooling_rate, max iterations, distances, travel times, service times, demands, vehicle capacity, max route time)
        offsprings population.append(new solution)

    # Δημιουργία του συνδυασμένου πληθυσμού γονέων-απογόνων
    parents population = [solution for solution,   in mating pool]
    combined population = parents population + offsprings population

    # Αξιολόγηση του συνδυασμένου πληθυσμού
    solutions objectives values = []
    for solution in combined_population:
        result = evaluate solution(solution, distances, travel times, service times,
                                   demands, vehicle capacity, max route time)
        objective1, objective2, objective3 = result['Number of Vehicles'], result['Total Distance Travelled'], result['Total Travel Time']

        solutions_objectives_values.append((objective1, objective2, objective3))  # Δημιουργία tuple με τις τιμές των objectives

    # Μη κυριαρχούμενη ταξινόμηση στον συνδυασμένο πληθυσμό
    fronts = non dominated sorting(solutions objectives values)
    pareto_front = []
    for i in fronts[0]:
        solution = combined population[i]
        if solution not in pareto front:
            pareto front.append(solution)
    print(f"First front solutions number: {len(pareto_front)}")
    # Υπολογισμός αποστάσεων συνωστισμού για κάθε λύση του συνδυασμένου πληθυσμού
    crowding distances = [0] * len(combined population)
    for front in fronts:
        calculate front crowding distances(front, solutions objectives values, crowding distances)
    # Δημιουργία του πληθυσμού της επόμενης γενιάς
    next_population = []
    front index = 0
    while len(next population) < population size:
        solutions in front = [combined population[i] for i in fronts[front index]]
        # Αν η προσθήκη των λύσεων του μετώπου front index δεν ξεπερνάει το population size
        if len(next_population) + len(fronts[front_index]) <= population_size:
            next_population.extend(solutions_in_front)
            front index += 1
        else:
            last front = fronts[front index]
            last_front.sort(key=lambda x: crowding_distances[x], reverse=True)
            next_population.extend(combined_population[i] for i in last_front[:population_size - len(next_population)])
            break
    population = next_population
```

4.2.3.2. Συνάρτηση προσομοιωμένης ανόπτησης simulated_annealing

```
import math
def simulated annealing(initial solution, initial temperature, cooling rate, max iterations,
