ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΜΕΛΕΤΗ ΕΥΡΕΤΙΚΩΝ ΑΛΓΟΡΙΘΜΩΝ ΤΩΝ OR-TOOLS ΓΙΑ ΤΟ ΠΡΟΒΛΗΜΑ ΔΡΟΜΟΛΟΓΗΣΗΣ ΟΧΗΜΑΤΩΝ ΜΕ ΧΡΟΝΙΚΑ ΠΑΡΑΘΥΡΑ ΚΑΙ ΕΤΕΡΟΓΕΝΗ ΣΤΟΛΟ

Υπό

ΚΩΝΣΤΑΝΤΙΝΟΥ ΚΑΠΗ

Μεταπτυχιακή Εργασία

Υπεβλήθη για την εκπλήρωση μέρους των
απαιτήσεων για την απόκτηση του
Μεταπτυχιακού Διπλώματος Ειδίκευσης

Βόλος, 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

Postgraduate thesis

OR-TOOLS HEURISTIC ALGORITHMS STUDY FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS AND HETEROGENEOUS FLEET.

by

Kapis Konstantinos

September, 2023

Volos

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)	Δρ. Σαχαρίδης Γεώργιος Αναπληρωτής Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Δρ. Αθανάσιος Λόης
Τρίτος Εξεταστής	Δρ. Στέλιος Κουκούμιαλος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας κ. Γεώργιο Σαχαρίδη, για την καθοδήγηση και την συγγραφή της διπλωματικής μου εργασίας. Επίσης, θα ήθελα να ευχαριστήσω θερμά τον κ. Αθανάσιο Λόη για την πολύτιμη βοήθεια του στην ανάπτυξη του κώδικα για το πρόβλημα δρομολόγησης οχημάτων. Ακόμη, είμαι ευγνώμων στον καθηγητή κ. Στυλιανό Κουκούμιαλο, για τις συμβουλές του για τη σωστή συγγραφή της διπλωματικής μου εργασίας .

Οφείλω ένα μεγάλο ευχαριστώ στους εργοδότες μου για όλη την κατανόηση και τη στήριξη που μου δείξανε καθ' όλη την διάρκεια του μεταπτυχιακού μου προγράμματος.

Τέλος, θέλω να ευχαριστήσω τους γονείς μου και την αδερφή μου για την υποστήριξη τους, όχι μόνο στο παρόν μεταπτυχιακό πρόγραμμα, αλλά και σε κάθε βήμα της ζωής μου.

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με ένα από τα σημαντικότερα προβλήματα στον κλάδο της εφοδιαστικής αλυσίδας, αυτό της διανομής προϊόντων και υπηρεσιών. Πιο συγκεκριμένα, ασχολείται με το πρόβλημα δρομολόγησης οχημάτων με χρονικά παράθυρα και ετερογενή στόλο περιορισμένης χωρητικότητας.

Το πρόβλημα δρομολόγησης οχημάτων ανήκει στην κατηγορία NP-hard προβλημάτων βελτιστοποίησης, τα οποία θεωρούνται από τα πιο δύσκολα προβλήματα μη πολυωνυμικού χρόνου. Η εισαγωγή των χρονικών παραθύρων καθιστά ακόμη πιο δύσκολη την επίλυση του, τόσο ως προς τον χρόνο όσο και στην αποτελεσματικότητα της λύσης. Κατά συνέπεια, αναπτύξαμε κώδικα, οποίος με την βοήθεια των Google or-tools και την χρήση κυρίως ευρετικών αλγορίθμων, παράγει καλές λύσεις και σε αποδεκτό χρόνο.

Επιπλέον, για τον υπολογισμό των αποστάσεων μεταξύ των πελατών, δημιουργήσαμε μέσω της Google ένα κλειδί(google geocoding API KEY), στο οποίο εισάγουμε τις ακριβείς τοποθεσίες τόσο των πελατών, όσο και της αποθήκης. Με την βοήθεια Excel υπολογίζουμε ακριβώς όλες τις αποστάσεις των πελατών και της αποθήκης, τόσο σε λεπτά, όσο και μέτρα. Επίσης, εισάγουμε όλα τα απαραίτητα δεδομένα για την επίλυση του προβλήματος στο Excel, όπως την ζήτηση των πελατών, τα χρονικά παράθυρα και τις χωρητικότητες του κάθε οχήματος.

Τέλος, μελετήσαμε τις αποδόσεις των αλγορίθμων σε συνδυασμό με τα metaheuristic για τρείς διαφορετικές περιπτώσεις. Η πρώτη είναι για διαφορετικό αριθμό οχημάτων, η δεύτερη για διαφορετικό αριθμό πελατών και η τρίτη για διαφορετικά χρονοπαράθυρα. Στην κάθε περίπτωση γίνεται παρουσίαση και σύγκριση των αποτελεσμάτων των αλγορίθμων σε συνδυασμό με τα metaheuristic, τόσο για την ποιότητα της λύσης, όσο και για τον χρόνο επίλυσης του εκάστοτε προβλήματος.

Πίνακας Περιεχομένων

Κεφάλαιο 1. ΕΙΣΑΓΩΓΗ

1.1 Πρόβλημα του Πλανόδιου Πωλητή (TSP)

Η πρώτη μαθηματική διατύπωση του προβλήματος του πλανόδιου πωλητή έγινε στις αρχές του 1800 από τον Βρετανό μαθηματικό Thomas Kirkman και τον Ιρλανδό μαθηματικό W.R. Hamilton. Παρ όλα αυτά η προέλευση του προβλήματος του πλανόδιου πωλητή παραμένει μέχρι και σήμερα ασαφής. Ένα εγχειρίδιο για τους πλανόδιους πωλητές από το 1832 αναφέρει το πρόβλημα και περιλαμβάνει παραδείγματα περιηγήσεων στην Γερμανία και στην Ελβετία αλλά δεν περιέχει καμία μαθηματική διατύπωση.

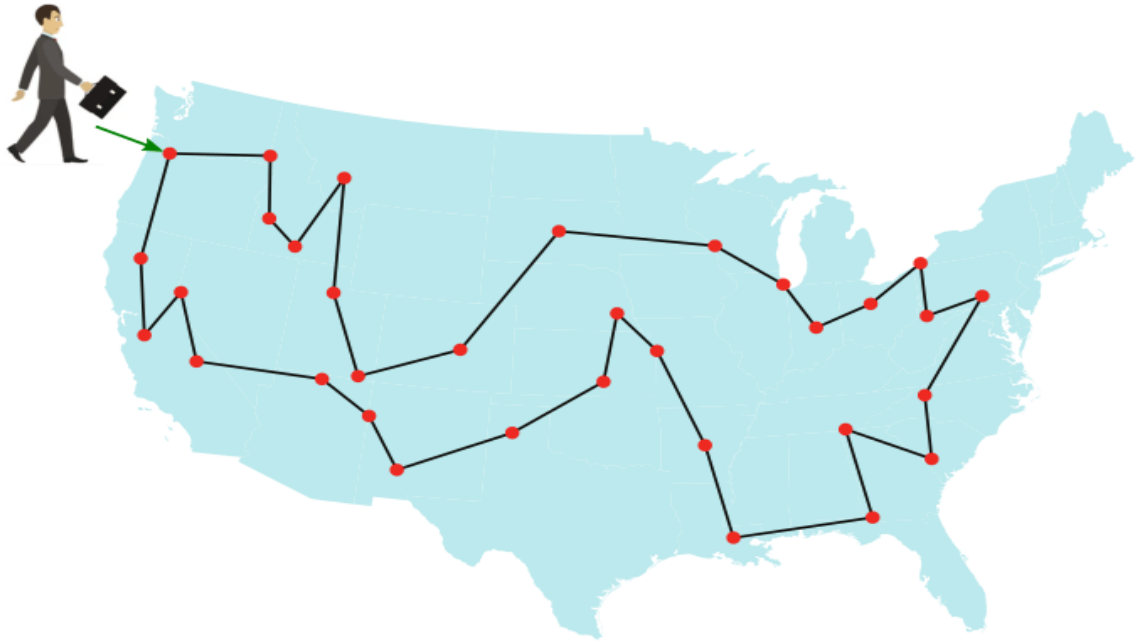
Οι πρώτες μαθηματικές προσεγγίσεις για το πρόβλημα του πλανόδιου πωλητή συναντώνται στην δεκαετία 1930, τόσο στην Βιέννη όσο και στο Χάρβαρντ. Ένας από τους πρώτους μαθηματικούς που προσέγγισε μαθηματικά το πρόβλημα πλανόδιου πωλητή ήταν ο Αμερικάνος Merrill M. Flood που προσπαθούσε να λύσει ένα πρόβλημα δρομολόγησης σχολικών λεωφορείων, το 1930.

Το 1954, οι George Dantzig, Ray Fulkerson και Selmer Johnson πρότειναν τον πρώτο πρακτικό αλγόριθμο για την επίλυση του TSP, ο οποίος είναι γνωστός ως αλγόριθμος Dantzig-Fulkerson-Johnson (DFJ). Αυτός ο αλγόριθμος βασίστηκε στον γραμμικό προγραμματισμό και ήταν σε θέση να λύσει περιπτώσεις του TSP με έως και 49 πόλεις, κάτι που ήταν σημαντικό επίτευγμα για την εκείνη εποχή.

Στη δεκαετία του 1960, οι ερευνητές άρχισαν να αναπτύσσουν ευρετικούς αλγόριθμους για την επίλυση του TSP, οι οποίοι παρείχαν ταχύτερες λύσεις από τον αλγόριθμο DFJ, αλλά δεν εξασφάλιζαν τη βέλτιστη λύση. Ένας από τους πρώτους και πιο διάσημους ευρετικούς αλγόριθμους είναι ο αλγόριθμος του πλησιέστερου γείτονα, ο οποίος προτάθηκε από τον μαθηματικό Merrill Flood το 1956.

Έκτοτε, το TSP παρέμεινε ένας ενεργός τομέας έρευνας, με πολλούς νέους αλγόριθμους και τεχνικές να αναπτύσσονται για την επίλυση όλο και πιο μεγάλων και πολύπλοκων περιπτώσεων του προβλήματος.

Το Πρόβλημα του Πλανόδιου Πωλητή (TSP-Travelling Salesman Problem) θέτει την ακόλουθη ερώτηση: "Λαμβάνοντας υπόψη μια λίστα με τις πόλεις και τις αποστάσεις μεταξύ κάθε ζεύγους πόλεων, ποια είναι η συντομότερη διαδρομή που επισκέπτεται κάθε πόλη και επιστρέφει στην πόλη εκκίνησης; ". Πρόκειται για ένα NP-hard πρόβλημα βελτιστοποίησης στην επιστήμη των υπολογιστών και στα μαθηματικά. Υπάρχουν τρεις κατηγορίες αλγόριθμων που μπορούν να δώσουν λύση σε αυτά τα προβλήματα. Οι ακριβείς αλγόριθμοι (exact algorithms) που βρίσκουν την βέλτιστη λύση, οι προσεγγιστικοί αλγόριθμοι (approximation algorithms), οι οποίοι δίνουν μια προσεγγιστική λύση παρέχοντας εγγυήσεις για το πόσο απέχει από την βέλτιστη λύση και οι ευρετικοί αλγόριθμοι (heuristics) που βρίσκουν γρήγορα μη βέλτιστη λύση, χωρίς να παρέχουν εγγυήσεις για το πόσο απέχει από αυτήν. Αυτοί οι αλγόριθμοι χρησιμοποιούν ευρετικές μεθόδους ή εμπειρικούς κανόνες, για να καθοδηγήσουν την αναζήτηση μιας λύσης. Εμείς στην εργασία μας θα ασχοληθούμε κυρίως με τους ευρετικούς αλγόριθμους.[1]

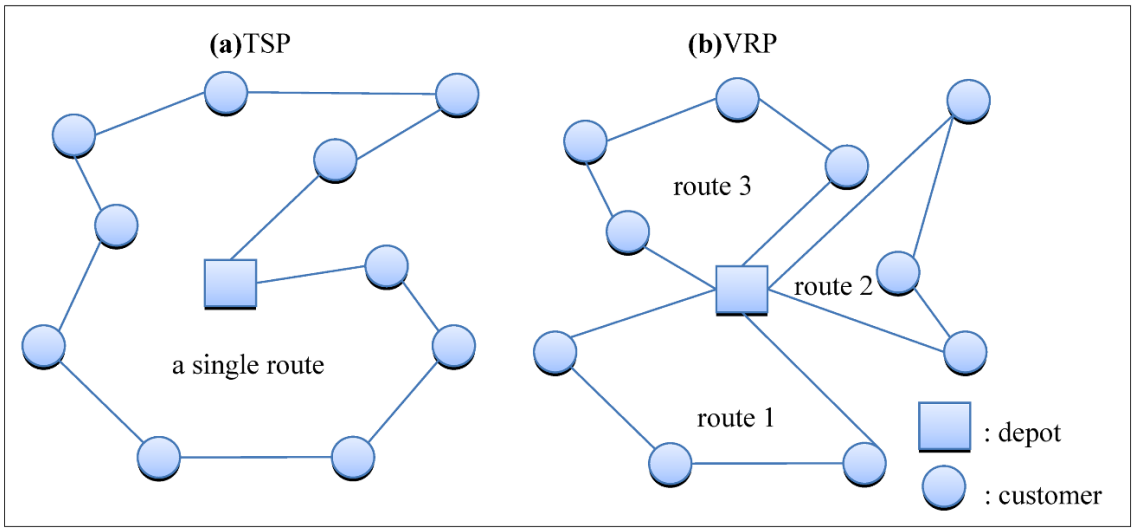


https://www.baeldung.com/cs/tsp-exact-solutions-vs-heuristic-vs-approximation-algorithms

1.2 Πρόβλημα Δρομολόγησης Οχημάτων (VRP)

Το Πρόβλημα Δρομολόγησης Οχημάτων (VRP-vehicle routing problem) είναι γενίκευση το προβλήματος του πλανόδιου πωλητή. Το Πρόβλημα Δρομολόγησης Οχημάτων καθορίζει τις βέλτιστες διαδρομές για έναν στόλο οχημάτων, με σκοπό την καλύτερη εξυπηρέτηση ενός συνόλου πελατών. Ο στόχος του VRP είναι να ελαχιστοποιήσει τη συνολική απόσταση ή το χρόνο που διανύουν τα οχήματα, ικανοποιώντας διάφορους περιορισμούς, όπως περιορισμούς χωρητικότητας, χρονικά παράθυρα και άλλους επιχειρησιακούς περιορισμούς, όπου θα αναλύσουμε παρακάτω.

Ο πρώτος αλγόριθμος για το Πρόβλημα Δρομολόγησης Οχημάτων εμφανίζεται το 1959 σε μια εργασία των Dantzig και ο Ramser, όπου μελέτησαν την μεταφορά βενζίνης στους πελάτες. Το 1964, οι Clarke και Wright βελτίωσαν την προσέγγιση των Dantzig και Ramser χρησιμοποιώντας έναν πιο αποτελεσματικό, άπληστο αλγόριθμο, γνωστό ως savings algorithm, τον οποίο θα δούμε αναλυτικά παρακάτω. [2]



Εικόνα 1.2 Διαφορά ανάμεσα στο TSP και VRP.

https://www.oga.ai/en/blog/vehicle-route-optimization-origins-and-variants/

1.3 Χρήση του VRP στην καθημερινή μας ζωή

Το Πρόβλημα Δρομολόγησης Οχημάτων (VRP) έχει ευρεία χρήση πάνω στην καθημερινή μας ζωή, όπως στην μεταφορά προϊόντων και ανθρώπων, στην διανομή φαγητού και πολλά άλλα. Μερικοί από τους πιο βασικούς κλάδους που εφαρμόζουν το Πρόβλημα Δρομολόγησης Οχημάτων (VRP), είναι οι εξής:

1. Συλλογή απορριμμάτων: Με την χρήση του VRP ο κάθε δήμος μπορεί να βελτιστοποιήσει τις διαδρομές των απορριμματοφόρων και των φορτηγών ανακύκλωσης, μειώνοντας τον χρόνο αποκομιδής των απορριμμάτων και ελαχιστοποιώντας την κατανάλωση καυσίμου. Έτσι, το VRP δημιουργεί τις κατάλληλες διαδρομές οι οποίες έχοντας τις λιγότερο δυνατές αποστάσεις να διανύονται λαμβάνοντας υπόψη την χωρητικότητα των απορριμματοφόρων, το οδικό δίκτυο καθώς επίσης και την πυκνότητα του πληθυσμού. Τέλος, αυτό μπορεί να συμβάλει στη μείωση των περιβαλλοντικών ρύπων, αφού μειώνουμε τις αποστάσεις και τον αριθμό των οχημάτων στο δρόμο.
2. Δημόσιες συγκοινωνίες: Το VRP μπορεί να βοηθήσει τους οργανισμούς δημόσιας συγκοινωνίας να βελτιστοποιήσουν τα δρομολόγια λεωφορείων και τρένων, βελτιώνοντας την αξιοπιστία των υπηρεσιών και μειώνοντας τον χρόνο ταξιδιού για τους επιβάτες, πράγμα πολύ σημαντικό ειδικά για τους εργαζόμενους. Αυτό μπορεί επίσης να βοηθήσει στη μείωση της κυκλοφοριακής συμφόρησης και τη μείωση των ρύπων.
3. Οχήματα έκτακτης ανάγκης: Το VRP μπορεί να χρησιμοποιηθεί από ομάδες απόκρισης έκτακτης ανάγκης, όπως η αστυνομία, η πυροσβεστική και τα ασθενοφόρα, για τη βελτιστοποίηση των διαδρομών και των χρόνων απόκρισης σε καταστάσεις έκτακτης ανάγκης. Αυτό μπορεί να βελτιώσει την αποτελεσματικότητα των υπηρεσιών έκτακτης ανάγκης και ενδεχομένως να σώσει ζωές.
4. Διανομή τροφίμων: Πρόκειται για μια από τις πιο απαιτητικές υπηρεσίες, αφού τα περισσότερα τρόφιμα έχουν περιορισμένο χρόνο ζωής. Για τον λόγο αυτό, οι υπηρεσίες διανομής θα πρέπει να εξασφαλίζουν την ποιότητα, την ασφάλεια, τη θερμοκρασία και την υγεία του προϊόντος, καθώς επίσης, θα

πρέπει να το παραδώσουν στον ελάχιστο δυνατό χρόνο. Μια γρήγορη διανομή διασφαλίζει ότι το φαγητό είναι ακόμα ζεστό, φρέσκο και έτοιμο για κατανάλωση. Οι υπηρεσίες διανομής τροφίμων είναι άκρως ανταγωνιστικές μεταξύ τους, αφού η γρήγορη διανομή αποτελεί βασικό παράγοντα διαφοροποίησης που ξεχωρίζει μια υπηρεσία(π.χ.BOX,efood,Wolt)-κατάστημα από τους ανταγωνιστές της. Η παροχή ταχύτερων χρόνων παράδοσης προσελκύει περισσοτέρους πελάτες και μπορεί να βοηθήσει μια υπηρεσία – κατάστημα να αποκτήσει ανταγωνιστικό αποτέλεσμα έναντι των υπολοίπων.

5. Γενικού τύπου μεταφορές προϊόντων :Πρόκειται κυρίως για παγκόσμιες μεταφορές προϊόντων, όπου η ταχεία μεταφορά αγαθών συμβάλει στην διατήρηση του παγκοσμίου εμπορίου. Σε αυτήν την κατηγορία είναι πολύ σημαντικό για την επιχείρηση να διανέμει βέλτιστα τα προϊόντα της , ώστε να φτάνουν σωστά και έγκαιρα στον πελάτη. Η βέλτιστη διανομή, μπορεί να μειώσει σημαντικά τα κόστη της επιχείρησης, αν αναλογιστούμε ότι το 10%-40% της τιμής ενός προϊόντος αφορά την μεταφορά του.
6. Military Logistics: Το VRP μπορεί να χρησιμοποιηθεί από στρατιωτικές ομάδες επιμελητείας για τη βελτιστοποίηση της μεταφοράς στρατευμάτων, προμηθειών και εξοπλισμού, διασφαλίζοντας ότι φτάνουν στους προορισμούς τους έγκαιρα και αποτελεσματικά.

Αυτά είναι μόνο μερικά παραδείγματα από τις πολλές εφαρμογές του Προβλήματος Δρομολόγησης Οχημάτων (VRP) στην καθημερινή μας ζωή. Οι δυνατότητες είναι ατελείωτες και καθώς η τεχνολογία προχωρά, μπορούμε να περιμένουμε να δούμε ακόμα πιο καινοτόμες εφαρμογές του VRP στην καθημερινή μας ζωή, αφού τα οφέλη του είναι πάρα πολλά, όπως μείωση κόστους, καλύτερη εξυπηρέτηση πελατών και μείωση των ρύπων.

Κεφάλαιο 2. ΜΟΝΤΕΛΟΠΟΙΗΣΗ

2.1 Βασικό μοντέλο του Προβλήματος Δρομολόγησης Οχημάτων (VRP)

Για την μαθηματική μοντελοποίηση του προβλήματος δρομολόγησης οχημάτων θα χρειαστούμε αρχικά να ορίσουμε κάποια σύνολα και δείκτες. Το οδικό δίκτυο διανομής μπορεί να περιγράφει χρησιμοποιώντας ένα γράφημα G, όπου τα τόξα είναι οι δρόμοι του οδικού δικτύου και οι κορυφές που είναι οι διασταυρώσεις των τόξων, είναι οι πελάτες και η αποθήκη. Κατά συνέπεια, ορίζουμε ένα σύνολο κόμβων N, όπου ο κόμβος 0 αντιπροσωπεύει την αποθήκη και οι κόμβοι 1,2,…,N αντιπροσωπεύουν τους πελάτες. Επίσης, η εταιρεία διανομής προϊόντων διαθέτει ένα στόλο οχημάτων K. Ορίζουμε ως K το σύνολο των οχημάτων που έχει στην διάθεση της η εταιρεία για την διανομή των προϊόντων της και Qk την χωρητικότητα του κάθε οχήματος. Εάν ο στόλος της εταιρείας είναι ομογενής, την χωρητικότητα την ορίζουμε ως Q, αφού είναι ίδια για όλα τα οχήματα.

Με τους δείκτες i,j ορίζουμε τους πελάτες για i, j =1,2,3,…,N, ενώ για i, j = 0 ορίζουμε την αποθήκη. Οι δείκτες i, j μας βοηθάνε ώστε να ξέρουμε κάθε φορά, κάθε μεταβλητή και κάθε δεδομένο σε ποιόν πελάτη αναφέρεται ή το όχημα σε ποιόν πελάτη είναι και σε ποιόν κατευθύνεται. Πιο συγκεκριμένα, ορίζουμε ως Cijk το κόστος μετακίνησης από τον πελάτη i στον πελάτη j. Όπως γίνεται αντιληπτό το i αντιπροσωπεύει τον κόμβο που ξεκινάει ένα όχημα, j την πόλη προορισμού και k το ακριβές όχημα που εκτελεί το συγκεκριμένο δρομολόγιο.

Στην συνέχεια, ορίζουμε την αντικειμενική συνάρτηση, η οποία υποδηλώνει τον κύριο σκοπό του προβλήματος δρομολόγησης οχημάτων. Συνήθως, ο σκοπός του προβλήματος δρομολόγησης οχημάτων είναι η ελαχιστοποίηση των διαδρομών, δηλαδή την συνολική απόσταση που θα διανύσουν τα οχήματα. Αξίζει να σημειωθεί, ότι πολλές φορές για την δρομολόγηση των οχημάτων μπορεί να ληφθούν υπόψη πολλοί και αντικρουόμενοι στόχοι, όπως η εξισορρόπηση των διαδρομών, όσο αναφορά το χρόνο ταξιδιού και το φορτίο του οχήματος ή η ελαχιστοποίηση του

αριθμού των οχημάτων που απαιτούνται για την εξυπηρέτηση όλων των πελατών και πολλοί άλλοι.

Τέλος, πρέπει να εξασφαλίσουμε την ομαλή επεξεργασία των στοιχείων από το πρόγραμμα και να αποφύγουμε λανθασμένες διαδρομές. Έτσι, προκειμένου το πρόγραμμα να μην δημιουργήσει δρομολόγια που δεν ξεκινούν και καταλήγουν στην αποθήκη ή δρομολόγια που δεν έχουν λογική συνοχή μεταξύ τους, είναι απαραίτητη η χρήση περιορισμών. Οι περιορισμοί μας εξασφαλίζουν ότι η χωρητικότητα των οχημάτων δεν παραβιάζεται, τα οχήματα ξεκινάνε και τελειώνουν στην αποθήκη, ότι όλοι οι πελάτες θα εξυπηρετηθούν, κάθε φορτηγό επισκέπτεται έναν πελάτη ακριβώς μια φορά και άλλους επιχειρησιακούς περιορισμούς ανάλογα με το είδος και σκοπό του προβλήματος δρομολόγησης οχημάτων.

- **Σύνολα:**

    1. Σύνολο οχημάτων K
    2. Σύνολο κόμβων N
    3. Δίκτυο διανομής G=(N,E)

- **Δείκτες:**

    1. Σημείο έναρξης (αποθήκη):

        $i,j = 0$

    2. Σύνολο κόμβων :

        $i,j \in N$, $V = A \cup \{0\}$, $i,j = 0,1,\ldots,n$

    3. Οχήματα-Στόλος:

        $k \in K$, $k = 0,1,\ldots,m$

    4. Πελάτες-Κόμβοι:

        $i,j \in A$, $i,j = 1,2,\ldots,n$

- **Δεδομένα:**

1. Qk: χωρητικότητα του οχήματος K
2. qi: ζήτηση του πελάτη i
3. qj: ζήτηση του πελάτη j
4. Cijk: κόστος μετακίνησης από τον πελάτη i στον πελάτη j
5. Tij: χρόνος μετακίνησης από τον πελάτη i στον πελάτη j
6. Tej: χρόνος εξυπηρέτησης του πελάτη j
7. M: ένας μεγάλος αριθμός (M=10000)
8. FCij: οι ρύποι που εκπέμπει ένα φορτηγό ανά μονάδα απόστασης, λόγω της μετάβασης από τον πελάτη i στον πελάτη j.

- **Μεταβλητή απόφασης:**

1. Xijk: Δυαδική μεταβλητή απόφασης που ισούται με 1, αν το όχημα k μετακινηθεί από τον πελάτη i στον πελάτη j, αλλιώς ισούται με 0.
2. Dijk: Ακέραια μεταβλητή που δηλώνει το βάρος του φορτίου k μεταξύ των πελατών i και j.
3. U_j: Ακέραια βοηθητική μεταβλητή που την χρησιμοποιούμε στον περιορισμό για την εξάλειψη των υποδιαδρομών.

- **Αντικειμενική συνάρτηση:**

1. Min $\sum_k \sum_i \sum_j (C_{ij} * X_{ijk})$

   Στόχος της αντικειμενικής συνάρτησης είναι η ελαχιστοποίηση του κόστος μεταφοράς των προϊόντων, προς τους πελάτες.

2. Min $\sum_k \sum_i \sum_j (FC_{ij} * X_{ijk})$

   Στόχος της αντικειμενικής συνάρτησης είναι η ελαχιστοποίηση εκπομπής ρύπων από την μεταφορά των προϊόντων, προς τους πελάτες.

3. Min $\sum_k \sum_j X_{0jk}$

Στόχος της αντικειμενικής συνάρτησης είναι η ελαχιστοποίηση του αριθμού φορτηγών που θα χρησιμοποιηθούν για την μεταφορά των προϊόντων, προς τους πελάτες.

- **Περιορισμοί:**

1. Αυτός ο περιορισμός εξασφαλίζει ότι κάθε πελάτης μπορεί να εξυπηρετηθεί μόνο μια φορά, από ένα φορτηγό k.

$$\sum_{j=0}^{N} \sum_{k=1}^{K} Xijk=1, \quad i \neq j \quad \forall i \in N$$

2. Εξασφαλίζει ότι κάθε όχημα k κάνει το πολύ ένα δρομολόγιο.

$$\sum_{j=1}^{N} X0jk \leq 1, \forall k \in K$$

3. Εξασφαλίζει ότι όταν ένα φορτηγό k επισκεφτεί έναν πελάτη θα πρέπει να αναχωρήσει από το ίδιο σημείο για να συνεχίσει το δρομολόγιο του.

$$\sum_{i=0}^{N} Xihk - \sum_{j=0}^{N} Xhjk = 0 \quad \forall h \in N \quad \forall k \in K$$

4. Οι συνολικές ζητήσεις των πελατών που ικανοποιούνται σε ένα δρομολόγιο οχήματος k δεν πρέπει να ξεπερνούν την συνολική την συνολική χωρητικότητα του οχήματος αυτού, Q_k.

$$\sum_{i=0}^{N} \sum_{j=1}^{N} q_j X_{ijk} <= Q_k \quad \forall \quad k \in K$$

5. Δεν επιτρέπει να γίνει μετάβαση από τον πελάτη i στον πελάτη j όταν i=j

$$\sum_{i=0}^{N} X_{iik}=0 \ \forall k \in K$$

6. Κάθε όχημα k πρέπει να ξεκινάει από την αποθήκη – depot.

$$\sum_{i=1}^{N} \sum_{j=0}^{N} X_{ijk} <= M \sum_{h=1}^{N} X_{0hk}$$

7. Εξάλειψη υποδιαδρομών.

$U_i - U_j + (N-1) \times X_{ijk} <= N-2 \; \forall \; k \in K \; , \forall \; i,j \in N$

8. Κάθε όχημα κατά την έναρξη του δρομολογίου, θα πρέπει να έχει φορτίο ίσο με την ζήτηση των πελατών που θα εξυπηρετήσει.

$$\sum_{j=1}^{N} \mathrm{Dijk} = \sum_{i=0}^{N} \sum_{j=1}^{N} \mathrm{Xijk} \times q_j \quad \forall \; k \in K$$

9. Κάθε όχημα θα πρέπει να επιστρέψει στην αποθήκη-depot με μηδενικό φορτίο.

$$\sum_{i=0}^{N} D_{i0k} = 0 \quad \forall \; k \in K$$

10. Η ζήτηση ενός πελάτη j, ισούται με το συνολικό φορτίο ενός οχήματος k πριν την εξυπηρέτησή του πελάτη j, μείον το συνολικό φορτίο του οχήματος k μετά την εξυπηρέτησή του πελάτη j.

$$\sum_{i=0}^{N} \sum_{k=1}^{K} \mathrm{Dijk} - \sum_{i=0}^{N} \sum_{k=1}^{K} \mathrm{Djik} = q_j \quad \forall \; j \in N$$

11. Για κάθε όχημα k ο συνολικός χρόνος μετακίνησης και εξυπηρέτησης των πελατών δεν πρέπει να υπερβαίνει τις οχτώ ώρες.

$$\sum_{i=0}^{N} \sum_{j=0}^{N} \mathrm{Xijk} \times (T_{ij} + Te_j) <= 480 \quad \forall \; k \in K$$

12. Η χωρητικότητα του οχήματος k πρέπει να είναι μεγαλύτερη του μηδενός.

$$Q_k > 0 \; \forall \; k \in K$$

13. Η ζήτηση των πελατών i και j, καθώς και το φορτίο από τον i στον j πρέπει να είναι μεγαλύτερες ή ίσες με το μηδέν.

$$q_i \; , \; q_j, \; D_{ijk} \geq 0 \quad \forall \; k \in K \; , \forall \; i,j \in N$$

2.2 Είδη Προβλήματος Δρομολόγησης Οχημάτων (VRP)

Εκτός από την βασική του μορφή, το πρόβλημα δρομολόγησης οχημάτων (VRP) μπορεί να εκφράσει πολλές παραλλαγές και εξειδικεύσεις, ανάλογα με την εκάστοτε υπηρεσία και τους εκάστοτε επιχειρησιακούς περιορισμούς. Η κάθε παραλλαγή έχει τα δικά της χαρακτηριστικά και στόχους.

2.2.1 Capacitated VRP (CVRP) – Πρόβλημα δρομολόγησης οχημάτων με περιορισμένη χωρητικότητα

Το πρόβλημα δρομολόγησης οχημάτων με περιορισμένη χωρητικότητα (CVRP) είναι μια παραλλαγή του προβλήματος δρομολόγησης οχημάτων (VRP), όπου ένας στόλος οχημάτων με περιορισμένη χωρητικότητα χρησιμοποιείται για την παράδοση αγαθών ή υπηρεσιών σε ένα σύνολο πελατών. Κάθε όχημα έχει έναν περιορισμό μέγιστης χωρητικότητας, που συνήθως αντιπροσωπεύεται είτε από βάρος είτε από όγκο, ο οποίος περιορίζει την ποσότητα των αγαθών που μπορεί να μεταφέρει. Στόχος του CVRP είναι να βρεθεί το βέλτιστο σύνολο διαδρομών για τα οχήματα που θα εξυπηρετήσουν τους εκάστοτε πελάτες, τηρώντας τον περιορισμό χωρητικότητας του κάθε οχήματος. Ο περιορισμός χωρητικότητας διασφαλίζει ότι οι συνολικές απαιτήσεις-ζητήσεις των πελατών που εκχωρούνται σε ένα όχημα δεν ξεπερνάνε την χωρητικότητα του. [3]

2.2.2 Πρόβλημα δρομολόγησης οχημάτων με χρονικά παράθυρα (VRPTW)

Μια πολύ συνηθισμένη παραλλαγή του προβλήματος δρομολόγησης οχημάτων (VRP), είναι πρόβλημα δρομολόγησης οχημάτων με χρονικά παράθυρα (VRPTW). Στο VRPTW κάθε πελάτης συσχετίζεται με ένα χρονικό παράθυρο, το οποίο

καθορίζει το επιτρεπόμενο χρονικό εύρος, στο οποίο μπορεί να τον επισκεφτεί ή να τον εξυπηρετήσει ένα όχημα. Έκτος από τα χρονικά παράθυρα, το VRPTW μπορεί να έχει και άλλους χρονικούς περιορισμούς, όπως μέγιστες ώρες εργασίας για τους οδηγούς και τα οχήματα ή μέγιστο χρόνο εξυπηρέτησης ανά πελάτη. Όπως γίνεται αντιληπτό, με την εισαγωγή του χρόνου στο πρόβλημα θα χρειαστούμε επιπλέον περιορισμούς και μεταβλητές, πράγμα που καθιστά το πρόβλημα ακόμη πιο περίπλοκό αυξάνοντας τον χρόνο επίλυσης του. Τέλος, στόχος του VRPTW είναι να βρεθούν οι βέλτιστες διαδρομές που ελαχιστοποιούν το συνολικό κόστος, το χρόνο ή την απόσταση, ικανοποιώντας τα χρονικά παράθυρα των πελατών. [4]

2.2.3 Πρόβλημα δρομολόγησης οχημάτων με παραλαβές και παραδώσεις- Vehicle Routing Problem with Pick up and Delivery (VRPPD)

Το πρόβλημα δρομολόγησης οχημάτων με παραλαβές και παραδώσεις (VRPPD) είναι ένα πολύπλοκο πρόβλημα συνδυαστικής βελτιστοποίησης, που περιλαμβάνει όχι μόνο την παράδοση αγαθών και υπηρεσιών σε πελάτες αλλά και την παραλαβή τους από τις διάφορες τοποθεσίες. Στο VRPPD τα οχήματα είναι υπεύθυνα τόσο για τις εργασίες παραλαβής όσο και για τις εργασίες παράδοσης, με στόχο την βελτιστοποίηση των διάδρομών , ικανοποιώντας τους εκάστοτε περιορισμούς.

Η θεμελιώδης διαφορά ανάμεσα στο VRPPD και στις άλλες παραλλαγές του VRP είναι η αυξομείωση της χωρητικότητας του οχήματος σε κάθε πελάτη. Για τον κάθε πελάτη ορίζουμε ακόμη μια μεταβλητή απόφασης την Pijk, η οποία είναι συνεχής μεταβλητή που εκφράζει την ποσότητα που έχει συλλεχθεί από το όχημα k ($\forall\ k \in K$) έως και το σημείο i ($\forall\ i \in N$), ενώ διασχίζει την διαδρομή (i,j). Επίσης θα χρειαστούμε τρεις επιπλέον περιορισμούς.

1. Η χωρητικότητα του οχήματος k δεν ξεπερνιέται ποτέ.

   $Pijk + Dijk \leq Qk \times Xijk \quad \forall\ k \in K\ ,\ \forall\ i,j \in N$

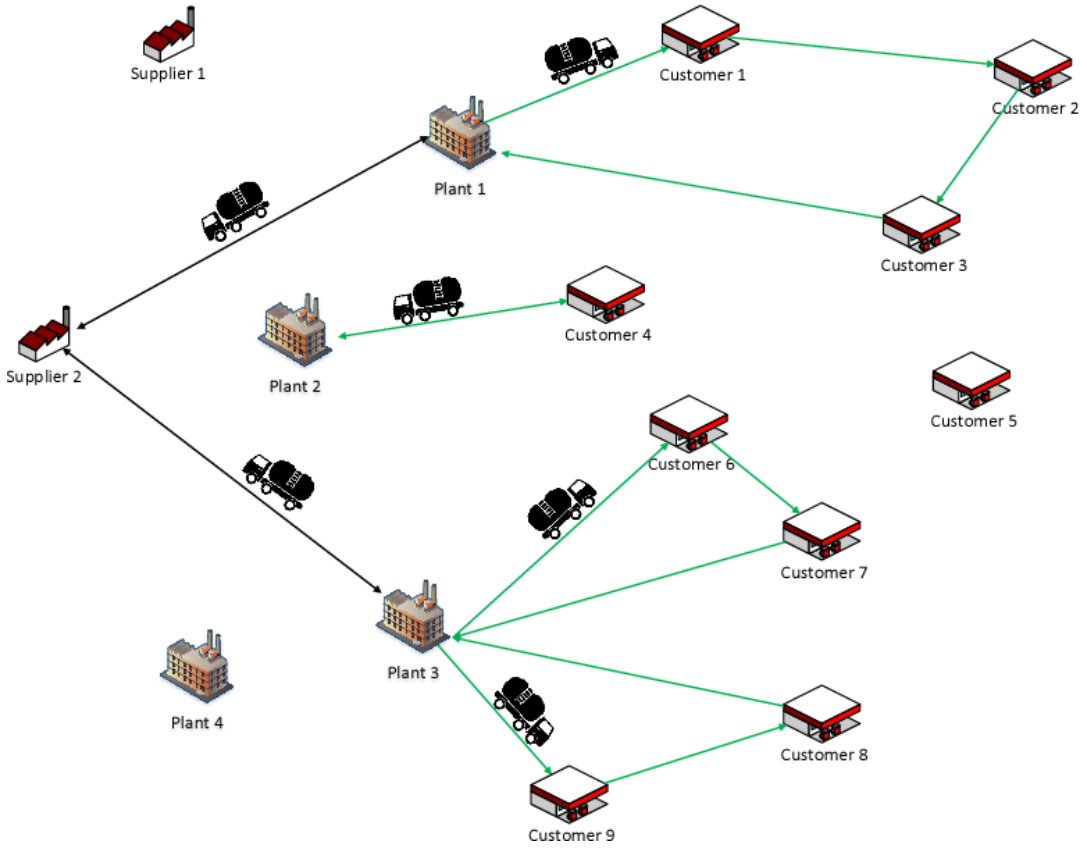
2. Η ποσότητα παραλαβής από ένα σημείο j, ισούται με το συνολικό φορτίο ενός οχήματος k μετά την παραλαβή από το σημείο j, μείον το συνολικό φορτίο του οχήματος k πριν την παραλαβή από το σημείο j.

$$\sum_{i=0}^{N}\sum_{k=1}^{K} Pjik - \sum_{i=0}^{N}\sum_{k=1}^{K} Pijk = p_j \quad \forall\, j \in N$$

3. Η ζήτηση ενός πελάτη j, ισούται με το συνολικό φορτίο ενός οχήματος k πριν την εξυπηρέτησή του πελάτη j, μείον το συνολικό φορτίο του οχήματος k μετά την εξυπηρέτησή του πελάτη j.

$$\sum_{i=0}^{N}\sum_{k=1}^{K} Dijk - \sum_{i=0}^{N}\sum_{k=1}^{K} Djik = q_j \quad \forall\, j \in N$$

Αξίζει να σημειώσουμε ότι σε κάποιες περιπτώσεις του VRPPD θα πρέπει να λάβουμε υπόψη τους περιορισμούς συμβατότητας, οι οποίοι εξασφαλίζουν ότι ορισμένες παραλαβές πρέπει να εκτελούνται πριν από τις αντίστοιχες παραδώσεις. [5]



Εικόνα 2.2.3 Πρόβλημα δρομολόγησης οχημάτων με παραλαβές και παραδώσεις

https://shop14030.radicalgest.com/category?name=inventory%20routing

2.2.4 Πρόβλημα δρομολόγησης ετερογενών οχημάτων- Vehicle Routing Problem with Heterogeneous Fleets (VRPHF)

Στο πρόβλημα δρομολόγησης ετερογενών οχημάτων (VRPHF), ο στόλος αποτελείται από διαφορετικούς τύπους οχημάτων με διαφορετικά χαρακτηριστικά. Τα οχήματα του στόλου ενδέχεται να διαφέρουν ως προς το κόστος, τη χωρητικότητα, την ταχύτητα, τις εκπομπές ρύπων, στο είδος του φορτίου που μεταφέρει, όπως για παράδειγμα αν το φορτίο είναι ψυχρό ή ξηρό, ή άλλα χαρακτηριστικά. Η ετερογένεια το στόλου προκαλεί διακυμάνσεις στο κόστος όπως η κατανάλωση καυσίμου, η συντήρηση ή τα λειτουργικά κόστη. Ο στόχος είναι να ελαχιστοποιηθεί το συνολικό κόστος του σχεδίου δρομολόγησης, λαμβάνοντας υπόψη το διαφορετικό κόστος που σχετίζεται με κάθε τύπο οχήματος, καθώς και τους εκάστοτε επιχειρησιακούς περιορισμούς.

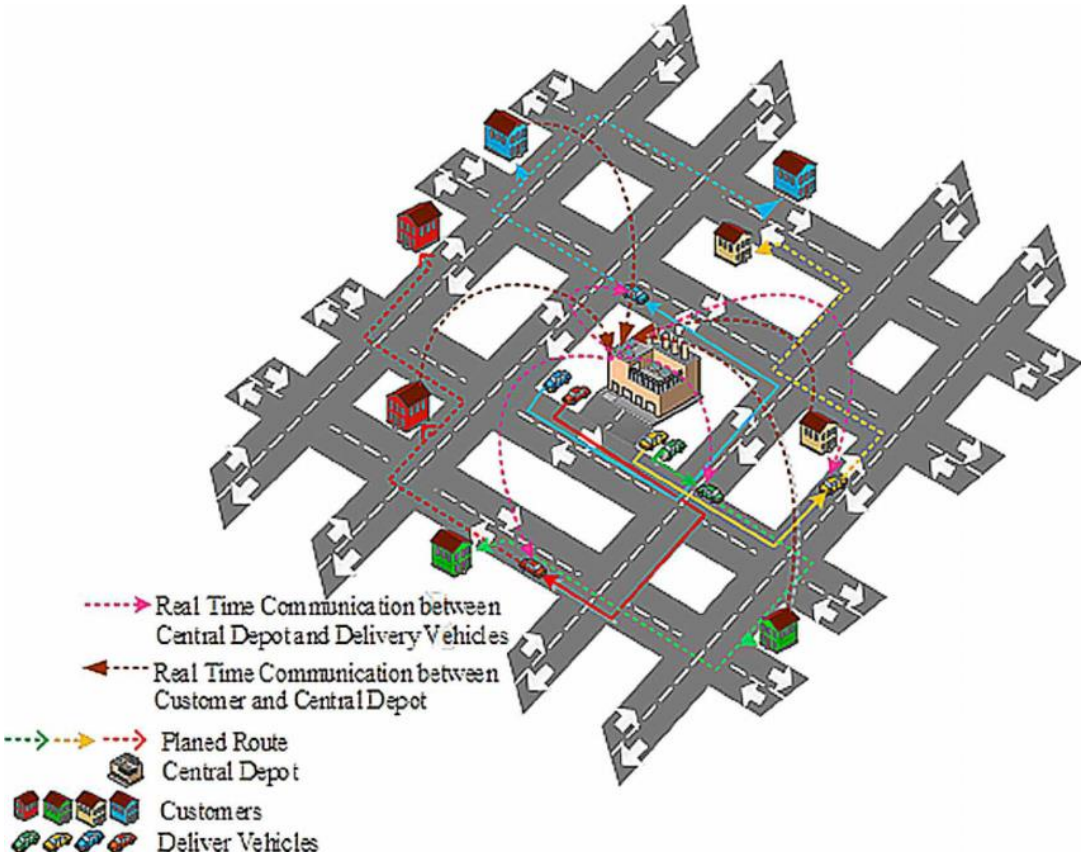
Πολλές φορές το VRPHF μπορεί να περιλαμβάνει πολλαπλούς αντικρουόμενους στόχους βελτιστοποίησης, όπως η ελαχιστοποίηση του κόστους, η εξισορρόπηση του φόρτου εργασίας σε διαφορετικούς τύπους οχημάτων, η ταχύτητα εξυπηρέτησης των πελατών ή ελαχιστοποίηση εκπομπών ρύπων. Αυτό καθιστά το πρόβλημα ακόμη πιο περίπλοκο αυξάνοντας σημαντικά τον χρόνο επίλυσης του. Για την επίλυση του χρησιμοποιούμε διάφορες προσεγγίσεις βελτιστοποίησης, όπως ο μαθηματικός προγραμματισμός, η μεταευρετική (π.χ. ευρετικούς αλγόριθμους, προσομοιωμένη ανόπτηση) ή υβριδικές μέθοδοι. [6]

2.2.5 Πρόβλημα δυναμικής δρομολόγησης οχημάτων- Dynamic Vehicle Routing Problem (DVRP)

Το Δυναμικό Πρόβλημα Δρομολόγησης Οχημάτων (DVRP) αντιπροσωπεύει δυναμικές και σε πραγματικό χρόνο αλλαγές στις απαιτήσεις των πελατών, στη διαθεσιμότητα οχημάτων ή άλλους παράγοντες κατά τον σχεδιασμό και στην εκτέλεση των διαδρομών. Στο DVRP, οι αποφάσεις δρομολόγησης πρέπει να λαμβάνονται δυναμικά καθώς νέες πληροφορίες γίνονται διαθέσιμες, επιτρέποντας προσαρμοστικές και αποκρινόμενες λύσεις δρομολόγησης.

Το DVRP λαμβάνει υπόψη του τη διαθεσιμότητα των οχημάτων σε πραγματικό χρόνο. Τα οχήματα ενδέχεται να μην είναι διαθέσιμα λόγων βλαβών, καθυστερήσεων ή άλλων απρόβλεπτων περιστάσεων, που απαιτούν εκ νέου ανάθεση εργασιών ή οχημάτων για την διατήρηση της αποτελεσματικής διαδρομής. Επίσης, σημαντικές παράμετροι στο DVRP αποτελούν οι παραλλαγές στις συνθήκες κυκλοφορίας, όπως η συμφόρηση ή το κλείσιμο τω δρόμων που μπορεί να επηρεάσουν τους χρόνους ταξιδιού. Το DVRP χρησιμοποιεί πληροφορίες κυκλοφορίας σε πραγματικό χρόνο ή ιστορικά δεδομένα, για την ενημέρωση διαδρομών και την βελτιστοποίηση τους.