          distances, travel times, service times, demands, vehicle capacity, max route time):
    current_solution = initial_solution
```

```
    current_temperature = initial_temperature
    # Αρχικοποίηση και αξιολόγηση της καλύτερης λύσης
    best_solution = current_solution
    evaluation = evaluate_solution(best_solution, distances, travel_times, service_times,
demands, vehicle_capacity, max_route_time)
    obj1, obj2, obj3 = evaluation['Number of Vehicles'], evaluation['Total Distance
Travelled'], evaluation['Total Travel Time']
    best_solution_objectives = (obj1, obj2, obj3)
    DE = initial_temperature/10
    for iteration in range(max_iterations):
        # Αξιολόγηση της τρέχουσας λύσης
        evaluation = evaluate_solution(current_solution, distances, travel_times,
service_times, demands, vehicle_capacity, max_route_time)
        obj1, obj2, obj3 = evaluation['Number of Vehicles'], evaluation['Total Distance
Travelled'], evaluation['Total Travel Time']
        current_solution_objectives = (obj1, obj2, obj3)
        # Αρχικοποίηση της γειτονικής λύσης που θα δημιουργηθεί κατά την τρέχουσα επανάληψη
        neighbor_solution = copy.deepcopy(current_solution)
        neighbor_solution_objectives = (0, 0, 0)
        flag = False
        while not flag:  # Επανάληψη έως ότου βρεθεί εφικτή λύση
            neighbor_solution = mutate(current_solution)  # Δημιουργία γειτονικής λύσης
            evaluation = evaluate_solution(neighbor_solution, distances, travel_times,
service_times, demands, vehicle_capacity, max_route_time)
            if evaluation['feasible']:
                obj1, obj2, obj3 = evaluation['Number of Vehicles'], evaluation['Total
Distance Travelled'], evaluation['Total Travel Time']
                neighbor_solution_objectives = (obj1, obj2, obj3)
                flag = True
        if dominates(neighbor_solution_objectives, current_solution_objectives) or
random.random() < math.exp(-DE / current_temperature):
            current_solution = neighbor_solution
            # Αν κυριαρχεί έναντι της καλύτερης έως τώρα, αντικατάσταση της best_solution με
τη γειτονική
            if dominates(neighbor_solution_objectives, best_solution_objectives):
                best_solution = neighbor_solution
                best_solution_objectives = neighbor_solution_objectives
        # Μείωση της θερμοκρασίας
        current_temperature *= cooling_rate
    return best_solution
```