Το DVRP απαιτεί εκ νέου βελτιστοποίηση διαδρομών και αναθέσεων κάθε φορά που συμβαίνει κάποια αλλαγή, όπως καινούργιες παραγγελίες πελατών ή ακυρώσεις. Αυτή η δυναμική εκ νέου βελτιστοποίηση στοχεύει στην ελαχιστοποίηση του κόστους, των αποστάσεων ταξιδιού ή άλλων κριτηρίων απόδοσης ως απάντηση στις μεταβαλλόμενες συνθήκες. Τέλος πολύ σημαντική είναι η διαχείριση επικοινωνίας και πληροφοριών. Το DVRP βασίζεται σε αποτελεσματικά συστήματα επικοινωνίας και διαχείρισης πληροφοριών για τη μετάδοση δεδομένων σε πραγματικό χρόνο μεταξύ οχημάτων, πελατών και κεντρικών κέντρων ελέγχου (π.χ. GPS σε κάθε όχημα, ώστε να γνωρίζει να πάσα στιγμή το κέντρου ελέγχου που βρίσκεται κάθε όχημα). Αυτό επιτρέπει έγκαιρες ενημερώσεις, συντονισμό και δυναμική λήψη αποφάσεων στη διαδικασία δρομολόγησης. [7]



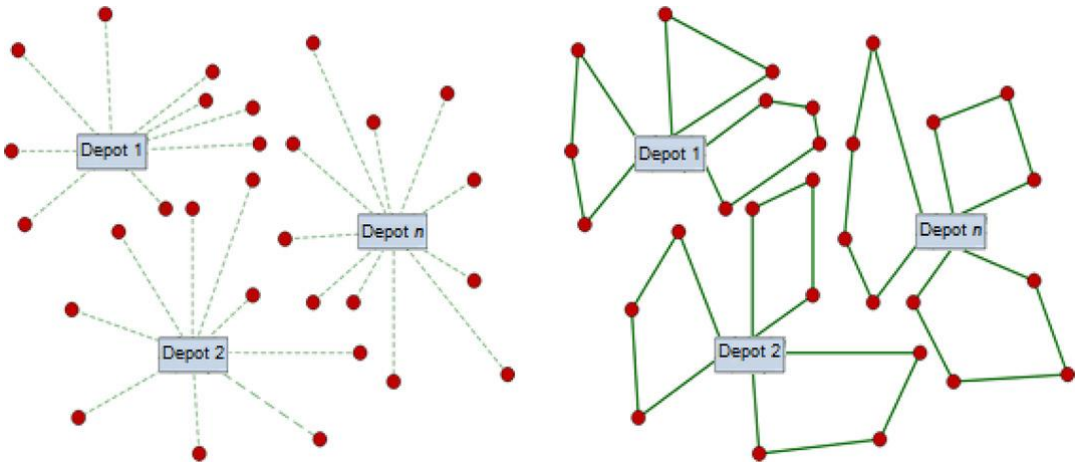
Εικόνα 2.2.5 Πρόβλημα δυναμικής δρομολόγησης οχημάτων.

https://www.researchgate.net/figure/Dynamic-Vehicle-Routing-Problem-DVRP_fig2_292335464

2.2.6 Πρόβλημα δρομολόγησης οχημάτων με πολλαπλές αποθήκες- Multi-Depot Vehicle Routing Problem (MDVRP)

Το MDVRP περιλαμβάνει πολλαπλές αποθήκες ή κέντρα διανομής όπου σταθμεύουν τα οχήματα και ξεκινούν τα δρομολόγια τους. Κάθε αποθήκη έχει το δικό της στόλο οχημάτων και μπορεί να εξυπηρετήσει ένα υποσύνολο πελατών εντός της καθορισμένης γεωγραφικής περιοχής. Το πρόβλημα βελτιστοποιεί τον προσδιορισμό των οχημάτων από την κάθε αποθήκη που θα πρέπει να ανατεθούν για να εξυπηρετήσουν τους εκάστοτε πελάτες. Αξίζει να σημειωθεί ότι το MDVRP περιλαμβάνει όχι μόνο δρομολόγια εντός της καθορισμένης γεωγραφικής περιοχής της κάθε αποθήκης, αλλά αρκετές φορές περιλαμβάνει επισκέψεις μεταξύ αποθηκών. Αυτό επιτρέπει την εξισορρόπηση του αποθέματος, την ανακατανομή των

πόρων ή ακόμα και την ενοποίηση των αποστολών μεταξύ των αποθηκών, με αποτέλεσμα τη βελτιστοποίηση της δρομολόγησης των οχημάτων. [8]



Εικόνα 2.2.6 Πρόβλημα δρομολόγησης οχημάτων με πολλαπλές αποθήκες
https://onlinelibrary.wiley.com/doi/10.1111/itor.12560

2.2.7 Το στοχαστικό πρόβλημα δρομολόγησης οχημάτων - The Stochastic Vehicle Routing Problem (SVRP)

Το στοχαστικό πρόβλημα δρομολόγησης οχημάτων ενσωματώνει την αβεβαιότητα και την τυχαιότητα στην διατύπωση του προβλήματος. Σε αυτήν την περίπτωση ένα ή περισσότερα στοιχεία τους προβλήματος είναι τυχαία. Πιο συγκεκριμένα χωρίζουμε τα στοιχεία αυτά σε τρείς κατηγορίες:

1. Στοχαστικοί πελάτες: Ορίζουμε ως Pi την πιθανότητα ο i πελάτης να πραγματοποιήσει παραγγελία και 1−Pi την πιθανότητα να μην πραγματοποιήσει παραγγελία.
2. Στοχαστικές ζητήσεις πελατών: Ορίζουμε ως di την ζήτηση κάθε πελάτη i, όπου di μια τυχαία μεταβλητή.
3. Στοχαστικοί χρόνοι: Ορίζουμε τυχαίες μεταβλητές Si τον χρόνο εξυπηρέτησης ενός πελάτη i και Tij το χρόνο ταξιδιού από τον πελάτη i στον πελάτη j.

Η επίλυση του SVRP γίνεται σε δυο στάδια. Στο πρώτο στάδιο καθορίζεται μια πρώτη λύση χωρίς να λαμβάνουμε υπόψη τις τυχαίες μεταβλητές. Στο δεύτερο στάδιο γνωρίζοντας πλέον τις τιμές των τυχαίων μεταβλητών επανα-βελτιστοποιούμε την αρχική λύση.

Το στοχαστικό πρόβλημα δρομολόγησης οχημάτων έχει ως αντικειμενική συνάρτηση $\sum_{i\leq j} c_{ij} X_{ij} + Q_x$, με στόχο την ελαχιστοποίηση της.

- Ορίζουμε ακέραια μεταβλητή Xij, η οποία μα δείχνει πόσες φόρες εμφανίζεται ο κάθε κόμβος-πελάτης στην πρώτη λύση. Εάν i,j ≥1 τότε η Xij μπορεί να πάρει μόνο τις τιμές 0 και 1. Εάν i=1 τότε η Xij παίρνει την τιμή 2 και δηλώνει ότι το όχημα επιστρέφει στην αποθήκη από τον κόμβο Vj.
- Με την Qx εισερχόμαστε στο δεύτερο στάδιο επίλυσης του προβλήματος. Σε αυτό το στάδιο το πρόβλημα είναι πλέον εξαρτώμενο στις συνθήκες των αλλαγών που πραγματοποιούνται. Για παράδειγμα, στο SVRP περιορισμένης χωρητικότητας με παραλαβές , πιθανές ενέργειες προσφυγής είναι οι εξής:
    1. Όταν το όχημα γεμίσει επιστρέψτε στην αποθήκη για να το ξεφορτώσετε και στην συνέχεια συνεχίστε το δρομολόγιο από εκεί που σταματήσατε.
    2. Όταν το όχημα γεμίσει επιστρέψτε στην αποθήκη για να το ξεφορτώσετε, αλλά τώρα κάνε βελτιστοποίηση της υπόλοιπης διαδρομής.
    3. Προγραμματισμός προληπτικής επιστροφής στην αποθήκη ακόμα και αν το όχημα δεν είναι πλήρες. Σε αυτήν την περίπτωση, η απόφαση εξαρτάται από το πόσο φορτίο έχει το όχημα την δεδομένη στιγμή ή την απόσταση μεταξύ αποθήκης και οχήματος.
    4. Επέστρεψε στην αποθήκη ακόμη και αν το όχημα δεν είναι γεμάτο, μόνο εάν γνωρίζεις ότι η παραλαβή στον επόμενο κόμβο-πελάτη , θα ανάγκαζε το όχημα να ξεπεράσει την χωρητικότητα του.

Επειδή ορισμένα στοιχεία και μεταβλητές του προβλήματος μεταβάλλονται τυχαία, δεν είναι δυνατόν να ικανοποιούνται όλοι οι περιορισμοί του προβλήματος για όλες τις τιμές των τυχαίων μεταβλητών. Για τον λόγο αυτόν ο υφιστάμενος θα

πρέπει να λάβει απόφαση εάν θα γίνει ικανοποίηση κάποιων από τους περιορισμούς ή πραγματοποίηση διορθωτικών ενεργειών στην περίπτωση που κάποιος περιορισμός παραβιάζεται. [9]

2.2.8 Πρόβλημα περιοδικής δρομολόγησης οχημάτων στοχαστικό - The Periodic Vehicle Routing Problem (PVRP)

Στο κλασικό πρόβλημα δρομολόγησης οχημάτων αλλά και στις περισσότερες παραλλαγές του, η περίοδος προγραμματισμού των δρομολογίων είναι μια μέρα. Στην περίπτωση όμως του PVRP, η περίοδος προγραμματισμού των δρομολογίων επεκτείνεται σε περισσότερες από μια ημέρες (M-days). Στην περίπτωση όπου η περίοδος προγραμματισμού M=1, δηλαδή οι πελάτες έχουν δεδομένες ζητήσεις σε καθημερινή βάση, τότε το PVRP ταυτίζεται με το κλασικό VRP, αφού η δρομολόγηση γίνεται κάθε ημέρα.

Στόχος του PVRP είναι ο σχεδιασμός ημερήσιων δρομολογίων, με σκοπό την ελαχιστοποίηση του συνολικού κόστους και την ικανοποίηση όλων των επιχειρησιακών περιορισμών. Η επίλυση του PVRP αποτελείται από τα εξής τρία στάδια:

1. Στο πρώτο στάδιο δημιουργούνται ομάδες διαφορετικών εφικτών συνδυασμών για κάθε πελάτη. Έστω ότι η περίοδος προγραμματισμού είναι τρείς μέρες, M=3. Τότε οι εφικτοί συνδυασμοί εξυπηρέτησης θα είναι οι εξής:

    1. (0,0,0)
    2. (1,0,0)
    3. (0,1,0)
    4. (0,0,1)
    5. (1,1,0)
    6. (0,1,1)

7. (1,0,1)
8. (1,1,1)

Με 1 ορίζουμε την πραγματοποίηση της διαδρομής και με 0 την μη πραγματοποίηση της διαδρομής για την αντίστοιχη ημέρα. Με βάση τον αριθμό επισκέψεων που πρέπει να πραγματοποιηθούνε στην περίοδο προγραμματισμού, προκύπτει ο ακολούθως πίνακας.

Πελάτες	Αριθμός Επισκέψεων	Πλήθος Συνδυασμών	Πιθανοί Συνδυασμοί
1	1	3	2,3,4
2	2	3	5,6,7
3	3	1	8

2. Στο δεύτερο στάδιο θα πρέπει να επιλεγεί μια από τις εφικτές λύσεις για κάθε πελάτη που βρέθηκαν στο στάδιο ένα, χωρίς όμως να παραβιάζονται οι εκάστοτε περιορισμοί. Συνεπώς θα πρέπει να επιλεγούν οι πελάτες που θα δεχθούν επίσκεψη για κάθε μια ημέρα της περιόδου.

3. Στο τελευταίο στάδιο λύνουμε το VRP για κάθε ημέρα της περιόδου. [10]

Κεφάλαιο 3. ΑΛΓΌΡΙΘΜΟΙ

Σε αυτό το κεφάλαιο θα δούμε τους αλγόριθμος που χρησιμοποιήσαμε καθώς και βήματα τους.

3.1 Αλγόριθμος Savings

Ο αλγόριθμος savings εφευρέθηκε από τους G.Clarke και J.W.Wright το 1964 και ανήκει στην κατηγορία των ευρετικών αλγορίθμων που χρησιμοποιούνται ευρέως στο Πρόβλημα Δρομολόγησης Οχημάτων. Τα βήμα τα του αλγορίθμου είναι τα εξής:

Βήμα 1: Φτιάξε μια αρχική λύση όπου κάθε πελάτης εξυπηρετείται από ένα n όχημα.

Βήμα 2: Για όλους τους πελάτες - ζεύγη που υπάρχουν i,j , i≠j υπολόγιζε τα savings – κέρδος της κάθε διαδρομής. Όπου savings $s_{ij} = c_{0i} + c_{0j} - c_{i,j}$.

Βήμα 3: Ταξινόμησε τα savings(κέρδος κάθε διαδρομής) προς φθίνουσα σειρά

Βήμα 4: Πάρε το πρώτο τόξο [i,j] από την λίστα των savings. Σύνδεσε τις δυο κορυφές-πελάτες μόνο εάν:

i) Οι κορυφές-πελάτες ανήκουν σε διαφορετικές διαδρομές
ii) Η χωρητικότητα του οχήματος δεν παραβιάζεται
iii) Τα i και j είναι ο πρώτος ή ο τελευταίος πελάτης της διαδρομής

Βήμα 5: Επανέλαβε το βήμα 4 μέχρι να ολοκληρωθεί η λίστα των savings ή η χωρητικότητα των οχημάτων να μην επιτρέπει άλλη συγχώνευση δρομολογίου. [11]

3.2 Αλγόριθμος Path Cheapest Arc

Ο αλγόριθμος Path Cheapest Arc εφευρέθηκε από τους Ravindra K. Ahuja, Thomas L. Magnanti και James B. Orlin. Ανάπτυξαν τον αλγόριθμο και τον παρουσίασαν στο βιβλίο τους με τίτλο "Network Flows: Theory, Algorithms, and Applications", το οποίο δημοσιεύτηκε το 1993. Οι Ravindra K. Ahuja, Thomas L. Magnanti και James B. Orlin είναι ειδικοί στον τομέα της βελτιστοποίησης και έχουν συμβάλει σημαντικά στους αλγορίθμους ροής δικτύου, με τις μελέτες και τις ιδέες τους.

Ο αλγόριθμος Path Cheapest Arc χρησιμοποιείται για την επίλυση προβλημάτων ροής ελαχίστου κόστους επιλέγοντας κάθε φορά τα φθηνότερα τόξα κατά μήκος των μονοπατιών του υπολειπόμενου δικτύου μέχρι να βρεθεί μια προσεγγιστική λύση. Αποτελεί ένα βασικό συστατικό στην οικογένεια των αλγορίθμων που χρησιμοποιούνται για την επίλυση διαφόρων προβλημάτων ροής δικτύου, όπως το πρόβλημα του πλανόδιου πωλητή.

Τα βήμα τα του αλγορίθμου είναι τα εξής:

Βήμα 1: Από ένα σύνολο κόμβων-πελατών επέλεξε τυχαία έναν, ως αρχικό

Βήμα 2: Από τον κόμβο-πελάτη που επέλεξες ως αρχικό, εξέτασε όλα τα πιθανά τόξα-διαδρομές που συνδέουν τον τρέχοντα κόμβο-πελάτη, με τους κόμβους-πελάτες που δεν έχει επισκεφτεί και υπολόγισε το κόστος (απόσταση, χρόνο ή βάρος) κάθε τόξου.

Βήμα 3: Επέλεξε το τόξο με το χαμηλότερο κόστος και πρόσθεσε το στην υπάρχουσα διαδρομή, συνδέοντας του δύο κόμβους-πελάτες.

Βήμα 4: Αφαίρεσε τον επιλεγμένο κόμβο από τους μη-συνδεδεμένους κόμβους και πρόσθεσέ τον στους συνδεδεμένους κόμβους.

Βήμα 5: Επανέλαβε τα βήματα 2 έως 4 μέχρι όλοι οι κόμβοι-πελάτες να γίνουν επισκέψιμοι. [12]

3.3 Αλγόριθμος Local Cheapest Insertion

Ο αλγόριθμος Local Cheapest Insertion είναι ένας ευρετικός αλγόριθμος, που χρησιμοποιείται ευρέως για την λύση του προβλήματος του πλανόδιου πωλητή. Είναι αποτέλεσμα επαναληπτικής ανάπτυξης και συνεισφορών από πολλούς ερευνητές και επαγγελματίες στον τομέα της βελτιστοποίησης και των συνδυαστικών αλγορίθμων. Ο κάθε ερευνητής βασισμένος στις προηγούμενες μελέτες, εισάγει τις δικές του ιδέες και βελτιώσεις. Κατά συνέπεια, είναι δύσκολο ο αλγόριθμος να αποδοθεί σε ένα συγκεκριμένο άτομο ή ομάδα.

Η ανάπτυξη του είναι μια συνεχής προσπάθεια που συνεχίζεται ακόμη και σήμερα, καθώς οι ερευνητές συνεχίζουν να εξερευνούν νέες τεχνικές και παραλλαγές για να βελτιώσουν την απόδοση και την αποτελεσματικότητα του, ενσωματώνοντας πληροφορίες από πεδία όπως η θεωρία των γράφων, η βελτιστοποίηση και οι αλγόριθμοι.

Στον αλγόριθμο Local Cheapest Insertion, οι πόλεις εισάγονται στην τρέχουσα περιήγηση βήμα-βήμα. Ο αλγόριθμος ξεκινά με έναν τυχαίο κόμβο, ως κόμβο εκκίνησης, και στη συνέχεια, επιλέγει τους βήμα-βήμα τους υπόλοιπους κόμβους και τους εισάγει στην τρέχουσα περιήγηση με βάση το φθηνότερο τοπικό τους κόστος. Πιο συγκεκριμένα τα βήμα τα του αλγορίθμου είναι τα εξής:

Βήμα 1: Επέλεξε αυθαίρετα έναν κόμβο-πελάτη εκκίνησης

Βήμα 2: Υπολόγισε το κόστος εισαγωγής κάθε μη συνδεδεμένου κόμβου-πελάτη σε κάθε πιθανή θέση στο δρομολόγιο.

Βήμα 3: Επέλεξε τον κόμβο-πελάτη που έχει ως αποτέλεσμα το χαμηλότερο κόστος και εισήγαγε τον στην επιλεγμένη θέση στο δρομολόγιο

Βήμα 4: Επανάλαβε τα βήματα 2 έως 3 μέχρι όλοι οι πελάτες να είναι συνδεδεμένοι σε ένα δρομολόγιο.

Βήμα 5: Δοκίμασε ανταλλαγές άκρων ανά ζεύγη στο δρομολόγιο, ώστε να μειωθεί το κόστος του.

Βήμα 6: Επανάλαβε το βήμα 5 έως ότου δεν είναι δυνατή η περεταίρω βελτίωση του δρομολογίου.

Αξίζει να σημειωθεί ότι ο αλγόριθμος Local Cheapest Insertion, εισάγει τους κόμβους βήμα-βήμα, ενημερώνοντας συνεχώς την περιήγηση με βάση τις τοπικά βέλτιστες επιλογές. Αυτή η επαναληπτική διαδικασία βοηθά στη δημιουργία μιας προσεγγιστικής λύσης που επιδιώκει να ελαχιστοποιήσει τη συνολική απόσταση στο πρόβλημα του πλανόδιου πωλητή. [13]

3.4 Αλγόριθμος Local Cheapest Arc

Ο αλγόριθμος Local Cheapest Arc, είναι και αυτός ένας ευρετικός αλγόριθμος που χρησιμοποιείται ευρέως για την επίλυση του προβλήματος του πλανόδιου πωλητή, παρέχοντας αξιόπιστες λύσεις. Έχει εξελιχθεί και αναπτυχθεί από τις συλλογικές προσπάθειες των ερευνητών από τις αρχές του 20ου αιώνα έως και σήμερα.

Ο αλγόριθμος Local Cheapest Arc, επιλέγει ένα μη συνδεδεμένο κόμβο-πελάτη και τον συνδέει με τον κόμβο-πελάτη που παράγει το φθηνότερο τμήμα της διαδρομής. Πιο συγκεκριμένα τα βήματα του αλγορίθμου είναι τα εξής:

Βήμα 1: Επέλεξε έναν αρχικό κόμβο-πελάτη, οποίος να μην είναι συνδεδεμένος με άλλον κόμβο-πελάτη.

Βήμα 2: Βρες τον κόμβο-πελάτη που βρίσκεται πιο κοντά στον επιλεγμένο κόμβο-πελάτη με βάση την απόσταση, χρόνο, κόστος (ανάλογα τι εξετάζουμε κάθε φορά).

Βήμα 3: Αξιολόγησε το κόστος εισαγωγής του επιλεγμένου κόμβου-πελάτη σε κάθε υπάρχουσα διαδρομή ή δημιούργησε μια καινούργια.

Βήμα 4: Επέλεξε την φθηνότερη επιλογή εισαγωγής στην διαδρομή με βάση την αξιολόγηση κόστους, χρόνου, απόστασης.

Βήμα 5: Ενημέρωσε την λύση εισάγοντας τον επιλεγμένο κόμβο-πελάτη στο δρομολόγιο.

Βήμα 6: Επανάλαβε τα βήματα 2-5 μέχρι όλοι οι κόμβοι-πελάτες να αντιστοιχούν σε ένα δρομολόγιο. [14]

3.5 Αλγόριθμος Global Cheapest Arc

Ένας ακόμη αποτελεσματικός ευρετικός αλγόριθμος είναι ο Global Cheapest Arc. Η ανάπτυξη του μπορεί να θεωρηθεί ως μια συλλογική προσπάθεια εντός της

ερευνητικής κοινότητας, με συνεισφορές από πολλούς επιστήμονες και ερευνητές όλα αυτά τα χρόνια.

Ο Global Cheapest Arc αλγόριθμος συνδέει επαναληπτικά δυο κόμβους-πελάτες, που παράγουν το φθηνότερο τμήμα διαδρομής. Τα βήμα τα του αλγορίθμου είναι τα εξής:

Βήμα 1: Ξεκίνα από μια κενή λύση.

Βήμα 2: Υπολόγισε το κόστος όλων των πιθανών τόξων μεταξύ κόμβων-πελατών, με βάση την απόσταση, χρόνο, βάρος ή κόστος ταξιδιού.

Βήμα 3: Ταξινόμησε τα τόξα σε αύξουσα σειρά κόστους.

Βήμα 4: Βρες μέσα από τα ταξινομημένα τόξα, αυτό που παράγει το φθηνότερο τμήμα της διαδρομής και δοκίμασε το στην λύση-δρομολόγιο.

Βήμα 5: Εάν το τόξο δεν παραβιάζει περιορισμούς όπως χρονικά παράθυρα, χωρητικότητα και άλλους επιχειρησιακούς περιορισμούς, πρόσθεσέ το στην λύση-δρομολόγιο.

Βήμα 6: Επανάλαβε τα βήματα 4-5 μέχρι να εξεταστούν όλα τα τόξα ή μέχρι να εκπληρωθεί μια συνθήκη τερματισμού, δηλαδή όλοι οι κόμβοι-πελάτες να έχουν εκχωρηθεί σε ένα δρομολόγιο. [14]

3.6 Αλγόριθμος Parallel Cheapest Insertion

Ο αλγόριθμος Parallel Cheapest Insertion, ανήκει και αυτός στην κατηγορία των ευρετικών αλγορίθμων και είναι μια παραλλαγή του αλγόριθμου, Best Insertion.

Δημιουργεί επαναληπτικά μια λύση εισάγοντας τον φθηνότερο κόμβο στην τρέχουσα λύση. Το κόστος εισαγωγής βασίζεται στην συνάρτηση κόστους τόξου και είναι πιο γρήγορος από τον αλγόριθμο Best Insertion.

Τα βήμα τα του αλγορίθμου είναι τα εξής:

Βήμα 1: Χώρισε τους πελάτες σε υποσύνολα ίσου μεγέθους, όπου κάθε υποσύνολο θα υποστεί επεξεργασία, από μια ξεχωριστή μονάδα. Ο αριθμός των υποσυνόλων πρέπει να αντιστοιχεί στον αριθμό διαθέσιμων μονάδων.

Βήμα 2: Εκχωρήστε σε κάθε μονάδα επεξεργασίας ένα υποσύνολο πόλεων για να εργαστείτε. Κάθε μονάδα θα εκτελέσει ανεξάρτητα τον αλγόριθμο Cheapest Insertion στο υποσύνολο της.

Βήμα 3: Δημιουργήστε μια κοινόχρηστη δομή δεδομένων για την αποθήκευση των ενδιάμεσων αποτελεσμάτων. Αυτή η κοινή δομή δεδομένων θα πρέπει να είναι προσβάσιμη από όλες τις μονάδες επεξεργασίας.

Βήμα 4: Μετά από κάθε επανάληψη του αλγορίθμου, όλες οι μονάδες επεξεργασίας συγχρονίζονται και μοιράζονται τις καλύτερες τοπικές περιηγήσεις.

Βήμα 5: Μόλις όλες οι μονάδες επεξεργασίας ολοκληρώσουν τις επαναλήψεις τους, μια μονάδα επεξεργασίας συλλέγει τις καλύτερες περιηγήσεις από όλες τις μονάδες επεξεργασίας και επιλέγει την καλύτερη συνολική περιήγηση, η οποία αντιπροσωπεύει και την λύση στο TSP ή VRP. [15]

3.7 Αλγόριθμος Christofides

Ο αλγόριθμος του Χριστοφίδη εφευρέθηκε το 1976 και πηρέ το όνομα του από τον Νίκο Χριστοφίδη, οποίος ήταν και αυτός που τον εφηύρε. Ανήκει την κατηγορία των προσεγγιστικών αλγορίθμων καθώς εγγυάται ότι η λύση που παρέχει είναι το πολύ 1,5 φορά χειρότερη από την βέλτιστη λύση.

Για να μπορέσουμε να εφαρμόσουμε τον αλγόριθμο του Χριστοφίδη θα πρέπει να ισχύουν οι τρείς εξής προϋποθέσεις:

i) Το δίκτυο μας να είναι συμμετρικό
ii) Να ισχύει η τριγωνική ανισότητα
iii) Το δίκτυο να είναι πλήρως συνδεδεμένο

Εάν ισχύουν αυτές οι τρείς προϋποθέσεις μπορέσουμε να εφαρμόσουμε τον αλγόριθμο του Χριστοφίδη και τα βήματα του αλγορίθμου είναι τα εξής:

Βήμα 1: Δημιούργησε το ελάχιστο spanning tree και ονόμασε το T.

Βήμα 2: Μετά την δημιουργία του spanning tree θα εμφανιστούν n κόμβοι περιττού βαθμού (οι οποίοι είναι πάντα περιττός ο αριθμός). Βρες το ελάχιστο-φθηνότερο ζευγάρωμα των κόμβων αυτών.

Βήμα 3: Το σύνολο των κόμβων που προκύπτουν από το ζευγάρωμα των κόμβων περιττού βαθμού ανήκουν στο σύνολο Μ. Ένωσε το σύνολο Μ με το spanning tree T και δημιούργησε ένα νέο γράφημα Η(H=MUT). Το νέο γράφημα Η που προκύπτει είναι ένα Eulerian path.

Βήμα 4: Μετατρέψτε το Eulerian path σε Hamiltonian κύκλο παρακάμπτοντας τις επαναλαμβανόμενες κορυφές. Ο Hamiltonian κύκλος που προκύπτει, είναι η λύση του TSP. [16]

3.8 Αλγόριθμος Path Most Constraint Arc

Τα βήματα του αλγορίθμου Path Most Constraint Arc είναι παρόμοια με τα βήματα του Path Cheapest Arc, με την μόνη διαφορά ότι ο Path Most Constraint Arc επιλέγει το τόξο με τους περισσότερους περιορισμούς, ενώ Path Cheapest Arc επιλέγει κάθε φορά το τόξο με το χαμηλότερο κόστος. Πιο συγκεκριμένα, τα βήματα του αλγορίθμου είναι τα εξής:

Βήμα 1: Ξεκίνησε με ένα κενό δρομολόγιο και ένα μη εκχωρημένο σύνολο πελατών ή τοποθεσιών.

Βήμα 2: Για κάθε πελάτη ή τοποθεσία που δεν έχει εκχωρηθεί, υπολόγισε τον αριθμό των περιορισμών που έχει με άλλες μη εκχωρημένες τοποθεσίες, λαμβάνοντας υπόψη παράγοντες όπως η απόσταση, ο χρόνος ή τους περιορισμούς χωρητικότητας.

Βήμα 3: Από όλους τους μη εκχωρημένους πελάτες, επέλεξε αυτόν με τους περισσότερους περιορισμούς. Σε περίπτωση ισοπαλίας περιορισμών μεταξύ πελατών, εφάρμοσε “ευρετικά ισοπαλίας” με βάση παράγοντες όπως η απόσταση

από τον τρέχοντα πελάτη, η ζήτηση, η χωρητικότητα ή οποιοδήποτε άλλον περιορισμό σχετίζεται με το πρόβλημα δρομολόγησης οχημάτων.

Βήμα 4: Πρόσθεσε τον επιλεγμένο πελάτη στο κατάλληλο δρομολόγιο λαμβάνοντας υπόψη τους χρονικούς περιορισμούς, τη χωρητικότητα του οχήματος και τυχόν άλλους σχετικούς περιορισμούς του εκάστοτε προβλήματος δρομολόγησης οχημάτων.

Βήμα 5: Ενημέρωσε τα δεδομένα που σχετίζονται με το δρομολόγιο του κάθε οχήματος, όπως την τρέχον χωρητικότητα του, την απόσταση που πρέπει να διανύσει και τον χρόνο που απαιτείται.

Βήμα 6: Εάν η ανάθεση του πελάτη στο δρομολόγιο οδηγεί σε μη εφικτή λύση, κάνε πίσω και εξερεύνησε άλλες δυνατότητες ανάθεσης του πελάτη, λαμβάνοντας υπόψη τους περιορισμούς και τις προδιαγραφές του προβλήματος.

Βήμα 7: Επανέλαβε τα βήματα 3-6 μέχρι να λυθεί το πρόβλημα δρομολόγησης ή λύση του προβλήματος να κριθεί ανέφικτη, με βάση τους δεδομένους περιορισμούς και προδιαγραφές του προβλήματος. [17]

Κεφάλαιο 4. Metaheuristics Algorithms

Τα πρώτα metaheuristics σχεδιάστηκαν στην δεκαετία του 1960 και αποτελούν ένα αλγοριθμικό πλαίσιο υψηλού επιπέδου που παρέχει ένα σύνολο κατευθυντήριων γραμμών ή στρατηγικών για την ανάπτυξη ευρετικών αλγορίθμων βελτιστοποίησης. Έχουν σχεδιαστεί για να βρίσκουν καλές λύσεις σε πολύπλοκα προβλήματα, σε εύλογο χρονικό διάστημα, ειδικά όταν οι παραδοσιακές μέθοδοι είναι μη πρακτικές ή αναποτελεσματικές.

Αξίζει να σημειώσουμε ότι τα metaheuristics συχνά βασίζονται σε πολύ απλές ιδέες, ενώ την αποτελεσματικότητα τους κρίνουν σε μεγάλο βαθμό η κατανόηση του προβλήματος και η ποιότητα του κώδικα. Επίσης, πολλά metaheuristics είναι αποτελεσματικά για ορισμένα προβλήματα ενώ για άλλου είδους προβλήματα δεν παράγουν καλές λύσεις. [18]

4.1 Tabu Search

Η Tabu Search (TS) είναι μια μεταευρετική μέθοδος αναζήτησης που δημιουργήθηκε από τον Αμερικάνο επιστήμονα υπολογιστών και ειδικό στην επιχειρησιακή έρευνα, Dr Fred Glover, το 1986 ενώ επισημοποιήθηκε 1989. Βασίζεται σε μια ισορροπία μεταξύ της εξερεύνησης νέων λύσεων και της εκμετάλλευσης υποσχόμενων λύσεων για την επαναληπτική βελτίωση της ποιότητας των λύσεων, αποφεύγοντας ταυτόχρονα να κολλάει σε τοπικό βέλτιστο.

Τα βήματα της μεθόδου είναι τα εξής :
Βήμα 1: Δημιούργησε μια αρχική λύση για το πρόβλημα βελτιστοποίησης (πιθανός επιλεγμένης τυχαία) , χρησιμοποιώντας ένα ευρετικό αλγόριθμο ή οποιαδήποτε άλλη κατάλληλη μέθοδο και όρισε αυτήν την αρχική λύση ως την καλύτερη μέχρι στιγμής.

Βήμα 2: Δημιούργησε μια λίστα tabu για να παρακολουθείς τις λύσεις ή τα στοιχεία των λύσεων που επισκέφτηκες πρόσφατα, αρχικοποιώντας την με την αρχική λύση του βήματος 1.

Βήμα 3: Καθόρισε μια συνθήκη που θα διακόπτει την αναζήτηση βέλτιστης λύσης , όπως τον μέγιστο αριθμό επαναλήψεων, ένα χρονικό όριο ή ένα οποιαδήποτε κριτήριο τερματισμού με βάση τη φύση του προβλήματος ή τους διαθέσιμου υπολογιστικούς πόρους.

Βήμα 4: Υπολόγισε την τιμή της αντικειμενικής συνάρτησης για την αρχική λύση.

Βήμα 5: Εξερεύνησε γειτονικές λύσεις εφαρμόζοντας κινήσεις ή τροποποιήσεις στην τρέχουσα λύση. Αυτές οι κινήσεις μπορεί να περιλαμβάνουν εναλλαγή στοιχείων, αλλαγή τιμών ή άλλες σχετικές λειτουργίες ανάλογα με τον τύπο του προβλήματος.(Στο δικό μας πρόβλημα μια εναλλαγή λύσεων μπορεί να είναι η εναλλαγή της σειράς που επισκέπτονται δυο ή περισσότεροι πελάτες).

Βήμα 6: Υπολόγισε τις τιμές της αντικειμενικής συνάρτησης για κάθε νέα λύση.

Βήμα 7: Έλεγξε εάν κάποια από τις γειτονικές λύσεις ικανοποιεί τα κριτήρια αναρρόφησης, τα οποία επιτρέπουν την αποδοχή ορισμένων κινήσεων tabu εάν οδηγούν σε σημαντικά καλύτερες λύσεις.

Βήμα 8: Επέλεξε την καλύτερη από τις γειτονικές λύσεις με βάση την τιμή της αντικειμενικής συνάρτησης και πρόσθεσε την νέα καλύτερη-τρέχουσα λύση.

Βήμα 9: Συμπεριέλαβε την νέα λύση στην λίστα tabu, διασφαλίζοντας ότι δεν θα επαναληφθεί στο εγγύς μέλλον.

Βήμα 10: Επανέλαβε τα βήματα 5-9 μέχρι να ικανοποιηθεί η συνθήκη τερματισμού.

Βήμα 11: Επέστρεψε την καλύτερη λύση που βρέθηκε καθ' όλη την εκτέλεση του αλγορίθμου. [19]

4.2 Simulated Annealing

Η Προσομοίωση Ανόπτησης (SA) είναι ένας μεταευρετικός αλγόριθμος βελτιστοποίησης που αναπτύχτηκε από τους Dr. Scott Kirkpatrick, Dr. Daniel Gelatt, and Dr. Mario Vecchi για την επίλυση του προβλήματος περιπλανώμενου πωλητή το 1983. Οι επιστήμονες εμπνεύστηκαν την προσομοίωση ανόπτησης από την διαδικασία ανόπτησης που χρησιμοποιείται στη μεταλλουργία, όπου ένα μέταλλο θερμαίνεται γρήγορα σε υψηλή θερμοκρασία και ψύχεται αργά ώστε να επιτευχθεί η καλύτερη επεξεργασία του. Ομοίως και στην προσομοίωση ανόπτησης μια διαδικασία αναζήτησης ξεκινά με μια κατάσταση υψηλής θερμοκρασίας (μια αρχική λύση) και μειώνει σταδιακά τη θερμοκρασία (μια παράμετρος ελέγχου) μέχρι να σε μια κατάσταση ελάχιστης ενέργειας (την βέλτιστη λύση).

Η μέθοδος προσομοίωση ανόπτησης είναι απλή και μπορεί να εφαρμοστεί με τα εξής βήματα.

Βήμα 1: Δημιούργησε μια αρχική λύση για το πρόβλημα βελτιστοποίησης (πιθανός επιλεγμένης τυχαία) , χρησιμοποιώντας ένα ευρετικό αλγόριθμο ή οποιαδήποτε άλλη κατάλληλη μέθοδο.

Βήμα 2: Δημιούργησε ένα πρόγραμμα ψύξης που μειώνει σταδιακά την παράμετρο "θερμοκρασία". Η θερμοκρασία ελέγχει την πιθανότητα αποδοχής χειρότερων λύσεων.

Βήμα 3: Δημιούργησε μια γειτονική λύση κάνοντας μια μικρή αλλαγή στην τρέχουσα λύση. Στην περίπτωση του TSP η αλλαγή μπορεί να είναι η εναλλαγή δυο πελατών στο δρομολόγιο της τρέχουσας λύσης.

Βήμα 4: Υπολόγισε την τιμή της αντικειμενικής συνάρτησης για τη νέα λύση.

Βήμα 5: Υπολόγισε τη μεταβολή της τιμής της αντικειμενικής συνάρτησης ΔΕ μεταξύ της νέας λύσης και της τρέχουσας λύσης. Πιο συγκεκριμένα $\Delta E = E(x_{new}) - E(x)$.

Εάν η ΔΕ είναι αρνητική, σημαίνει ότι η νέα λύση είναι καλύτερη και αποδεχόμαστε την νέα λύση ως τρέχουσα λύση.

Εάν η ΔΕ είναι θετική, αποδεχόμαστε τη νέα λύση με μια πιθανότητα που καθορίζεται από τον τύπο πιθανότητας αποδοχής, η οποία εξαρτάται τόσο από τη ΔΕ όσο και από την τρέχουσα θερμοκρασία. Το κλασικό κριτήριο αποδοχής στην μέθοδο προσομοίωσης ανόπτησης προέρχεται από τη στατιστική μηχανική και βασίζεται στην κατανομή πιθανοτήτων Boltzmann.

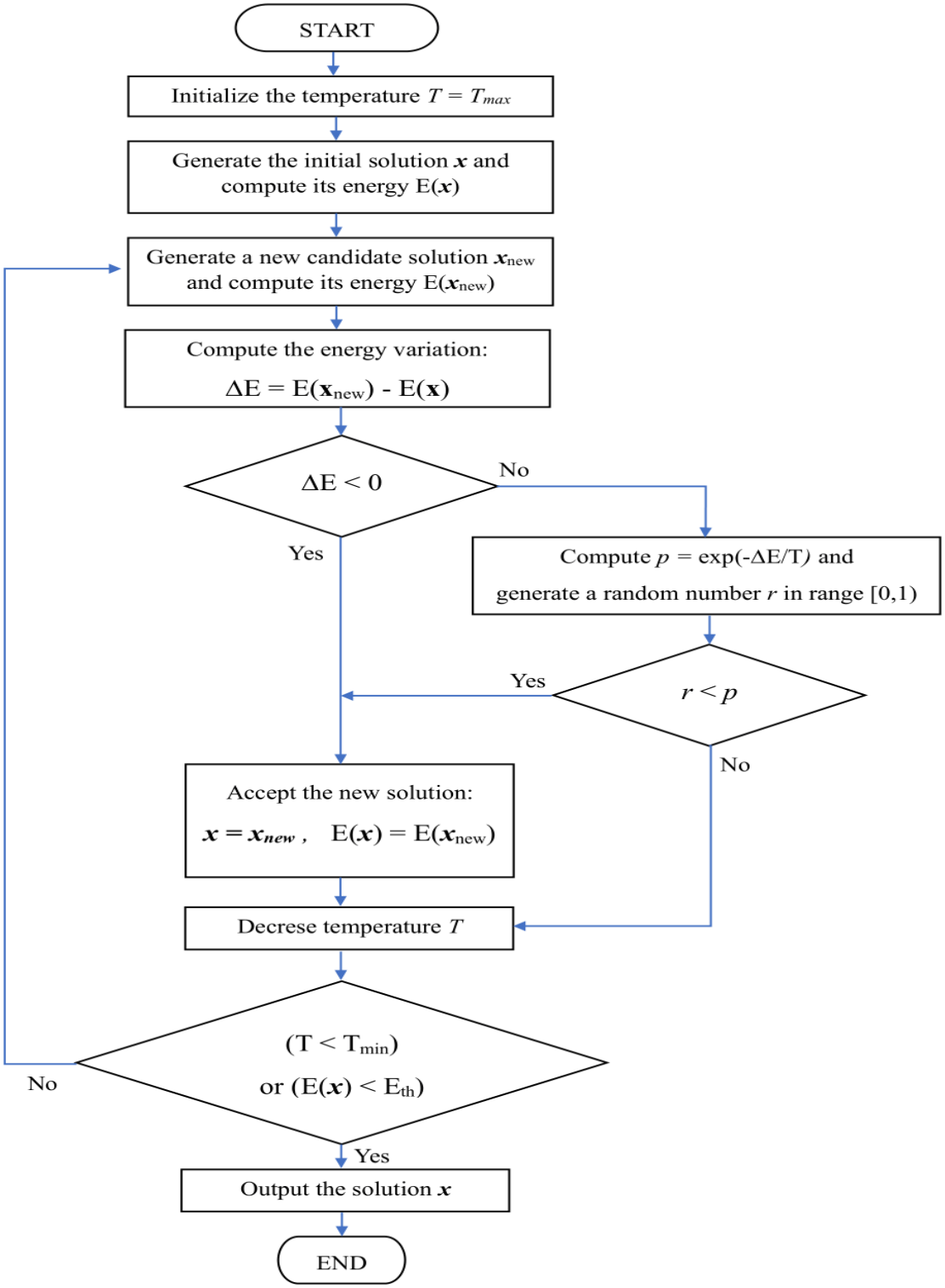
Ένα σύστημα σε θερμική ισορροπία σε θερμοκρασία T μπορεί να βρεθεί σε κατάσταση με ενέργεια E με πιθανότητα ανάλογη του $Prob(E) = \exp(-E/kT)$ όπου k η σταθερά Boltzmann.

Έτσι, μια υποψήφια λύση με υψηλότερη ενέργεια γίνεται αποδεκτή όταν $p = \exp(-\Delta E/T)$ για k=1 είναι μεγαλύτερη από το r, όπου r ένας τυχαίος αριθμός στο διάστημα [0,1).

Βήμα 6: Μείωσε τη θερμοκρασία σύμφωνα με το επιλεγμένο πρόγραμμα ψύξης, το οποίο περιλαμβάνει συνήθως τον πολλαπλασιασμό της τρέχουσας θερμοκρασίας με έναν παράγοντα ψύξης, όπως το 0,95, για τη σταδιακή μείωση της.

Βήμα 6: Επανέλαβε τα βήματα 3-5 μέχρι να ικανοποιηθεί το κριτήριο διακοπής. Το κριτήριο διακοπής μπορεί μα είναι ένας προκαθορισμένος αριθμός επαναλήψεων, όταν η θερμοκρασία έχει φτάσει σε ένα ορισμένο επίπεδο Tmin ή ενέργεια της τρέχουσας λύσης βρίσκεται σε χαμηλότερο επίπεδο από ένα σταθερό όριο Eth.

Βήμα 7: Επέστρεψε την καλύτερη λύση που βρέθηκε κατά την εκτέλεση του αλγορίθμου. [20]



Εικόνα 4.2. Βήματα αλγορίθμου Προσομοίωσης Ανόπτησης- Simulated Annealing.

https://www.baeldung.com/cs/simulated-annealing

4.3 Greedy Descent

Ο αλγόριθμος Greedy Descent είναι μια ευρετική μέθοδος βελτιστοποίησης που χρησιμοποιείται στην επιστήμη των υπολογιστών και στα μαθηματικά για την επίλυση διαφόρων προβλημάτων βελτιστοποίησης. Οι άπληστοι αλγόριθμοι εφαρμόζουν τη στρατηγική της καλύτερης δυνατής επιλογής σε κάθε βήμα, με βάση τους περιορισμούς σε αυτό το βήμα, στοχεύοντας στην άμεση βελτίωση, χωρίς να εξετάζουν πως οι επιλογές σε κάθε βήμα μπορούν να επηρεάσουν τη συνολική λύση του προβλήματος. Για τον λόγο αυτό, τις περισσότερες φορές οι άπληστοι αλγόριθμοι αποτυγχάνουν να παράγουν βέλτιστες λύσεις, ενώ ακόμη μπορεί και να παράγουν τη μοναδική χειρότερη δυνατή λύση του προβλήματος.

Οι άπληστοι αλγόριθμοι είναι απλοί στην εφαρμογή τους και ακολουθούν τα εξής βήματα:

Βήμα 1: Ξεκίνα με μια κενή λύση ή μια αρχική διαμόρφωση.

Βήμα 2: Επέλεξε έναν κόμβο-πελάτη ως αρχικό.

Βήμα 3: Σύνδεσε τον αρχικό κόμβο-πελάτη με τον κόμβο-πελάτη που έχει το φθηνότερο κόστος στο δρομολόγιο, χωρίς να παραβιάζεις τους περιορισμούς του προβλήματος.

Βήμα 4: Ενημέρωσε την τρέχουσα λύση με βάση την επιλεγμένη επιλογή και συνέχισε να προσθέτεις κόμβου-πελάτες στο δρομολόγιο μέχρι να επισκεφτείς όλους τους κόμβους-πελάτες ακριβώς μια φορά.

Βήμα 5: Όταν συμπεριλάβεις όλους τους κόμβους-πελάτες στο δρομολόγιο, αυτή είναι και η λύση του προβλήματος. [21]

4.4 Guided Local Search

Ο αλγόριθμος Guided Local Search έχει τη ρίζα του σε μία αρχιτεκτονική νευρωνικού δικτύου που ονομάζεται GENET και αναπτύχθηκε από τους Chang Wang, Edward Tsang και Andrew Davenport. Πρόκειται για μια στρατηγική υψηλού

επιπέδου, με τη χρήση ποινών, ώστε να βοηθήσει τους αλγόριθμους τοπικής αναζήτησης να ξεφύγουν από τοπικά βέλτιστα.

Για να εφαρμόσουμε τον Guided Local Search πρέπει να ορίσουμε τα χαρακτηριστικά λύσης για το δεδομένο πρόβλημα. Κάθε χαρακτηριστικό f_i συνδέεται με ένα κόστος c_i και μια ποινή p_i. Τα χαρακτηριστικά και το κόστος τις περισσότερες φορές πηγάζουν από την αντικειμενική συνάρτηση. Για παράδειγμα, στο πρόβλημα δρομολόγησης οχημάτων, ένα χαρακτηριστικό θα μπορούσε να είναι αν το όχημα Κ από τον πελάτη Α επισκεφθεί αμέσως τον πελάτη Β. Ως κόστος μπορεί να οριστεί η απόσταση μεταξύ του πελάτη Α και Β ή ο χρόνος που απαιτείται για την μετάβαση του οχήματος Κ από τον πελάτη Α στον πελάτη Β. Οι ποινές αρχικοποιούνται στην τιμή 0 και αυξάνονται κατά 1 κάθε φορά που η τοπική αναζήτηση φτάνει σε ένα τοπικό βέλτιστο.

Για να υλοποιήσουμε την διαδικασία, θα πρέπει για όλα τα χαρακτηριστικά i να ορίσουμε μια λειτουργία δείκτη I_i , το οποίο θα μας υποδεικνύει εάν ένα χαρακτηριστικό υπάρχει στην τρέχουσα λύση ή όχι. Ο δείκτης I_i είναι παίρνει την τιμή 1 όταν η λύση x έχει χαρακτηριστικό το i, αλλιώς παίρνει την τιμή 0.

$$I_i(x) = \begin{cases} 1 & \text{if solution } x \text{ has feature } i \\ 0 & \text{otherwise.} \end{cases}$$

https://acrogenesis.com/or-tools/documentation/user_manual/manual/metaheuristics/GLS.html

Ο Guided Local Search χρησιμοποιεί μια συνάρτηση επαυξημένου κόστους, την οποία αυξάνει προσθέτοντας ποινές σε επιλεγμένα χαρακτηριστικά, κάθε φορά που ο αλγόριθμος τοπικής αναζήτησης εγκλωβίζεται σε ένα τοπικό βέλτιστο. Σκοπός είναι να γίνει το τοπικό βέλτιστο πιο δαπανηρό από τον υπόλοιπο χώρο αναζήτησης, όπου τα χαρακτηριστικά αυτά δεν υπάρχουν.

$$g(x) = f(x) + \lambda a \sum_{1 \le i \le m} I_i(x) p_i$$

Εικόνα 4.4.1:Συνάρτηση επαυξημένου κόστους.

https://en.wikipedia.org/wiki/Guided_local_search

Το λ είναι παράμετρος του Guided Local Search, ονομάζεται παράγοντας ποινής και χρησιμοποιείται για τον συντονισμό αναζήτησης λύσης. Μια υψηλή τιμή για το λ, οδηγεί τον αλγόριθμο σε εντελώς διαφορετικές λύσεις (διαφοροποίηση), ενώ μια χαμηλή τιμή του λ οδηγεί τον αλγόριθμο στην αναζήτηση παρόμοιων λύσεων (εντατικοποίηση). Το **α** χρησιμοποιείται για να εξισορροπηθεί το τμήμα ποινής της αντικειμενικής συνάρτησης σε σχέση με τις αλλαγές στην αντικειμενική συνάρτηση. Για να υπολογίσουμε το **α**, θα πρέπει ο αλγόριθμος τοπικής αναζήτησης να βρει το πρώτο τοπικό βέλτιστο. Στην συνέχεια υπολογίζουμε το **α**, διαιρώντας την μέση αλλαγή της αντικειμενικής συνάρτησης μέχρι το πρώτο τοπικό βέλτιστο με τον αριθμό των δυνατοτήτων του Guided Local Search στο παρόν πρόβλημα.