4.2.4. Αποτελέσματα εφαρμογής 2ου προτεινόμενου υβριδικού αλγορίθμου

Τα αποτελέσματα παρουσιάζονται όπως ορίστηκε στον κώδικα παραπάνω για μέγιστο αριθμό γενεών = 100 (κριτήριο τερματισμού), πιθανότητα μετάλλαξης = 20%, μέγεθος πληθυσμού = 500, μέγεθος δεξαμενής ζευγαρώματος = 250, επαναλήψεις προσομοιωμένης ανόπτησης (για κάθε λύση του δεύτερου μισού σε κάθε επανάληψη) = 50, αρχική θερμοκρασία = 100, ρυθμός ψύξης εκθετικός με α = 0,99. Τα αποτελέσματα φαίνονται παρακάτω (ανά 5 γενιές) για ένα τρέξιμο του προγράμματος:

Generation 5
First front solutions number: 16
Generation 10
First front solutions number: 35
Generation 15
First front solutions number: 41
Generation 20
First front solutions number: 52
Generation 25
First front solutions number: 67
Generation 30
First front solutions number: 81

Generation 35
First front solutions number: 62
Generation 40
First front solutions number: 94
Generation 45
First front solutions number: 81
Generation 50
First front solutions number: 93
Generation 55
First front solutions number: 85
Generation 60
First front solutions number: 94
Generation 65
First front solutions number: 85
Generation 70
First front solutions number: 110
Generation 75
First front solutions number: 110
Generation 80
First front solutions number: 119
Generation 85
First front solutions number: 109
Generation 90
First front solutions number: 108
Generation 95
First front solutions number: 122
Generation 100
First front solutions number: 109

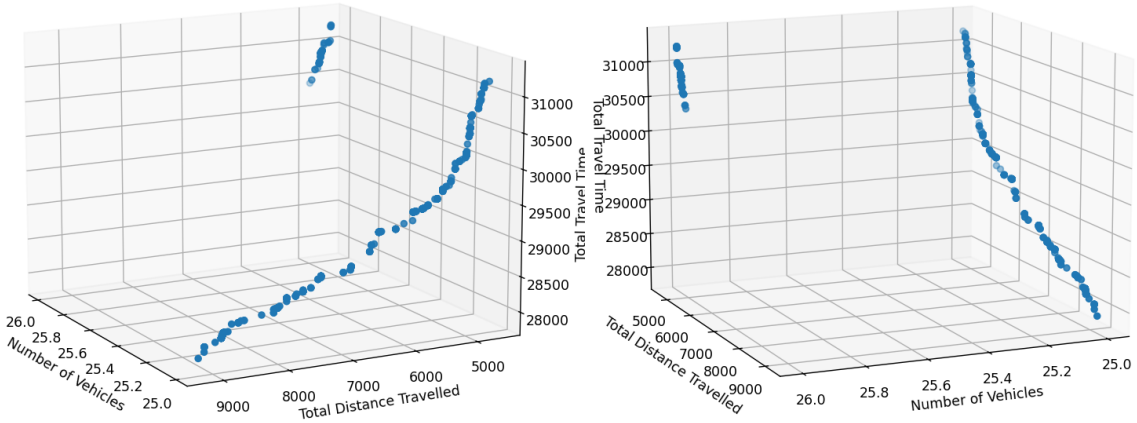
Το πρόγραμμα τερματίστηκε μετά από 100 γενιές, με τις εξής μετρήσεις απόδοσης:

Αριθμός μη κυριαρχούμενων λύσεων: 109

Spacing metric: 0.51938738466126

Hypervolume indicator: 5.857525596246314

Τέλος, η απεικόνιση του μετώπου Pareto (από δύο διαφορετικές γωνίες):



Εικόνα 4.5. *Το μέτωπο Pareto από την εφαρμογή του συνδυασμού NSGA-II και Simulating annealing στο πρόβλημα C_1_2_1*

Για να αξιολογηθεί η αποτελεσματικότητα αλλά και η ευστάθεια του παραπάνω αλγορίθμου απαιτείται επίσης **στατιστική ανάλυση** των αποτελεσμάτων μετά από 20 τρεξίματα του προγράμματος με τις παραμέτρους που ορίστηκαν παραπάνω, η οποία παρουσιάζεται παρακάτω:

Run #	Non-dominated solutions	Spacing metric	Hypervolume indicator
1	109	0,51938738466126	5,85752559624631
2	102	0,41904654573444	6,02919393109076
3	100	0,52308666109602	5,14528739410993
4	110	0,45803596636400	5,09830029721346
5	101	0,42449257255608	6,00003058900980
6	109	0,41233343779124	5,84586561959688