Η πρωτοτυπία και η αποτελεσματικότητα του Guided Local Search, είναι ότι ένα χαρακτηριστικό i τιμωρείται όταν η χρησιμότητα του είναι αρκετά μεγάλη. Σκοπός είναι να τιμωρούνται τα δαπανηρά χαρακτηριστικά, χωρίς όμως να τιμωρούνται υπερβολικά όταν εμφανίζονται συχνά. Για να το επιτύχουμε αυτό θα χρησιμοποιήσουμε την εξής βοηθητική συνάρτηση:

$$\mathrm{util}(x, i) = I_i(x)\frac{c_i(x)}{1 + p_i}.$$

Εικόνα 4.4.2: Τύπος υπολογισμού χρησιμότητας τιμωρίας ενός χαρακτηριστικού i.

https://en.wikipedia.org/wiki/Guided_local_search

Με βάση τον παραπάνω τύπο όταν ένα χαρακτηριστικό i δεν υπάρχει σε μια λύση x, τότε η χρησιμότητα τιμωρίας του για αυτήν την λύση είναι 0, αφού $I_i(x)=0$. Διαφορετικά, η χρησιμότητα τιμωρίας ενός χαρακτηριστικού είναι ανάλογη του κόστους c_i. Όσο μεγαλύτερο είναι το κόστος c_i τόσο μεγαλύτερη είναι χρησιμότητα τιμωρίας του χαρακτηριστικού αυτού. Αντιθέτως, όσες περισσότερες φορές έχει τιμωρηθεί ένα χαρακτηριστικό i (δηλαδή όσο μεγαλύτερο p_i), τόσο χαμηλότερη είναι η χρησιμότητα τιμωρίας του.

Βήμα 1: Ξεκίνα με μια αρχική λύση, η οποία μπορεί να δημιουργηθεί τυχαία ή μέσω μια ευρετικής μεθόδου.

Βήμα 2: Όρισε όλες τις ποινές p_i ίσων με το μηδέν.

Βήμα 3: Καθόρισε την συνάρτηση επαυξημένου στόχου.

Βήμα 4: Εφάρμοσε μια διαδικασία τοπικής αναζήτησης ώστε να εξερευνηθεί η γειτονία της υπάρχουσας λύσης.

Βήμα 5: Υπολόγισε την χρησιμότητα των χαρακτηριστικών.

Βήμα 6: Κάνε χρήση ποινών στα χαρακτηριστικά με την μεγαλύτερη χρησιμότητα.

Βήμα 7: Επανέλαβε τα βήματα 4-8 μέχρι το κριτήριο τερματισμού να ικανοποιηθεί. Κριτήριο τερματισμού μπορεί να είναι ένα όριο στον αριθμό κινήσεων που εκτελούνται από τον αλγόριθμο ή ένας μέγιστος χρόνος που έχει στην διάθεση του ο αλγόριθμος για την αναζήτηση της λύσης.

Βήμα 8: Αφού ικανοποιηθεί η συνθήκη τερματισμού, επέστρεψε στον χρήστη την καλύτερη λύση που βρέθηκε. [22]

4.5 Generic Tabu Search

Παρόμοια είναι τα βήματα του metaheuristic Generic Tabu Search με το metaheuristic Tabu Search, με την διαφορά ότι ο Generic Tabu Search χρησιμοποιεί την αναζήτηση tabu στην αντικειμενική τιμή της λύσης για να καταφέρνει να ξεφύγει από τοπικά βέλτιστα. Πιο συγκεκριμένα τα βήματα του αλγορίθμου είναι τα εξής:

Βήμα 1: Ξεκίνα με μια τυχαία αρχική λύση με τη βοήθεια ενός ευρετικού αλγορίθμου.

Βήμα 2: Προσδιόρισε μια αντικειμενική συνάρτηση που να αξιολογεί την ποιότητα μιας λύσης.

Βήμα 3: Αναζήτησε νέες γειτονικές λύσεις εφαρμόζοντας κινήσεις στην τρέχουσα λύση. Μια κίνηση μπορεί να είναι για παράδειγμα η εναλλαγή δυο πελατών στην σειρά επίσκεψης στο δρομολόγιο ή η τοποθέτηση ενός πελάτη στο δρομολόγιο άλλου οχήματος.

Βήμα 4: Αξιολόγησε όλες τις γειτονικές λύσεις χρησιμοποιώντας τη συνάρτηση στόχου και επέλεξε την καλύτερη.

Βήμα 5: Δημιούργησε μια λίστα tabu όπου θα αποθηκεύονται οι πρόσφατες λύσεις , ώστε να απαγορευτούν κινήσεις που οδηγούν σε ήδη υπάρχον λύσεις.

Βήμα 6: Εισήγαγε κριτήρια αναρρόφησης ώστε να επιτραπούν κινήσεις που θεωρούνται υποσχόμενες, ακόμα και αυτές που βρίσκονται στην λίστα tabu.

Βήμα 7: Λαμβάνοντας υπόψη τη λίστα tabu, ενημέρωσε την τρέχουσα λύση με την καλύτερη που βρέθηκε στη γειτονιά.

Βήμα 8: Επανέλαβε τα βήματα μέχρι να ικανοποιηθεί το κριτήριο τερματισμού, όπως μέγιστος αριθμός επαναλήψεων ή ένας μέγιστος χρόνος που έχει στην διάθεση του ο αλγόριθμος για την αναζήτηση της λύσης.

Βήμα 9: Αφού ικανοποιηθεί το κριτήριο τερματισμού επέστρεψε από την λίστα tabu την καλύτερη λύση. [23]

Κεφάλαιο 5. Εργαλεία που χρησιμοποιήθηκαν.

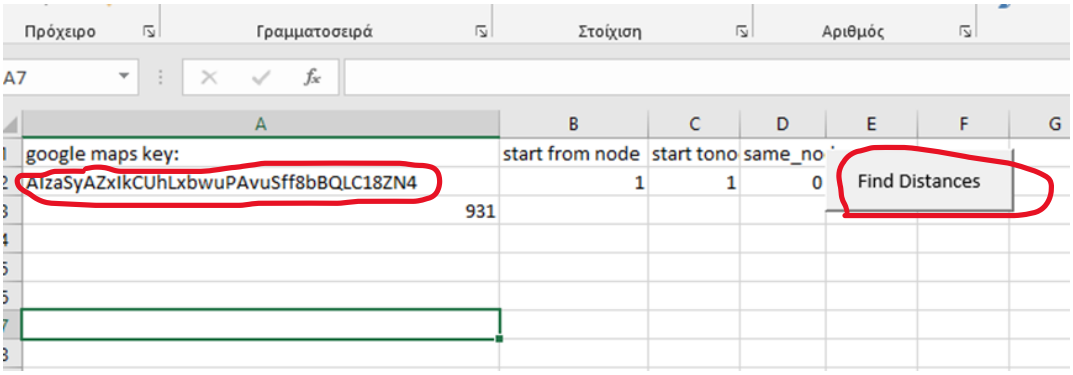
Η υλοποίηση του κώδικα έγινε σε προσωπικό υπολογιστή με τα εξής χαρακτηριστικά:

- Επεξεργαστής: Intel(R) Core(TM) i5-3340 CPU
- Εγκατεστημένη μνήμη: 4 GB
- Λογισμικό: Windows 10 Pro 64 bit
- Χρησιμοποιήθηκε Excel Office 16
- Microsoft Visual Studio 2022 C++

5.1 Excel

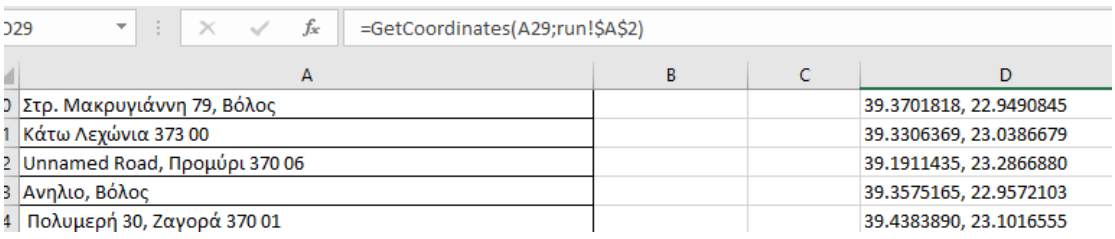
Η συλλογή και η διαχείριση των δεδομένων είναι μια δύσκολη και απαιτητική εργασία. Έτσι δημιουργήσαμε ένα φύλλο Excel όπου ταξινομήσαμε και επεξεργαστήκαμε ταυτόχρονα τα δεδομένα.

Η εύρεση των αποστάσεων θα ήταν μια αρκετά επίπονη διαδικασία αν δεν χρησιμοποιούσαμε τα απαραίτητα εργαλεία. Πιο συγκεκριμένα, δημιουργήθηκε ένα κλειδί μέσω της Google το οποίο με την εισαγωγή της διεύθυνσης ή το όνομα της επιχείρησης του πελάτη ή αποθήκης εντοπίζει τις γεωγραφικές συντεταγμένες τους. Στην συνέχεια προγραμματίσαμε στο Excel με του που δίνουμε μια διεύθυνση ή ένα όνομα μιας επιχείρησης να υπολογίζει μόνο του τις αποστάσεις μεταξύ των πελατών αλλά και τις αποστάσεις πελατών και αποθήκης. Στην συγκεκριμένη περίπτωση, υπολόγισε όλες τις χωρικές και χρονικές αποστάσεις 30 πελατών και 1 αποθήκης ταυτόχρονα μόλις σε πέντε λεπτά.



Εικόνα 5.1.1 Πρόγραμμα υπολογισμού χρονικών και χωρικών αποστάσεων στο Excel.

Στην εικόνα 5.1.1 είναι κυκλωμένα με κόκκινο χρώμα το κλειδί που δημιουργήσαμε στην Google για να εντοπίζει τις γεωγραφικές συντεταγμένες των πελατών και της επιχείρησης καθώς και το κουμπί που δίνει την εντολή για τον υπολογισμό των χρονικών και χωρικών αποστάσεων. Στην εικόνα 5.2 βλέπουμε ενδεικτικά πως το Excel εντοπίζει τις γεωγραφικές συντεταγμένες των πελατών.



Εικόνα 5.1.2 Εύρεση γεωγραφικών συντεταγμένων(x,y)

Στην εικόνα 5.1.3 παρουσιάζονται ενδεικτικά τα αποτελέσματα εύρεσης των χρονικών και χωρικών αποστάσεων.

	A	B	C	D	E
	i->j	i	j	Απόσταση σε μέτρα	Χρόνος σε λεπτά
	0->1	0	1	7487	13
	0->2	0	2	8426	17
	0->3	0	3	11942	23
	0->4	0	4	15254	25

Εικόνα 5.1.3 Χρονικές και χωρικές αποστάσεις μεταξύ των σημείων.

	A	B	C	F
1	veh_id	veh_capacity	veh_dscr	
2	0	14	V-14	
3	1	17	V-17-A	
4	2	17	V-17-B	
5	3	17	V-17-C	
6	4	4	V-4-A	
7	5	4	V-A-B	
8				
9				

Εικόνα 5.1.4 Εισαγωγή δεδομένων για τα οχήματα.

Στην καρτέλα Vehicles που υπάρχει στο Excel που δημιουργήσαμε εισάγουμε τα δεδομένα που αφορούν τα οχήματα. Πιο συγκεκριμένα, στην στήλη A του Excel βάζουμε πόσα φορτηγά έχουμε στην διάθεση μας ξεκινώντας να μετράμε από το 0, δηλαδή το 0 είναι το 1, λόγω της γλώσσας προγραμματισμού που χρησιμοποιήσαμε στον κώδικα. Στην στήλη B του Excel βάζουμε την μέγιστη χωρητικότητα του οχήματος σε ευρωπαλέτες διαστάσεων 80 × 120 εκ.

	A	B	C	D	E
1	max_waiting_time	max_time_per_vehicle	service_time	exec_time_limit	
2	30	1440	0	4	
3					
4					
5					

Εικόνα 5.1.5 Καρτέλα Prefs.

Στην καρτέλα Prefs που υπάρχει στο Excel που δημιουργήσαμε στην στήλη A βάζουμε τον χρόνο (σε λεπτά) που μπορεί ένα όχημα να περιμένει σε έναν κόμβο άμα το χρονοπαράθυρο δεν το βολεύει. Στην στήλη B ορίζουμε τα λεπτά που είναι διαθέσιμο το όχημα για το δρομολόγιο. Στην στήλη C ορίζουμε ένα μέσο χρόνο εξυπηρέτησης για τον κάθε πελάτη σε λεπτά. Τέλος, στην στήλη D ορίζουμε μέγιστο χρονικό όριο (σε δευτερόλεπτα) που έχει στην διάθεση του ο αλγόριθμος για να λύσει το πρόβλημα.

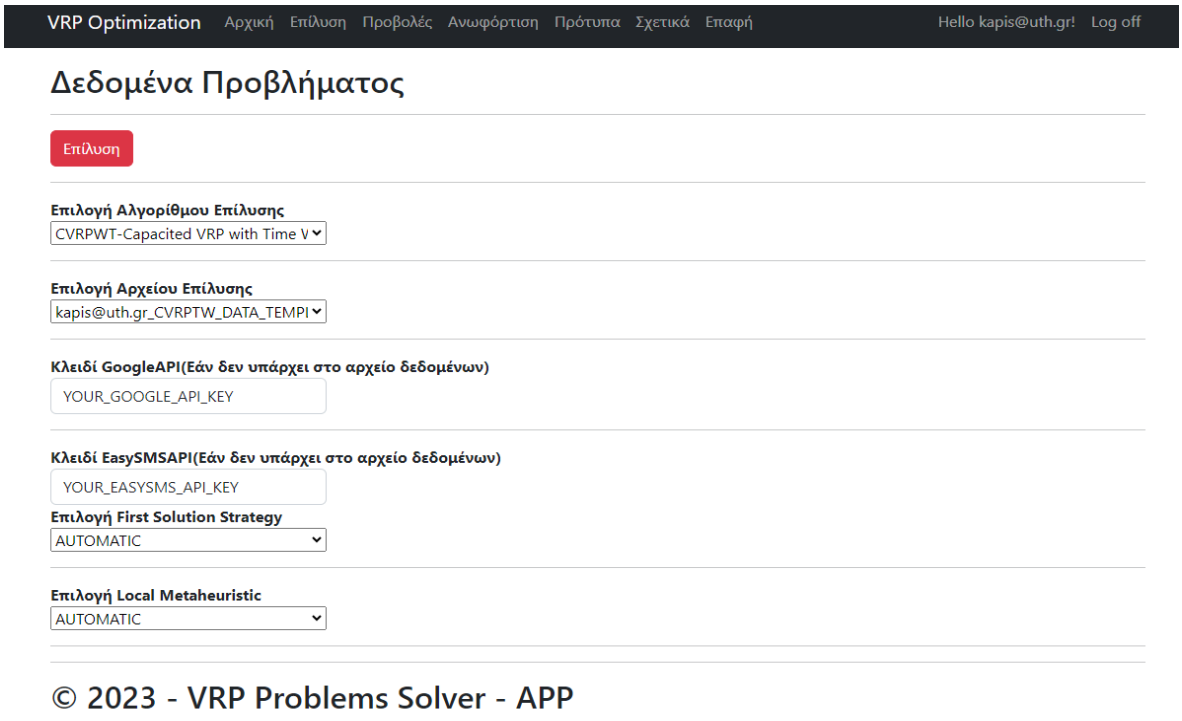
	A	B	C	D	E	F
1	demand_id	node_id	dem_load	dem_et	dem_lt	dem_descr
2	0	0	0	0	5	d00
3	1	1	2	360	540	d01
4	2	2	2	0	120	d02
5	3	3	2	0	420	D03
6	4	4	3	110	420	D04
7	5	5	4	0	420	d05
8	6	6	2	100	420	d06
9	7	7	1	300	420	d07
10	8	8	2	300	420	d08
11	9	9	2	0	180	d09
12	10	10	3	0	540	d10

Εικόνα 5.1.6 Δεδομένα ζήτησης και χρονοπαραθύρων.

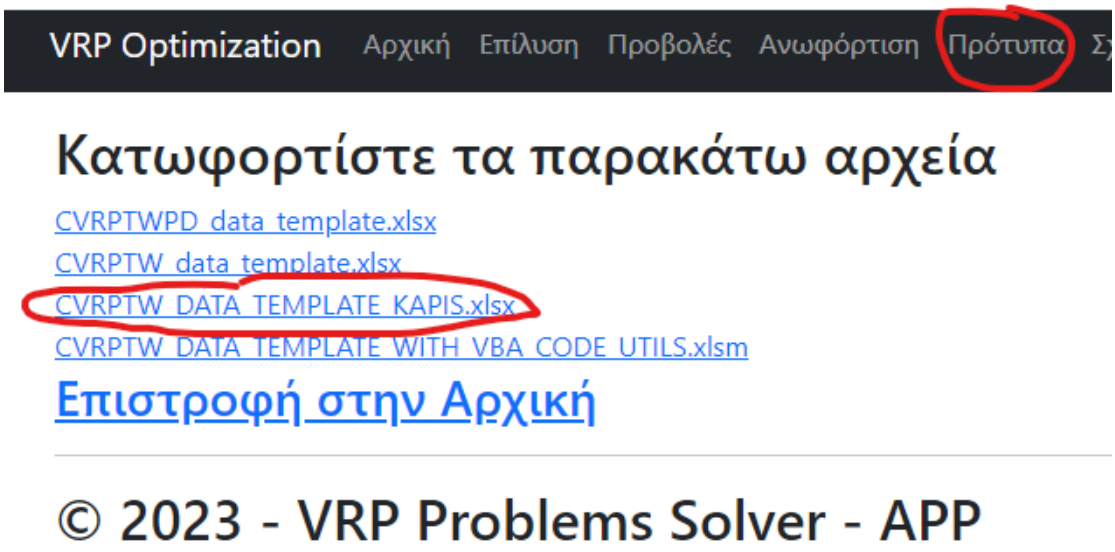
Στην καρτέλα Demands στο Excel που δημιουργήσαμε, στην στήλη C εισάγουμε την ζήτηση του κάθε πελάτη σε ευρωπαλέτες. Στην στήλη D εισάγουμε το κάτω όριο του χρονικού παραθύρου και στην στήλη E το άνω όριο χρονικού παραθύρου που είναι ο πελάτης διαθέσιμος για εξυπηρέτηση.

5.2 Εφαρμογή για την επίλυση του VRP.

Ο δεύτερος επιτηρητής της διπλωματικής μου κ. Αθανάσιος Λόης δημιούργησε μια εφαρμογή όπου ενσωμάτωσε τον κώδικα που δημιουργήσαμε ώστε να γίνει πιο εύχρηστος για όλους.

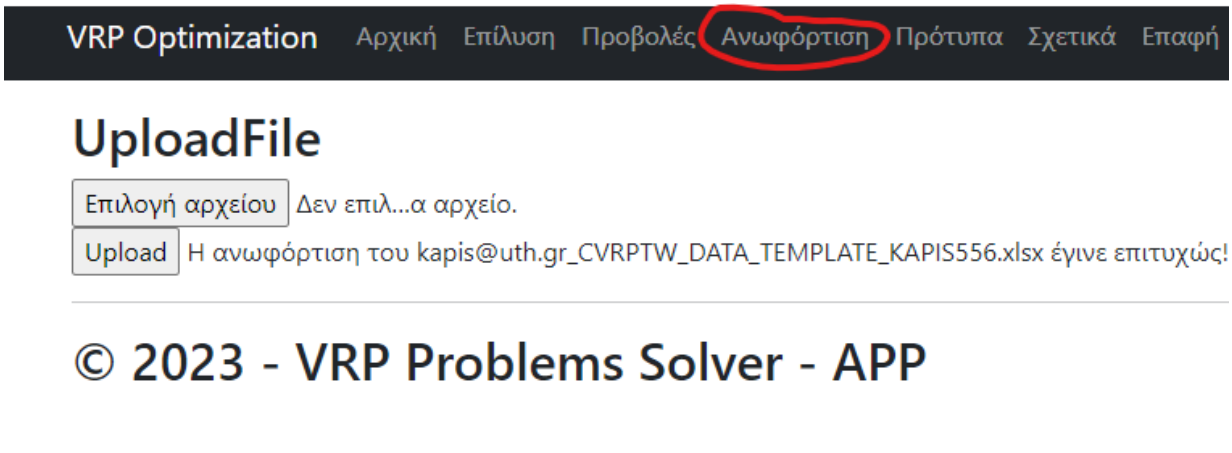


Εικόνα 5.2.1 Menu της εφαρμογής.



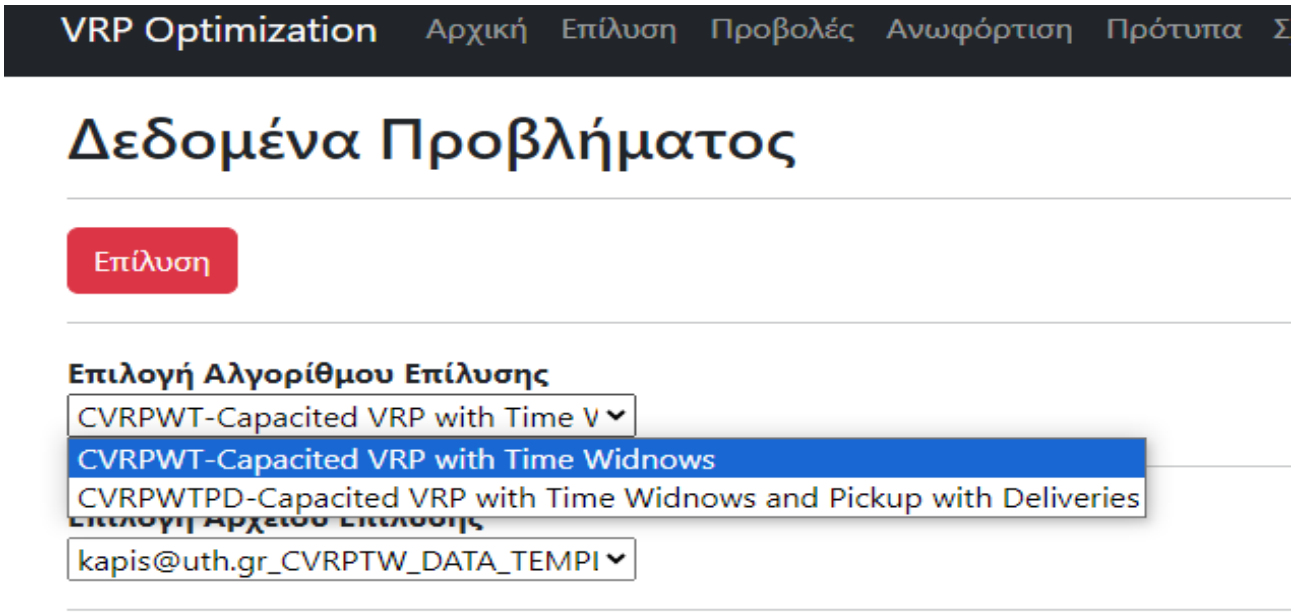
Εικόνα 5.2.2 Εύρεση πρότυπου αρχείου Excel.

Στην επιλογή πρότυπα θα βρούμε το Excel που παρουσιάσαμε προηγούμενος, ώστε ο χρήστης να εισάγει προσεκτικά τα δεδομένα και το Excel να κάνει την απαραίτητη επεξεργασία, για να είναι έτοιμη η εφαρμογή να λύσει το πρόβλημα.



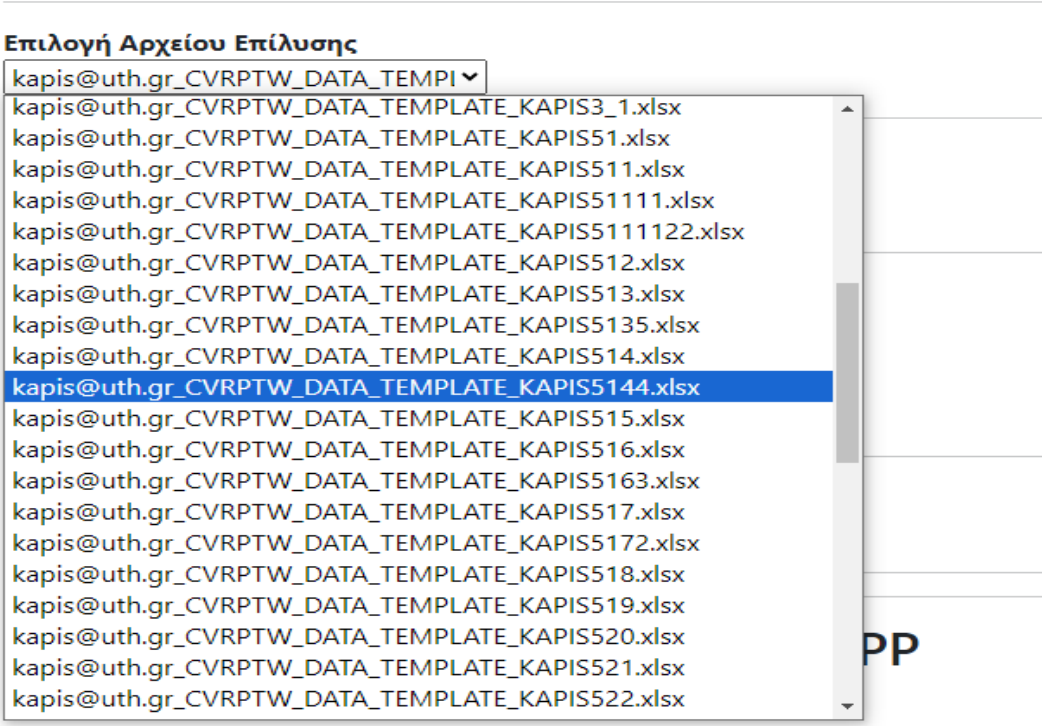
Εικόνα 5.2.3 Upload του αρχείου Excel.

Στην συνέχεια, από το menu επιλέγουμε την επιλογή Ανωφόρτιση κάνουμε Upload το Excel στην εφαρμογή.



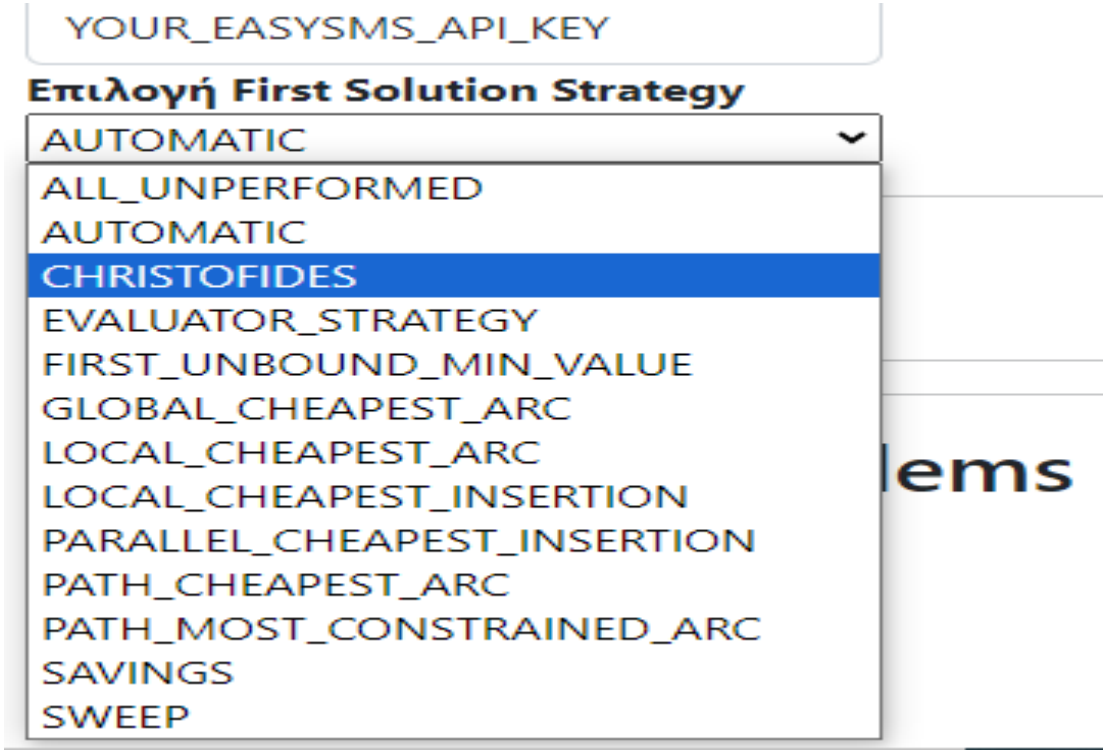
Εικόνα 5.2.4 Επιλογή είδους VRP.

Στην επιλογή αλγορίθμου επίλυσης διαλέγουμε το είδος του VRP που θέλουμε να μας λύσει.



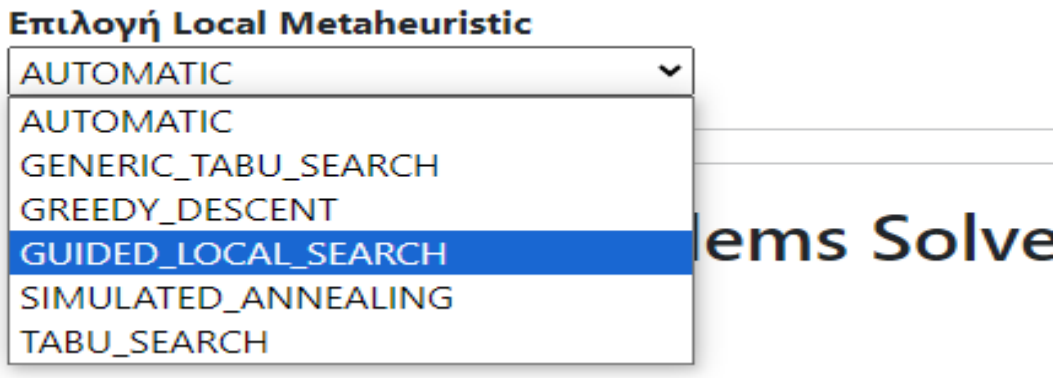
Εικόνα 5.2.5 Επιλογή αρχείου επίλυσης.

Εδώ επιλέγουμε το αρχείο προς επίλυση από την εφαρμογή.



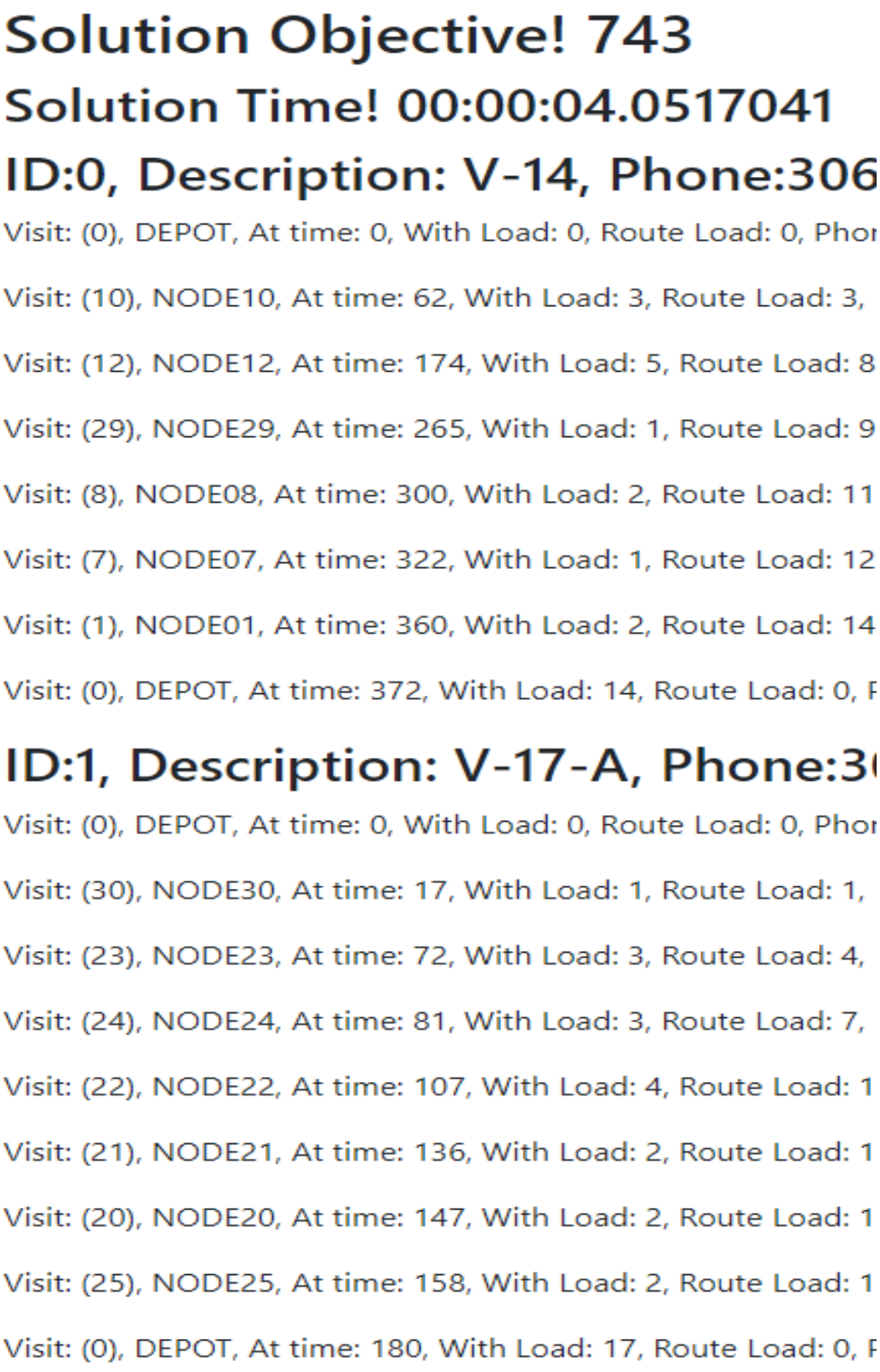
Εικόνα 5.2.6 Επιλογή αλγορίθμου επίλυσης.

Σε αυτό το σημείο επιλέγουμε με ποιον από τους διαθέσιμους αλγόριθμος , θέλουμε η εφαρμογή να προσπαθήσει να λύσει το πρόβλημα.



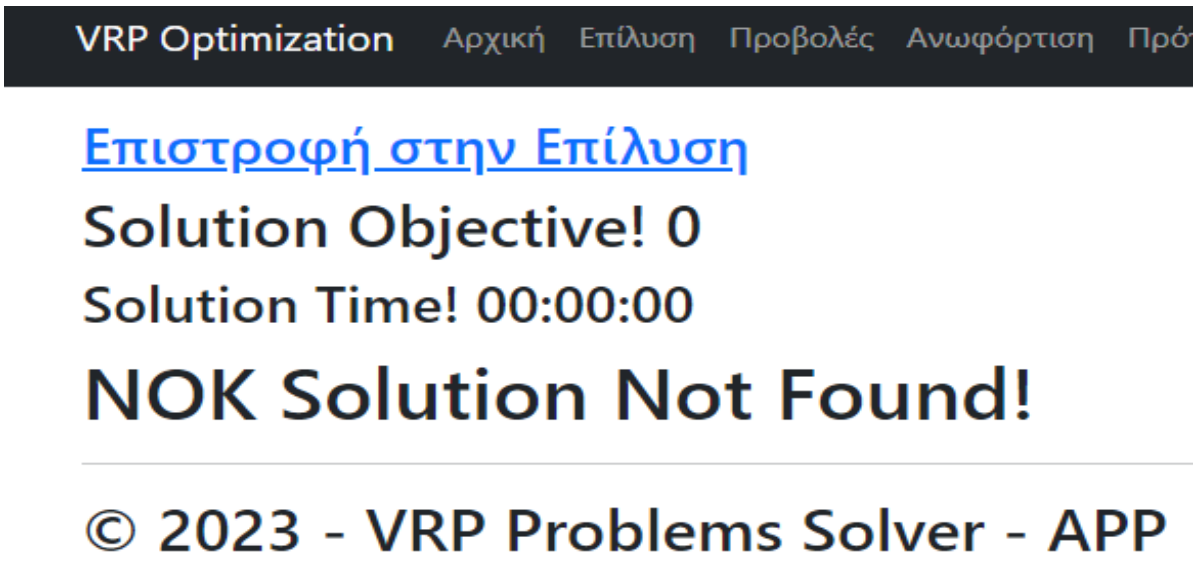
Εικόνα 5.2.6 Επιλογή Metaheuristic.

Τέλος, επιλέγουμε το metaheuristic που πιστεύουμε ότι είναι κατάλληλο για να βοηθήσει τον αλγόριθμο να που επιλέξαμε να δώσει την καλύτερη δυνατή λύση.



Εικόνα 5.2.7. Ενδεικτική λύση του VRP.

Στην εικόνα 5.2.7 βλέπουμε πως είναι τα αποτελέσματα της λύσης που μας δίνει η εφαρμογή για το πρόβλημα δρομολόγησης οχημάτων με χρονοπαράθυρα . Στην αρχή μας δίνει την τιμή της αντικειμενικής συνάρτησης καθώς και τον χρόνο επίλυσης του προβλήματος. Επίσης, μας δίνει με τη σειρά τους πελάτες που επισκέπτεται το κάθε όχημα ξεχωριστά, την χρονική στιγμή που επισκέπτεται τον κάθε πελάτη καθώς και την ποσότητα που ξεφορτώνει σε κάθε πελάτη. Τέλος, μας δίνει για κάθε όχημα τον συνολικό χρόνο ταξιδιού καθώς και την συνολική ποσότητα που φορτώθηκε στο όχημα στην αρχή του δρομολογίου.



Εικόνα 5.2.7. Μήνυμα εφαρμογής όταν δεν βρίσκει λύση

Στην εικόνα 5.2.7. βλέπουμε το μήνυμα που εμφανίζει η εφαρμογή όταν δεν μπορεί να λύσει το πρόβλημα. Αυτό συμβαίνει όταν κάποιος αλγόριθμος δεν μπορεί να λύσει το πρόβλημα ή όταν το πρόβλημα δεν έχει εφικτή λύση.

5.3 OR-TOOLS

Τα Google OR-TOOLS είναι μια δωρεάν σουίτα λογισμικού ανοιχτού κώδικα που αναπτύχθηκε από την Google το 2011 και από τότε έχει εξελιχθεί σε μια ευέλικτη εργαλειοθήκη για την επίλυση προβλημάτων:

- γραμμικού προγραμματισμού (LP)

- προγραμματισμού μικτού ακέραιου αριθμού (MIP)
- προγραμματισμού περιορισμών (CP)
- δρομολόγησης οχημάτων (VRP)
- και πολλών άλλων σχετικών προβλημάτων βελτιστοποίησης

Για κάθε ένα είδους προβλήματος που αναφέραμε προηγουμένους, τα OR-TOOLS διαθέτουν ευέλικτες βιβλιοθήκες, κάθε μια σχεδιασμένη για συγκεκριμένα προβλήματα βελτιστοποίησης.

Ένα από τα πλεονέκτημα των OR-TOOLS είναι ότι διαθέτουν μια ποικιλία γλωσσών προγραμματισμού, όπως:

- C++
- Python
- DotNet
- Java

Το μεγάλο πλεονέκτημα των OR-Tools είναι ότι παρέχουν εκτεταμένη τεκμηρίωση, σεμινάρια και αναλυτικά παραδείγματα για να βοηθήσει τους χρήστες να κατανοήσουν και να εφαρμόσουν αποτελεσματικά τα OR-Tools. Πιο συγκεκριμένα, εξηγεί στον χρήστη αναλυτικά το πρόβλημα με παραδείγματα και στην συνέχεια τον καθοδηγεί βήμα-βήμα, εξηγώντας το κάθε βήμα κάθε φορά για το πως πρέπει να λύσει το εκάστοτε πρόβλημα βελτιστοποίησης. Φυσικά, το πρόβλημα που καλείται ένας χρήστης να λύσει, μπορεί να έχει κάποιες διάφορες με το πρόβλημα του παραδείγματος των OR-TOOLS, αλλά και εδώ υπάρχει λύση. Τα OR-TOOLS μέσω του GitHud έχουν δημιουργήσει μια κοινότητα όπου οι χρήστες αλληλοεπιδρούν μεταξύ τους. Έτσι, όποια δυσκολία και αν αντιμετωπίσει ένας χρήστης μπορεί μέσω της κοινότητας να ζητήσει βοήθεια από τα μέλη της. Επίσης, μέσω της κοινότητας παρέχεται ένα φιλικό περιβάλλον συνεργασίας όπου οι χρήστες μπορούν να αναφέρουν σφάλματα, να βοηθούν ο ένας τον άλλον, ακόμη και να συνεισφέρουν κώδικα και ιδέες για την ανάπτυξη και την βελτίωση των OR-TOOLS.

Αν και τα Google OR-TOOLS έχουν επιτύχει αξιοσημείωτη επιτυχία, οι προσπάθειες για βελτίωση δεν σταματούν ποτέ. Η ομάδα ανάπτυξης εργάζεται καθημερινά για την επεκτασιμότητα ορισμένων λύσεων για προβλήματα μεγάλης κλίμακας. Επίσης, στο μέλλον η σουίτα των OR-TOOLS μπορεί να εμπλουτίσει τις υπηρεσίες της, με αναδυόμενες τεχνολογίες, όπως η μηχανική μάθηση και η τεχνητή νοημοσύνη. Με την ενσωμάτωση αυτών των τεχνολογιών, θα δώσει στους χρήστες νέες δυνατότητες για την επίλυση πολύπλοκων προβλημάτων, τα οποία στο παρελθόν ήταν αδύνατο να λυθούν με τις παραδοσιακές προσεγγίσεις βελτιστοποίησης.

Κεφάλαιο 6. Σύγκριση Αποτελεσμάτων

Σε αυτό το κεφάλαιο θα σχολιάσουμε τα αποτελέσματα από τα πειράματα που πραγματοποιήσαμε. Μελετήσαμε τρεις περιπτώσεις:

1. Διαφορετικό πλήθος φορτηγών
2. Διαφορετικά χρονοπαράθυρα.
3. Διαφορετικός αριθμός πελατών.

Σε αυτό το σημείο αξίζει να αναφέρουμε κάποια γενικά συμπεράσματα που ισχύουν και για τις τρεις περιπτώσεις που μελετήσαμε, ώστε να μην αναγράφονται συνέχεια.

Όταν ένας αλγόριθμος αποτύγχανε στο να δώσει λύση σε μια από τις περιπτώσεις που μελετήσαμε, αυτό γινόταν με οποιοδήποτε metaheuristic και αν δοκιμάζαμε. Παρομοίως, όταν ένας αλγόριθμος επίλυε μια από τις περιπτώσεις που μελετήσαμε, συνέχιζε να επιλύει το πρόβλημα με οποιοδήποτε metaheuristic και αν δοκιμάζαμε, απλά με διαφορετική λύση και σε διαφορετικό χρόνο. Έτσι, τόσο η επιτυχία, όσο και η αποτυχία στην εύρεση εφικτής λύσης στις περιπτώσεις που εξετάσαμε εξαρτιόταν αποκλειστικά και μόνο από τον αλγόριθμο που χρησιμοποιούσαμε, ενώ τα metaheuristic βελτίωναν στην συνέχεια την λύση που μας έδινε ο αλγόριθμος.

Σε όλους τους αλγόριθμους για όλες τις περιπτώσεις που μελετήσαμε, τους δόθηκε περισσότερος χρόνος για την επίλυση των προβλημάτων, παρ' όλα αυτά πάντα μας έδιναν την ίδια λύση-δρομολόγιο. Όταν ένας αλγόριθμος αποτύγχανε στο να δώσει λύση σε μια από τις περιπτώσεις που μελετήσαμε, συνέχιζε να αποτυγχάνει, ακόμη και αν αυξάναμε τον χρόνο επίλυσης σε μη επιτρεπτά όρια (3 ώρες).

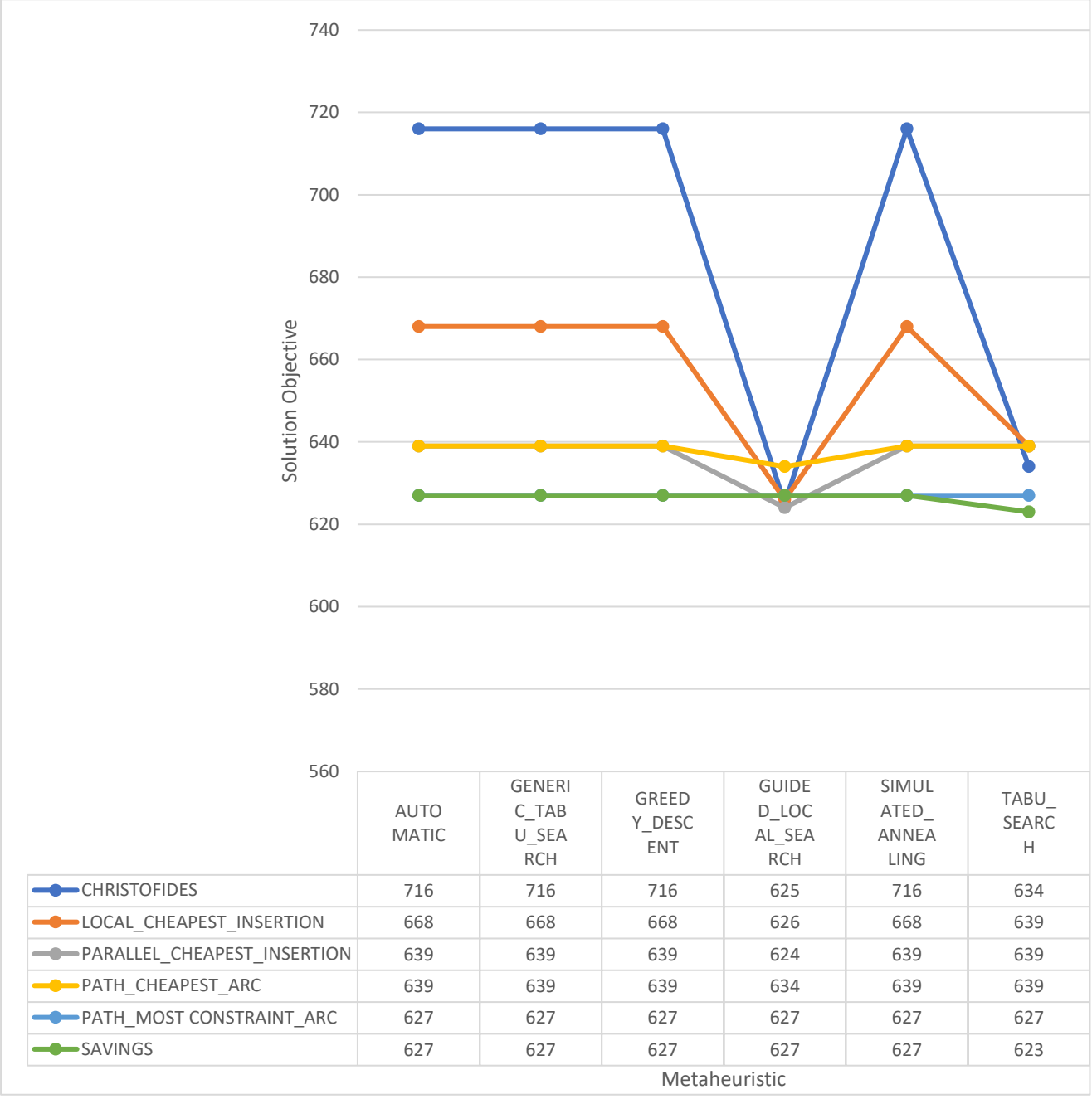
Όταν ο χρήστης καλείται να επιλέξει ποιο metaheuristic θα χρησιμοποιήσει, τα google or-tools δίνουν τη δυνατότητα επιλογής Automatic, όπου το πρόγραμμα επιλέγει μόνο του ποιο metaheuristic θα χρησιμοποιήσει για την λύση του προβλήματος. Στην μελέτη που πραγματοποιήσαμε κάθε φορά που επιλέγαμε την επιλογή Automatic το πρόγραμμα επέλεγε πάντα το metaheuristic Greedy Descent, αφού σε όλες τις περιπτώσεις είχαν την ίδια λύση-δρομολόγιο και τον ίδιο χρόνο επίλυσης. Προφανώς το πρόγραμμα επέλεγε το metaheuristic Greedy Descent, αφού καθ' όλη τη μελέτη που πραγματοποιήσαμε, ήταν αυτό που επίλυε σε όλες τις περιπτώσεις πιο γρήγορα το πρόβλημα, δίνοντας όμως πάντα χειρότερη λύση σε σχέση με τα υπόλοιπα metaheuristic. Τέλος, το Automatic θα παρουσιάζεται μαζί τα υπόλοιπα metaheuristic σε όλα τα σχεδιαγράμματα και όλους τους πίνακες, ως απόδειξη των παραπάνω, χωρίς να γίνουν περαιτέρω αναφορές, αφού ότι ισχύει για το metaheuristic Greedy Descent, ισχύει και για το Automatic.