7	111	0,45411575070797	6,07293134049128
8	109	0,50786916545251	5,76653259561227
9	98	0,40399527201729	5,77541148076500
10	91	0,35759828921240	5,53394531573658
11	101	0,40614171913530	5,89712251431238
12	124	0,54194238258975	5,83763519253674
13	100	0,40240422953441	5,58618721080647
14	101	0,44383307477442	5,21041935224098
15	113	0,43734470577816	5,08665580189725
16	95	0,46142769829622	5,88098439307716
17	86	0,40115736239141	5,04782348101118
18	96	0,39684734796209	5,88161100848951
19	91	0,46212763025152	5,82084703847350
20	91	0,38140341945545	5,99096569809859
Μέση τιμή	**101,90**	**0,44**	**5,57**
Τυπική απόκλιση	**9,02**	**0,049**	**0,54**

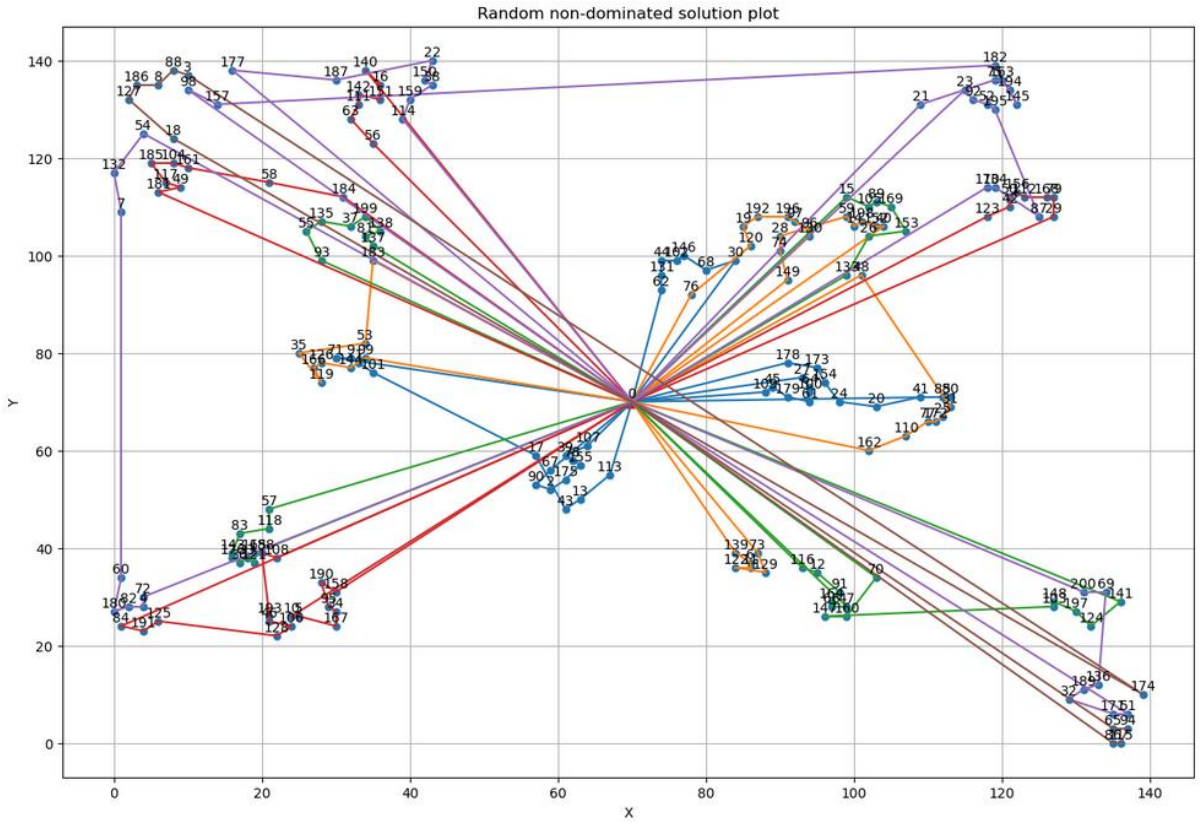
4.2.5. Σχολιασμός αποτελεσμάτων

Όπως γίνεται αντιληπτό από την σύγκριση των αποτελεσμάτων των δύο αλγορίθμων, η εφαρμογή του 1^{ου} προτεινόμενου υβριδικού αλγορίθμου (συνδυασμός NSGA-II και Simulated annealing) σε σύγκριση με τον NSGA-II οδήγησε σε καλύτερα αποτελέσματα, καθώς:

1. Ο αριθμός των (μοναδικών) μη κυριαρχούμενων λύσεων στο μέτωπο Pareto είναι μεγαλύτερος (μέση τιμή 101,90 έναντι 93,55), υποδεικνύοντας μία πιο ευρεία εξερεύνηση του χώρου αναζήτησης και ότι παρέχεται ένα μεγαλύτερο εύρος αντισταθμίσεων μεταξύ των αντικειμενικών συναρτήσεων.

2. Η κατανομή (spacing metric) του μετώπου Pareto έχει μικρότερη τιμή (μέση τιμή 0,44 έναντι 0,57), υποδεικνύοντας ένα πιο ομοιόμορφα κατανεμημένο και με μεγαλύτερη ποικιλομορφία σύνολο λύσεων.
3. Ο υπερόγκος του μετώπου Pareto έχει μεγαλύτερη τιμή (μέση τιμή 5,57 έναντι 3,94), υποδεικνύοντας ένα σύνολο λύσεων με καλύτερη κατανομή στον χώρο αναζήτησης και με καλύτερης ποιότητας λύσεις.

Ενδεικτικά, η απεικόνιση μίας τυχαία επιλεγμένης λύσης από τις μη κυριαρχούμενες λύσεις του μετώπου Pareto που προέκυψε από τη χρήση του 2^{ου} υβριδικού αλγορίθμου φαίνεται παρακάτω, όπου διακρίνονται οι τοποθεσίες των 200 πελατών και οι διαδρομές των (25 σε αριθμό) οχημάτων:



Εικόνα 4.6. *Απεικόνιση μίας μη κυριαρχούμενης λύσης του βέλτιστου μετώπου Pareto*

Βιβλιογραφία

[1] Ιωάννης Μαρινάκης, Αθανάσιος Μυγδαλάς: «Σχεδιασμός και βελτιστοποίηση της εφοδιαστικής αλυσίδας», Εκδόσεις Σοφία, 2008

[2] Ιωάννης Μαρινάκης, Αθανάσιος Μυγδαλάς: «Συνδυαστική βελτιστοποίηση», Πολυτεχνείο Κρήτης, 2004

[3] Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Φ. Κόκκορας, Η. Σακελλαρίου: «Τεχνητή νοημοσύνη – Β Έκδοση»

[4] Harland, C.M. (1996), Supply Chain Management: Relationships, Chains and Networks. British Journal of Management, 7: S63-S80.