6.1 Μελέτη περίπτωσης Φορτηγών

Η εταιρία που μας έδωσε τα δεδομένα διαθέτει έναν ετερογενή στόλο 6 φορτηγών. Στο εγγύς μέλλον σκοπεύει να αλλάξει τον στόλο της λόγω παλαιότητας με νέα υπερσύγχρονα και πιο οικονομικά από άποψης κατανάλωσης οχήματα. Έτσι μας ζητήθηκε να κάνουμε μια μελέτη για πόσα φορτηγά και σε ποιες χωρητικότητες μπορούν να καλύψουν την ημερήσια ζήτηση από τους πελάτες της.

6.1.1 Μελέτη περίπτωσης για 2 Φορτηγά

Σε αυτό το πείραμα προσπαθήσαμε να λύσουμε το πρόβλημα με δυο φορτηγά, χωρητικότητας 36 ευρωπαλετών το ένα Τα αποτελέσματα που προέκυψαν από το πείραμα είναι τα εξής:



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	716	716	716	625	716	634
LOCAL_CHEAPEST_INSERTION	668	668	668	626	668	639
PARALLEL_CHEAPEST_INSERTION	639	639	639	624	639	639
PATH_CHEAPEST_ARC	639	639	639	634	639	639
PATH_MOST CONSTRAINT_ARC	627	627	627	627	627	627
SAVINGS	627	627	627	627	627	623

Metaheuristic

Οι αλγόριθμοι Global Cheapest Arc και Local Cheapest Arc απέτυχαν να δώσουν λύση στο συγκεκριμένο πείραμα. Αντιθέτως, οι υπόλοιποι 6 έξι αλγόριθμοι κατάφεραν να λύσουν το πρόβλημα σε μικρό χρονικό διάστημα. Ο αλγόριθμος

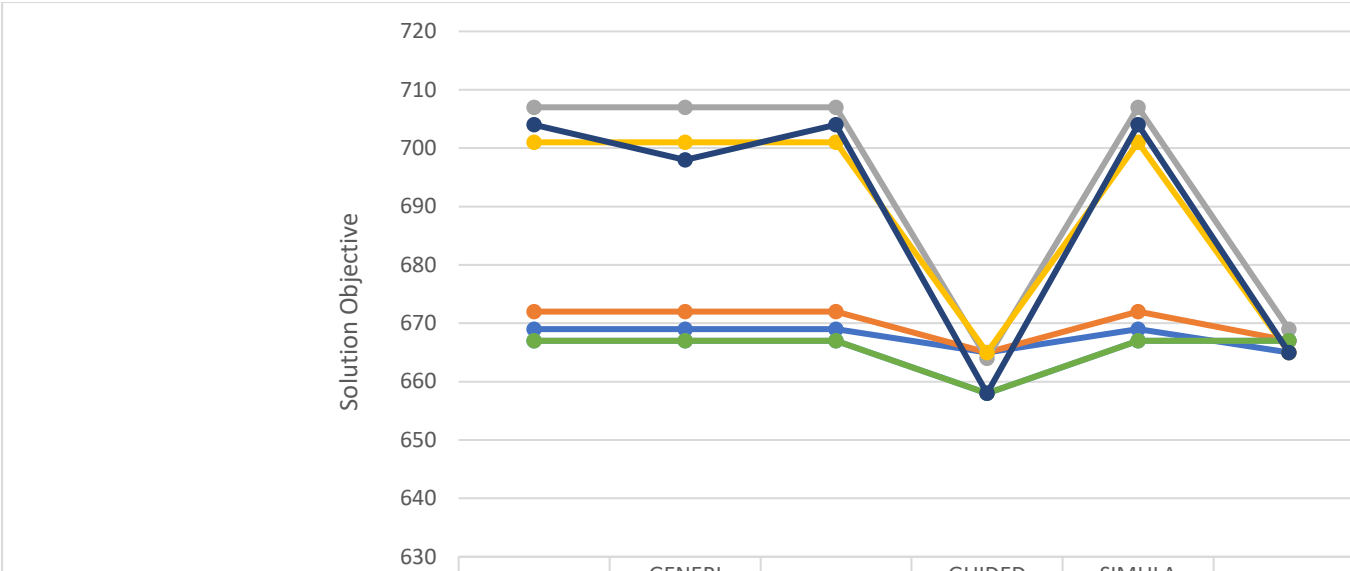
Savings ήταν αυτός που έδωσε τα καλύτερα αποτελέσματα και σε συνδυασμό με το Tabu Search έδωσε την καλύτερη στο πρόβλημα (623 λεπτά). Εξίσου καλά αποτελέσματα έδωσε και ο αλγόριθμος Path Most Constraint Arc, οποίος ήταν σταθερός στις λύσεις του με όλα τα metaheuristic που συνδυάστηκε (627 λεπτά). Την χειρότερη λύση την έδωσε ο αλγόριθμος του Christofide σε συνδυασμό με τα metaheuristic Generic Tabu Search, Greedy Descent και Simulated Annealing (716 λεπτά), ενώ όταν τον συνδυάσαμε με το Guided Local Search η λύση έφτασε πολύ κοντά στην καλύτερη αυτού του προβλήματος (625 λεπτά).

Πίνακας 6.1.1 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0098816	4.0002701	0.0099798	4.0014707	3.9999631	4.0046503
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0084742	4.0000301	0.0083326	4.0001445	4.0000091	4.0006296
PARALLEL_CHEAPEST_INSE	0.0089661	4.0000026	0.0091699	3.9999888	3.9999964	4.0019134
PATH_CHEAPEST_ARC	0.0129079	3.9999018	0.0132709	3.9999040	3.9999274	4.0006998
PATH_MOST CONSTRAINT_ARC	0.0129358	3.9999791	0.0130087	3.9999868	3.9999083	4.0028681
SAVINGS	0.0090145	3.9998880	0.0089579	3.9999921	4.0000138	4.0016257

6.1.2 Μελέτη περίπτωσης για 3 Φορτηγά

Σε αυτό το πείραμα λύσαμε το πρόβλημα με 3 φορτηγά, χωρητικότητας 25 ευρωπαλετών το ένα. Τα αποτελέσματα που προέκυψαν από το πείραμα είναι τα εξής:



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	669	669	669	665	669	665
GLOBAL_CHEAPEST_ARC	672	672	672	665	672	667
LOCAL_CHEAPEST_INSERTION	707	707	707	664	707	669
PARALLEL_CHEAPEST_INSERTION	701	701	701	665	701	665
PATH_CHEAPEST_ARC	667	667	667	658	667	667
PATH_MOST CONSTRAINT_ARC	667	667	667	658	667	667
SAVINGS	704	698	704	658	704	665

Metaheuristic

Σε αυτό το πείραμα μόνο ο αλγόριθμος Local Cheapest Arc δεν κατάφερε να δώσει λύσει στο πρόβλημα, με τους υπόλοιπους αλγόριθμους να λύνουν το πρόβλημα σε μικρό χρονικό διάστημα.

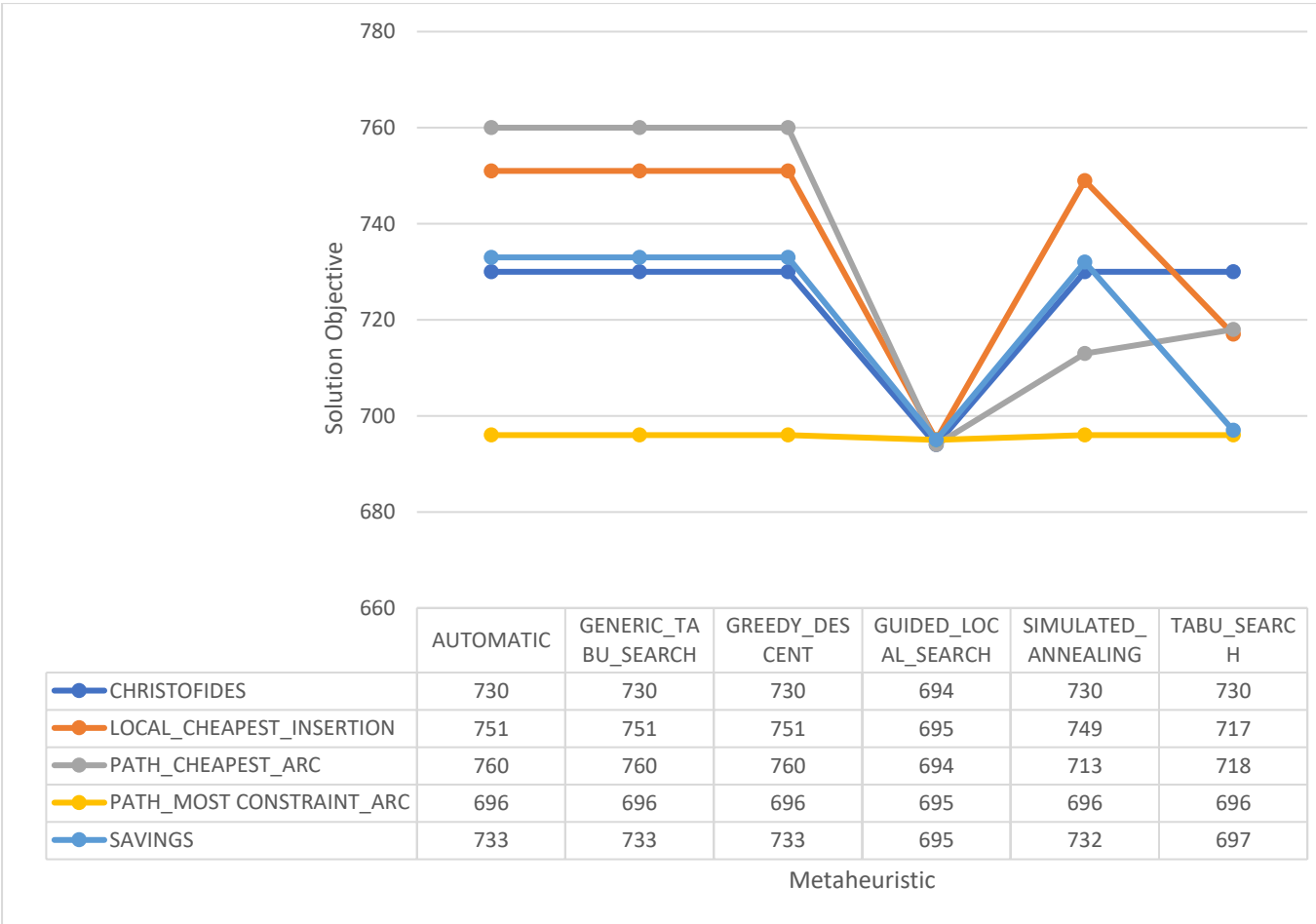
Τις καλύτερες λύσεις τις έδωσαν οι αλγόριθμοι Path Cheapest Arc και Path Most Constraint Arc, όπου σε συνδυασμό με τα metaheuristic Generic Tabu Search, Greedy Descent, Simulated Annealing, Tabu Search έδωσαν λύση 667 λεπτά. Ακόμη καλύτερη ήταν η λύση σε συνδυασμό με τον Guided Local Search, όπου βελτίωσε την λύση και για τους δύο αλγορίθμους στα 658 λεπτά, όπου ήταν και η καλύτερη λύση του πειράματος αυτού. Τέλος, τα χειρότερα αποτελέσματα τα έδωσε ο αλγόριθμος Local Cheapest Insertion όταν συνδυάστηκε με τα Generic Tabu Search, Greedy Descent, Simulated Annealing (707 λεπτά), ενώ με την χρήση του Guided Local Search βελτίωσε τη λύση του στα 664 λεπτά.

Πίνακας 6.1.2 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0259720	4.0001432	0.0249147	4.0005781	4.0000314	4.0008067
GLOBAL_CHEAPEST_ARC	0.0158143	4.0000917	0.0158028	3.9999799	4.0000912	4.0009345
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0094537	4.0001184	0.0094936	4.0004140	3.9999568	4.0006375
PARALLEL_CHEAPEST_INSE	0.0084158	4.0000854	0.0083267	4.0002936	3.9999836	4.0014860
PATH_CHEAPEST_ARC	0.0239557	3.9999590	0.0213364	4.0001804	4.0000807	4.0001341
PATH_MOST CONSTRAINT_ARC	0.0225508	3.9998721	0.0211791	4.0006387	4.0002133	4.0043529
SAVINGS	0.0606400	4.0007260	0.0082792	3.9998650	4.0003141	4.0000396

6.1.3 Μελέτη περίπτωσης για 4 Φορτηγά

Το πείραμα αυτό το μελετήσαμε με 4 φορτηγά, όπου τα 3 έχουν χωρητικότητα 17 ευρωπαλέτες και το ένα 22 ευρωπαλέτες. Τα αποτελέσματα που προέκυψαν από το πείραμα είναι τα εξής:



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	730	730	730	694	730	730
LOCAL_CHEAPEST_INSERTION	751	751	751	695	749	717
PATH_CHEAPEST_ARC	760	760	760	694	713	718
PATH_MOST CONSTRAINT_ARC	696	696	696	695	696	696
SAVINGS	733	733	733	695	732	697

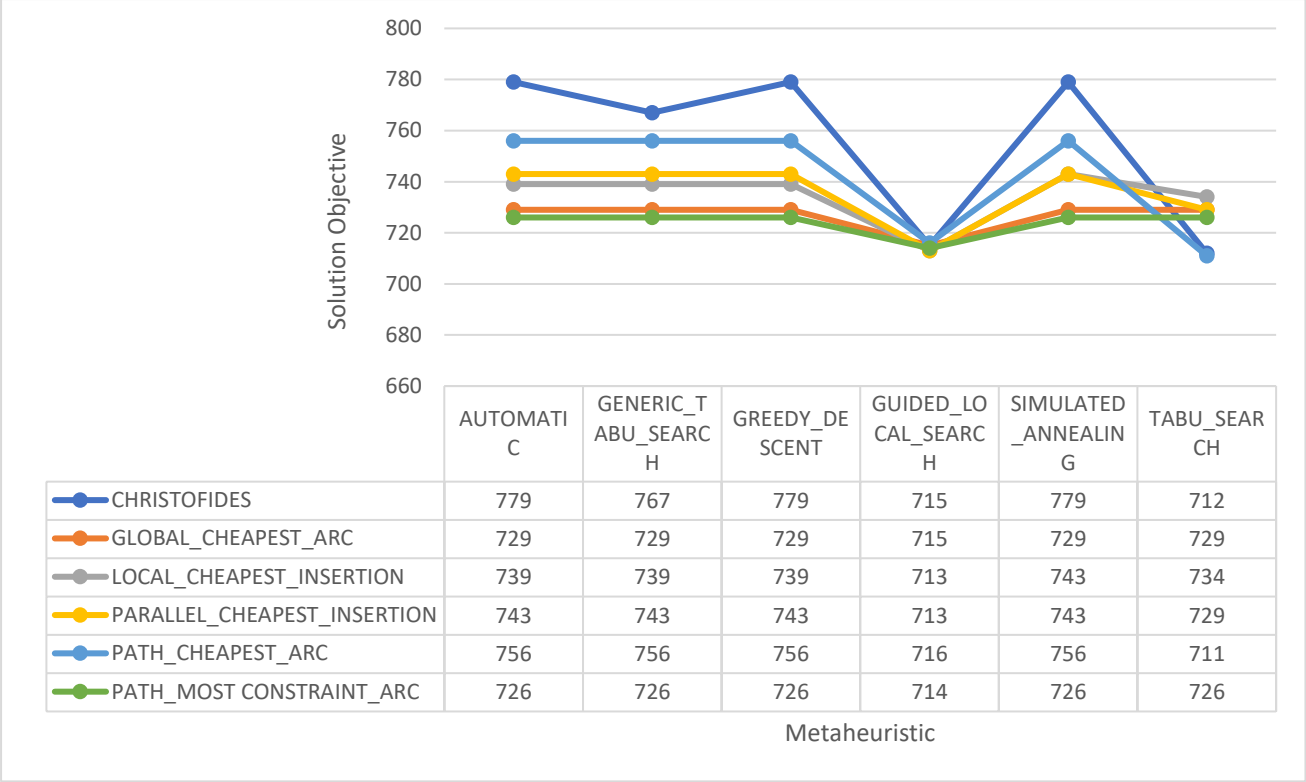
Σε αυτό το πείραμα οι αλγόριθμοι που δεν κατάφεραν να δώσουν λύση ήταν οι Global Cheapest Arc, Local Cheapest Arc και Parallel Cheapest Insertion. Από τους αλγόριθμους που κατάφεραν να λύσουν το πρόβλημα, τις καλύτερες λύσεις συνολικά τις έδωσε ο αλγόριθμος Path Most Constraint Arc. Την χειρότερη και την καλύτερη λύση σε αυτό το πρόβλημα την έδωσε ο αλγόριθμος Path Cheapest Arc. Πιο συγκεκριμένα, την καλύτερη λύση την έδωσε σε συνδυασμό με τον Guided Local Search (694 λεπτά), ενώ τη χειρότερη λύση την έδωσε σε συνδυασμό με τον Generic Tabu Search και Greedy Descent(760 λεπτά).

Πίνακας 6.1.3 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0145561	3.9999932	0.0146430	4.0005633	4.0001177	4.0001784
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0071020	4.0000278	0.0070061	4.0000184	4.0002681	4.0044085
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.0160224	4.0000134	0.0163111	4.0000272	4.0000631	4.0043322
PATH_MOST CONSTRAINT_ARC	0.0147381	4.0000206	0.0146786	4.0001232	3.9999778	4.0000073
SAVINGS	0.0085955	4.0000429	0.0087142	4.0003962	3.9999567	4.0008087

6.1.4 Μελέτη περίπτωσης για 5 Φορτηγά

Το πείραμα αυτό το μελετήσαμε με 5 φορτηγά, 1 χωρητικότητας 18 ευρωπαλετών, 3 χωρητικότητας 17 ευρωπαλετών και 1 χωρητικότητας 4 ευρωπαλέτων. Τα αποτελέσματα που προέκυψαν από το πείραμα είναι τα εξής:



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	779	767	779	715	779	712
GLOBAL_CHEAPEST_ARC	729	729	729	715	729	729
LOCAL_CHEAPEST_INSERTION	739	739	739	713	743	734
PARALLEL_CHEAPEST_INSERTION	743	743	743	713	743	729
PATH_CHEAPEST_ARC	756	756	756	716	756	711
PATH_MOST CONSTRAINT_ARC	726	726	726	714	726	726

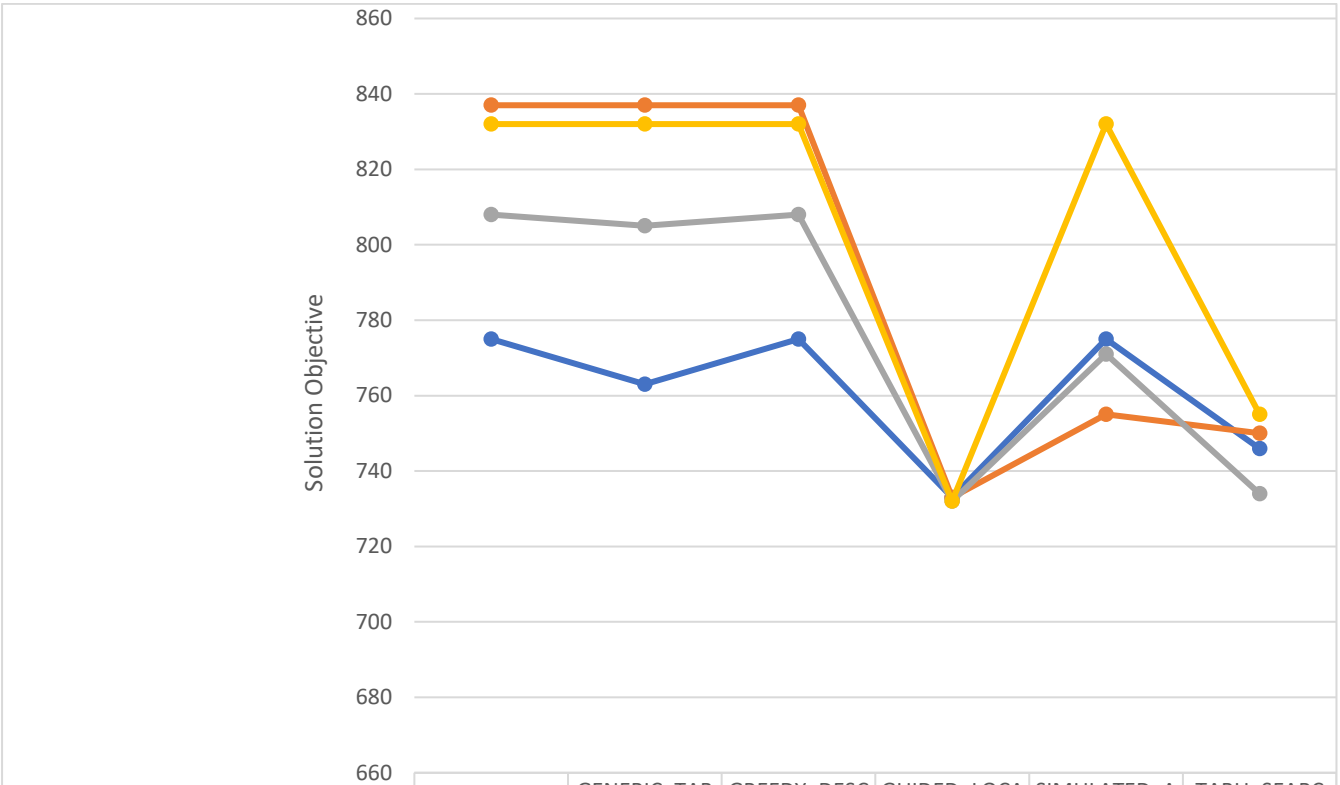
Στο πείραμα 6.1.4. οι αλγόριθμοι που απέτυχαν να λύσουν το πρόβλημα ήταν οι Local Cheapest Arc και Savings. Αντίθετα, οι αλγόριθμοι Local Cheapest Insertion και Parallel Cheapest Insertion, όχι μόνο έλυσαν το πρόβλημα, αλλά σε συνδυασμό με τον Guided Local Search έδωσαν την καλύτερη λύση σε αυτό το πρόβλημα(713 λεπτά).Την χειρότερη λύση στο πρόβλημα την έδωσε ο αλγόριθμος του Christofide σε συνδυασμό με το metaheuristic Greedy Descent(779 λεπτά).

Πίνακας 6.1.4 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0134397	4.0005066	0.0136223	3.9999944	4.0002412	4.0009104
GLOBAL_CHEAPEST_ARC	0.0213812	4.0000153	0.0215108	3.9999796	3.9999671	4.0031419
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0089917	4.0006417	0.0089532	4.0003022	3.9999723	4.0011026
PARALLEL_CHEAPEST_INSE	0.0105847	3.9999253	0.0104082	4.0001228	3.9999418	4.0002985
PATH_CHEAPEST_ARC	0.0620997	4.0001478	0.0098579	4.0011877	4.0002148	4.0037181
PATH_MOST CONSTRAINT_ARC	0.0170577	4.0001544	0.0201930	4.0001657	4.0003732	4.0014133
SAVINGS	-	-	-	-	-	-

6.1.5 Μελέτη περίπτωσης για 6 Φορτηγά

Αυτό το πείραμα προσπαθήσαμε να το λύσουμε με 6 φορτηγά, 3 χωρητικότητας 17 ευρωπαλετών,2 χωρητικότητας 4 ευρωπαλετών και 1 χωρητικότητας 14 ευρωπαλετών. Τα αποτελέσματα που προέκυψαν από το πείραμα είναι τα εξής:



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	775	763	775	733	775	746
LOCAL_CHEAPEST_INSERTION	837	837	837	733	755	750
PATH_CHEAPEST_ARC	808	805	808	732	771	734
PATH_MOST CONSTRAINT_ARC	832	832	832	732	832	755

Σε αυτό το πείραμα οι αλγόριθμοι Global Cheapest Arc, Local Cheapest Arc, Parallel Cheapest Insertion και Savings δεν κατάφεραν να λύσουν το πρόβλημα. Αντιθέτως, οι αλγόριθμοι Christofides, Local Cheapest Insertion, Path Cheapest Arc και Path Most Constraint Arc κατάφεραν να λύσουν το πρόβλημα σε σύντομο χρονικό διάστημα. Τις καλύτερες λύσεις συνολικά τις έδωσε ο αλγόριθμος του Christofide, ενώ τις χειρότερες λύσεις τις έδωσε ο αλγόριθμος Local Cheapest Insertion όταν συνδυάστε με τα metaheuristic Generic Tabu Search και Greedy Descent(837 λεπτά).

Παρ ‘όλα αυτά για ακόμη μια φορά όταν συνδυάστηκε με το metaheuristic Guided Local Search έφτασε στο επίπεδο της καλύτερης λύσης (733 λεπτά).