[5] Lenstra, Jan & Kan, A.. (2006). Complexity of vehicle routing and scheduling problems. Networks. 11. 221 - 227. 10.1002/net.3230110211

[6] Laporte, Gilbert & Ropke, Stefan & Vidal, Thibaut. (2014). Chapter 4: Heuristics for the Vehicle Routing Problem.

[7] E. Aarts and J.K. Lentra. Local Search in Combinatorial Optimization. Wiley and Sons, 1997.

[8] S.R Thangiah, J.Y. Potvin, and T. Sung. Heuristics approaches to vehicle routing with backhauls and time windows. Computers and Operations Research, 23: 1043-1057, 1996.

[9] J.Pirie Hart, Andrew W. Shogan, Semi-greedy heuristics: An empirical study, Operations Research Letters, Volume 6, Issue 3, 1987, Pages 107-114, ISSN 0167-6377,

[10] Currin, Andrew & Korovin, Konstantin & Ababi, Maria & Roper, Katherine & Kell, Douglas & Day, Philip & King, Ross. (2017). Computing exponentially faster: Implementing a nondeterministic universal Turing machine using DNA. Journal of The Royal Society Interface. 14. 10.1098/rsif.2016.0990.

[11] Lenstra, J.K. and Kan, A.H.G.R. (1981), Complexity of vehicle routing and scheduling problems. Networks, 11: 221-227.

[12] Liu, Wan-Yu & Lin, Chun-Cheng & Chiu, Ching-Ren & Tsao, You-Song & Wang, Qunwei. (2014). Minimizing the Carbon Footprint for the Time-Dependent Heterogeneous-Fleet Vehicle Routing Problem with Alternative Paths. Sustainability. 6. 4658-4684.

[13] Liong choong yeun, et al.,” Vehicle routing problem: models and solutions “, Journal of quality measurement and analysis, JQMA 4(1) 2008, 205-218

[14] Khaoula, Bouanane, Benadada Youssef and Bencheikh Ghizlane. “Multi-Depots Vehicle Routing Problem with Simultaneous Delivery and Pickup and Inventory Restrictions: Formulation and Resolution.” International Journal of Advanced Computer Science and Applications (2019).

[15] Herdianto, Bachtiar and Komarudin. “Guided Clarke and Wright Algorithm to Solve Large Scale of Capacitated Vehicle Routing Problem.” 2021 IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA) (2021): 449-453.

[16] Billy E. Gillett & Leland R. Miller, 1974. "A Heuristic Algorithm for the Vehicle-Dispatch Problem," Operations Research, INFORMS, vol. 22(2), pages 340-349, April.

[17] Euchi, Jalel & Sadok, Abdeljawed. (2021). Hybrid genetic-sweep algorithm to solve the vehicle routing problem with drones. Physical Communication. 44. 101236. 10.1016/j.phycom.2020.101236.

[18] Rajwar, K., Deep, K. & Das, S. An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. Artif Intell Rev 56, 13187–13257 (2023).

[19] Shen, Yang, Mingde Liu, Jian Yang, Yuhui Shi and Martin Middendorf. “A Hybrid Swarm Intelligence Algorithm for Vehicle Routing Problem With Time Windows.” IEEE Access 8 (2020): 93882-93893.

[20] M. Dorigo, M. Birattari and T. Stutzle, "Ant colony optimization," in *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28-39, Nov. 2006

[21] Karaboga, D., Basturk, B. (2007). Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems. In: Melin, P., Castillo, O., Aguilar, L.T., Kacprzyk, J., Pedrycz, W. (eds) Foundations of Fuzzy Logic and Soft Computing. IFSA 2007.

[22] Feo, Thomas A. and Mauricio G. C. Resende. “Greedy Randomized Adaptive Search Procedures.” Journal of Global Optimization 6 (1995): 109-133.

[23] Di Somma, Marialaura. (2016). OPTIMAL OPERATION PLANNING OF DISTRIBUTED ENERGY SYSTEMS THROUGH MULTI-OBJECTIVE APPROACH: A NEW SUSTAINABILITY-ORIENTED PATHWAY.

[24] Waters, C.D.J. (1987) A Solution Procedure for the Vehicle Scheduling Problem Based on Iterative Route Improvement. Journal of Operational Research Society, 38, 833-839.

[25] Qadir, Junaid & Ali, Salman & Vasilakos, Athanasios. (2016). Genetic Algorithms in Wireless Networking: Techniques, Applications, and Issues. Soft Computing. 20. 10.1007/s00500-016-2070-9.

[26] Yu, Ling & Shen, Zhiqi & Miao, Chunyan & Lesser, Victor. (2011). Genetic algorithm aided optimization of hierarchical multiagent system organization. 10th International Conference on Autonomous Agents and Multiagent Systems 2011, AAMAS 2011. 2. 1169-1170.

[27] Yang, Li. (2022). Research on Logistics Distribution Vehicle Path Optimization Based on Simulated Annealing Algorithm. Advances in Multimedia. 2022. 1-8. 10.1155/2022/7363279.

[28] Bajaj, Punam and Harnoor Kaur. “Relevance of artificial bee colony algorithm over other swarm intelligence algorithms.” (2013).

[29] E. Zitzler and L. Thiene. Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. In Conference on Parallel Problem Solving from Nature (PPSN V), pages 292-301, Amsterdam, 1998.

[30] Pereira, J.L.J., Oliver, G.A., Francisco, M.B. et al. A Review of Multi-objective Optimization: Methods and Algorithms in Mechanical Engineering Problems. Arch Computat Methods Eng 29, 2285–2308 (2022).

[31] Hansen, Michael Pilegaard. “Tabu Search for Multiobjective Optimization: MOTS.” (1997).

[32] Adil Baykasoglu, Stephen Owen & Nabil Gindy (1999), A taboo search based approach to find the Pareto optimal set in multiple objective optimization, *Engineering Optimization*, 31:6, 731-748.

[33] Mergos, Panagiotis & Sextos, Anastasios. (2018). Multi-objective optimum selection of ground motion records with genetic algorithms.

[34] Verma, Shanu & Pant, Millie & Snasel, Vaclav. (2021). A Comprehensive Review on NSGA-II for Multi-Objective Combinatorial Optimization Problems. IEEE Access. 9. 10.1109/ACCESS.2021.3070634.

[35] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002

[36] Demir, Ibrahim & Corut Ergin, Fatma & Kiraz, Berna. (2019). A New Model for the Multi-Objective Multiple Allocation Hub Network Design and Routing Problem. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2927418.

[37] Zitzler, Eckart, Marco Laumanns and Lothar Thiele. “SPEA2: Improving the strength pareto evolutionary algorithm.” (2001).

[38] Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, Ludovic Salomon. Performance indicators in multiobjective optimization. European Journal of Operational Research, 2020, 292 (2), pp.397 - 422. 10.1016/j.ejor.2020.11.016. hal-03862074v3.

[39] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Theory of the Hypervolume Indicator: Optimal μ-Distributions and the Choice of the Reference Point. In Foundations of Genetic Algorithms (FOGA 2009), S. 87–102, ACM, New York, NY, USA, 2009.

[40] Li, Miqing & Zheng, Jinhua. (2009). Spread Assessment for Evolutionary Multi-Objective Optimization. 216-230. 10.1007/978-3-642-01020-0_20.