Πίνακας 6.1.5 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0779620	4.0008309	0.0238133	4.0012931	4.0000343	4.0035744
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0117080	3.9999895	0.0117638	4.0000264	4.0000485	4.0016243
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.0102797	4.0001225	0.0100405	4.0000639	4.0002906	4.0019144
PATH_MOST CONSTRAINT_ARC	0.0122898	4.0000608	0.0125246	3.9998834	4.0004871	4.0044788
SAVINGS	-	-	-	-	-	-

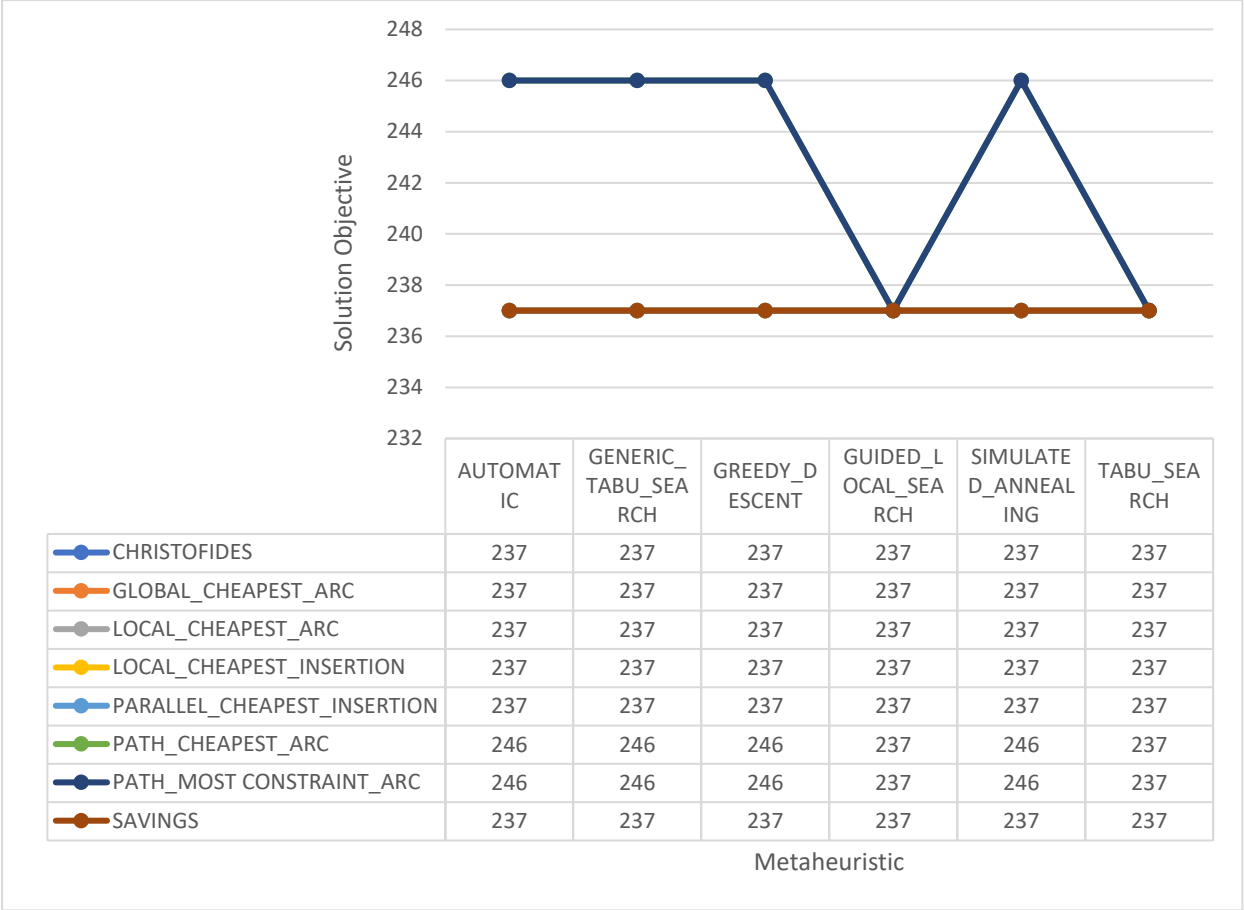
6.1.6 Γενικά συμπεράσματα για την μελέτη Φορτηγών

Σύμφωνα με τα αποτελέσματα παρατηρούμε ότι όσο λιγότερα φορτηγά έχουμε στην διάθεση μας, τόσο καλύτερη είναι η λύση. Έτσι, συμβουλεύουμε την εταιρία ότι ο καλύτερος αριθμός φορτηγών για την εξυπηρέτηση των πελατών της είναι 3 φορτηγά χωρητικότητας 23 ευρωπαλετών το ένα. Προφανώς τα δυο φορτηγά έδωσαν καλύτερα αποτελέσματα από τα τρία, αλλά τα φορτηγά χωρητικότητας 36 ευρωπαλετών δεν μπορούν να κινηθούν μέσα στον ιστό της πόλης, για αυτό και προτείναμε στην εταιρεία τρία φορτηγά χωρητικότητας 23 ευρωπαλετών το ένα.

6.2 Μελέτη περίπτωσης για διαφορετικό αριθμό πελατών

Σε αυτήν την ενότητα μελετήσαμε τη συμπεριφορά των αλγορίθμων και των metaheuristic με βάση τους πελάτες. Κρατήσαμε σταθερά τα χρονοπαράθυρα, τον αριθμό των φορτηγών και τις χωρητικότητες τους.

6.2.1 Μελέτη περίπτωσης για 10 πελάτες



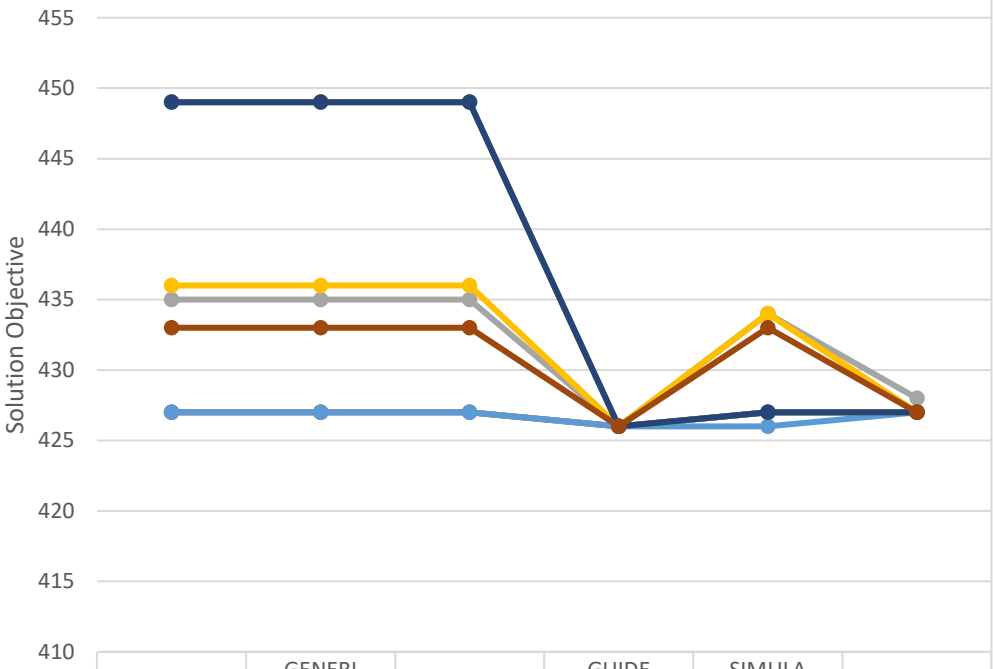
	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	237	237	237	237	237	237
GLOBAL_CHEAPEST_ARC	237	237	237	237	237	237
LOCAL_CHEAPEST_ARC	237	237	237	237	237	237
LOCAL_CHEAPEST_INSERTION	237	237	237	237	237	237
PARALLEL_CHEAPEST_INSERTION	237	237	237	237	237	237
PATH_CHEAPEST_ARC	246	246	246	237	246	237
PATH_MOST CONSTRAINT_ARC	246	246	246	237	246	237
SAVINGS	237	237	237	237	237	237

Σύμφωνα με τα αποτελέσματα του πειράματος 6.2.1, παρατηρούμε ότι σχεδόν όλοι οι αλγόριθμοι μας έδωσαν την ίδια λύση. Οι αλγόριθμοι Global Cheapest Arc, Local Cheapest Arc, Local Cheapest Insertion, Parallel Cheapest Insertion, Savings και Christofides μας έδωσαν ακριβώς τα ίδια αποτελέσματα. Η λύση που δώσανε είναι 237 λεπτά για ένα πλήρες δρομολόγιο εξυπηρέτησης 10 πελατών, η οποία ήταν ίδια με όποιο metaheuristic τους συνδυάσαμε. Την διαφορά έκαναν οι αλγόριθμοι, Path Cheapest Arc και Path Most Constraint Arc, όπου σε συνδυασμό με τα metaheuristic Generic Tabu Search, Greedy Descent και Simulated Annealing έδωσαν λύση 246 λεπτά, μόλις 9 λεπτά περισσότερα από την καλύτερη λύση(237). Παρ ’όλα αυτά, όταν συνδυάστηκαν με τα metaheuristic Guided Local Search και Tabu Search η λύση τους βελτιώθηκε και κατάφεραν να δώσουν το ίδιο καλή λύση με τους προηγούμενους αλγόριθμους, δηλαδή 237 λεπτά για ένα πλήρες δρομολόγιο εξυπηρέτησης 10 πελατών.

Πίνακας 6.2.1 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0041400	3.9998182	0.0041416	3.9997080	3.9996759	4.0008852
GLOBAL_CHEAPEST_ARC	0.0034344	3.9997095	0.0035371	4.0004538	3.9997160	4.0001743
LOCAL_CHEAPEST_ARC	0.0032001	3.9996874	0.0032006	4.0000709	3.9996676	3.9997494
LOCAL_CHEAPEST_INSERT	0.0029821	3.9997586	0.0029756	3.9996786	3.9997424	4.0001586
PARALLEL_CHEAPEST_INSE	0.0034253	3.9996945	0.0033836	3.9998740	3.9996162	3.9998583
PATH_CHEAPEST_ARC	0.0034990	3.9998968	0.0034432	4.0000037	3.9996793	3.9999116
PATH_MOST CONSTRAINT_ARC	0.0036946	4.0006576	0.0032811	3.9998196	3.9998051	3.9998114
SAVINGS	0.0025177	3.9998266	0.0026363	3.9997462	3.9997566	4.0007689

6.2.2 Μελέτη περίπτωσης για 15 πελάτες



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	427	427	427	426	427	427
GLOBAL_CHEAPEST_ARC	427	427	427	426	427	427
LOCAL_CHEAPEST_ARC	435	435	435	426	434	428
LOCAL_CHEAPEST_INSERTION	436	436	436	426	434	427
PARALLEL_CHEAPEST_INSERTION	427	427	427	426	426	427
PATH_CHEAPEST_ARC	449	449	449	426	427	427
PATH_MOST CONSTRAINT_ARC	449	449	449	426	427	427
SAVINGS	433	433	433	426	433	427

Σε αυτό το πείραμα καθώς προσθέσαμε πέντε ακόμα πελάτες στο δρομολόγιο, παρατηρούμε ότι οι λύσεις των αλγορίθμων με τα εκάστοτε metaheuristic αρχίζουν να έχουν μικρές διάφορες μεταξύ τους. Η καλύτερη λύση είναι τα 426 λεπτά και την έδωσαν όλοι οι αλγόριθμοι όταν συνδυάστηκαν με το metaheuristic Guided Local Search. Η χειρότερη λύση είναι τα 449 λεπτά και την παρήγαγαν οι αλγόριθμοι Path Cheapest Arc και Path Most Constraint Arc όταν συνδυάστηκαν με τα metaheuristic Generic Tabu Search και Greedy Descent.

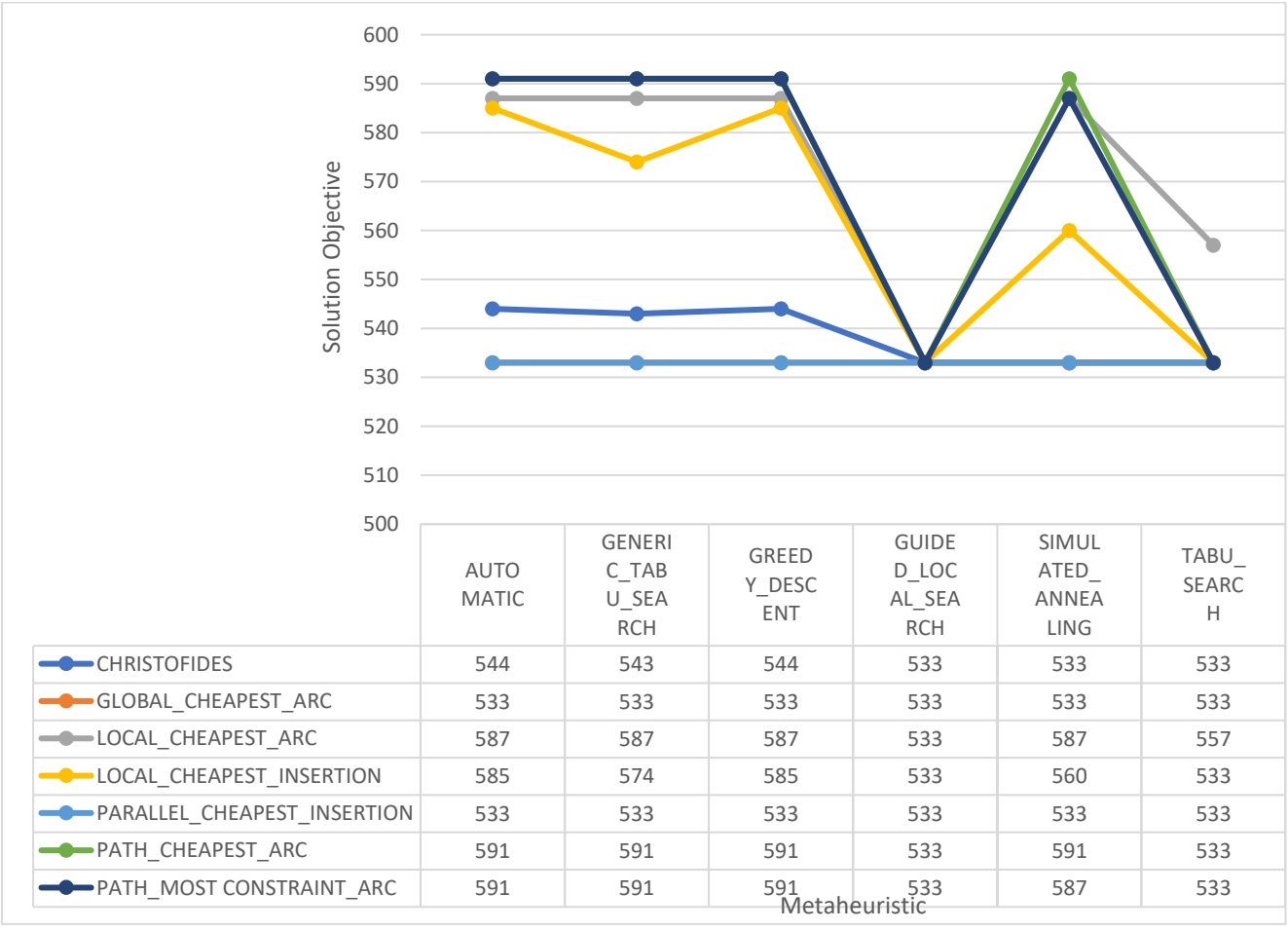
Αξίζει να σημειώσουμε ότι σε αυτό το πείραμα οι αλγόριθμοι Global Cheapest Arc και Christofides ήταν σταθεροί στις λύσεις που δίναν χωρίς να επηρεάζονται από το metaheuristic που συνδυαζόταν. Το ίδιο ισχύει και για τα metaheuristic Guided

Local Search και Tabu Search, τα οποία παρέμεναν σταθερά στις λύσεις που έδιναν, χωρίς να επηρεάζονται από τον αλγόριθμο που συνδυαζόταν.

Πίνακας 6.2.2 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0058747	4.0000070	0.0058068	4.0000541	4.0000787	4.0007191
GLOBAL_CHEAPEST_ARC	0.0056144	4.0000674	0.0068870	3.9997395	3.9997653	3.9997790
LOCAL_CHEAPEST_ARC	0.0082098	3.9997924	0.0084727	3.9999180	4.0000581	3.9998290
LOCAL_CHEAPEST_INSERT	0.0052650	3.9999044	0.0052506	4.0001235	3.9997442	3.9999356
PARALLEL_CHEAPEST_INSE	0.0073826	4.0001629	0.0075523	3.9998625	3.9997677	4.0003351
PATH_CHEAPEST_ARC	0.0048156	3.9997838	0.0045789	3.9998546	3.9998548	4.0004784
PATH_MOST CONSTRAINT_ARC	0.0043920	3.9998483	0.0048101	3.9998667	3.9997707	4.0000647
SAVINGS	0.0038253	3.9998123	0.0037731	3.9999060	3.9997909	4.0013222

6.2.3 Μελέτη περίπτωσης για 20 πελάτες



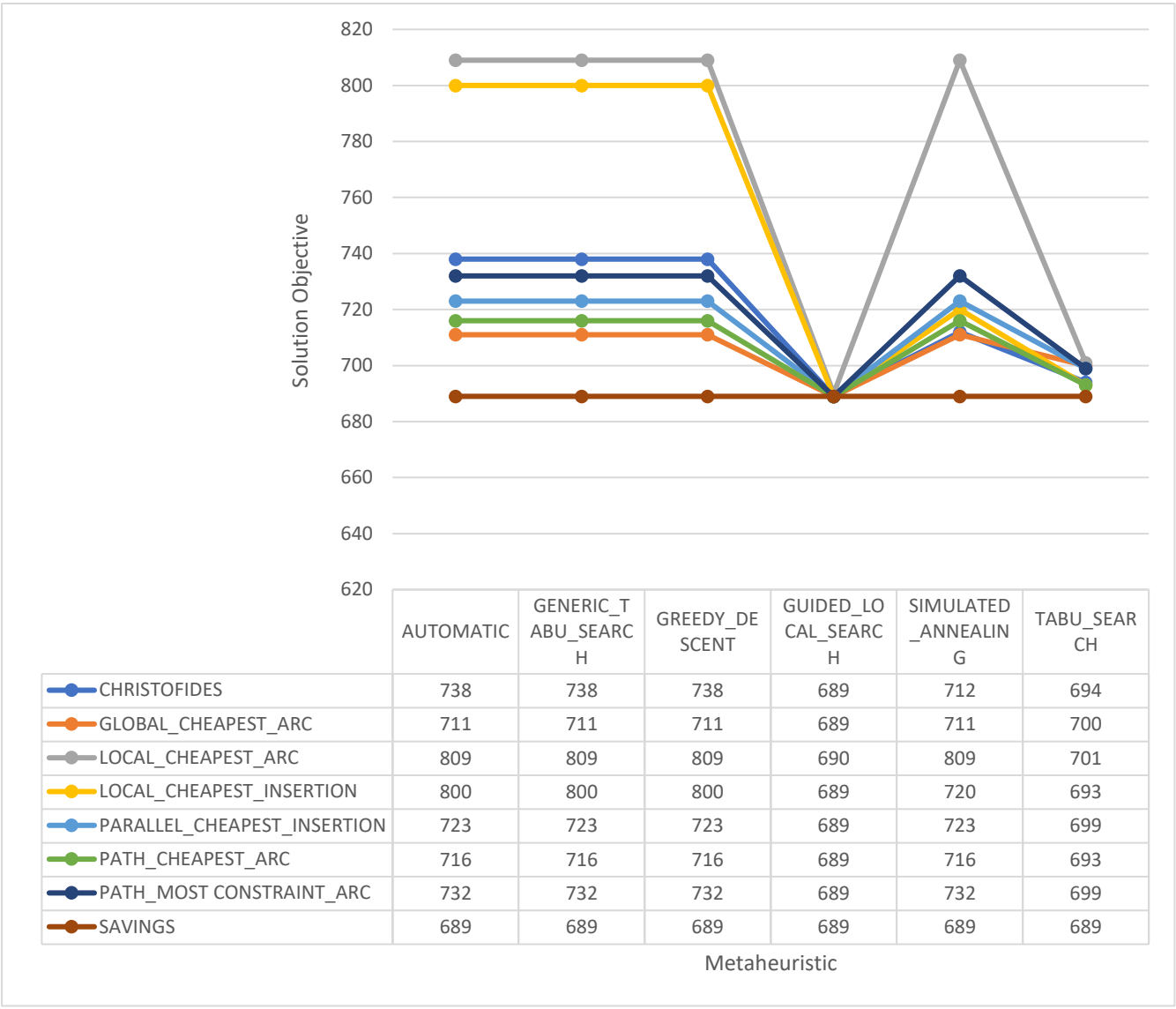
Στο πείραμα 6.2.3 ο αλγόριθμος Savings δεν κατάφερε να λύσει το πρόβλημα, ενώ οι υπόλοιποι 7 αλγόριθμοι κατάφεραν σε συνδυασμό με τα metaheuristic να λύσουν το πρόβλημα σε μικρό χρονικό διάστημα. Πιο συγκεκριμένα, οι αλγόριθμοι Global Cheapest Arc και Parallel Cheapest Insertion ήταν σταθεροί στην λύση τους με όποιο metaheuristic συνδυάστηκαν και έδωσαν την καλύτερη λύση στο πείραμα αυτό, η οποία είναι τα 533 λεπτά. Η χειρότερη λύση είναι τα 591 λεπτά και την έδωσε ο αλγόριθμος Path Cheapest Arc όταν συνδυάστηκε με τα metaheuristic Generic Tabu Search,Greedy Descent και Simulated Annealing και ο αλγόριθμος Path Most Constraint Arc όταν συνδυάστηκε με τα metaheuristic Generic Tabu Search και Greedy Descent. Παρ ’όλο που οι δυο αυτοί αλγόριθμοι σε συνδυασμό με τα προαναφερθέντα metaheuristic έδωσαν τα χειρότερα αποτελέσματα σε αυτό το πείραμα, όταν συνδυάστηκαν με τα metaheuristic Guided Local Search και Tabu Search, όχι μόνο βελτίωσαν την λύση τους, αλλά έφτασαν και στο επίπεδο της καλύτερης λύσης, τα 533 λεπτά.

Αξίζει να επισημάνουμε ότι για ακόμα μια φορά το metaheuristic Guided Local Search ήταν σταθερό στην λύση που έδινε, η οποία ήταν και η καλύτερη για το πείραμα 6.2.3, με όποιον αλγόριθμο και αν συνδυάστηκε. Αρκετά καλά αποτελέσματα παρουσίασε και το metaheuristic Tabu Search,το οποίο βοήθησε όλους τους αλγόριθμους να επιτύχουνε την καλύτερη λύση(533 λεπτά) για αυτό το πείραμα, εκτός του αλγορίθμου Local Cheapest Arc.

Πίνακας 6.2.3 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0107162	4.0001896	0.0104286	3.9998293	3.9999006	4.0025768
GLOBAL_CHEAPEST_ARC	0.0063414	3.9998184	0.0062906	3.9998794	3.9998970	3.9999780
LOCAL_CHEAPEST_ARC	0.0124788	4.0000710	0.0125418	3.9999008	4.0001676	4.0001463
LOCAL_CHEAPEST_INSERT	0.0082605	3.9999248	0.0080919	3.9998968	3.9999989	4.0018647
PARALLEL_CHEAPEST_INSE	0.0087586	4.0003674	0.0085105	4.0000641	4.0001416	4.0017591
PATH_CHEAPEST_ARC	0.0063965	4.0002686	0.0063715	3.9998363	3.9998555	3.9998375
PATH_MOST CONSTRAINT_ARC	0.0064867	3.9998115	0.0066174	3.9998934	3.9999096	4.0007138
SAVINGS	-	-	-	-	-	-

6.2.4 Μελέτη περίπτωσης για 25 πελάτες



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	738	738	738	689	712	694
GLOBAL_CHEAPEST_ARC	711	711	711	689	711	700
LOCAL_CHEAPEST_ARC	809	809	809	690	809	701
LOCAL_CHEAPEST_INSERTION	800	800	800	689	720	693
PARALLEL_CHEAPEST_INSERTION	723	723	723	689	723	699
PATH_CHEAPEST_ARC	716	716	716	689	716	693
PATH_MOST CONSTRAINT_ARC	732	732	732	689	732	699
SAVINGS	689	689	689	689	689	689

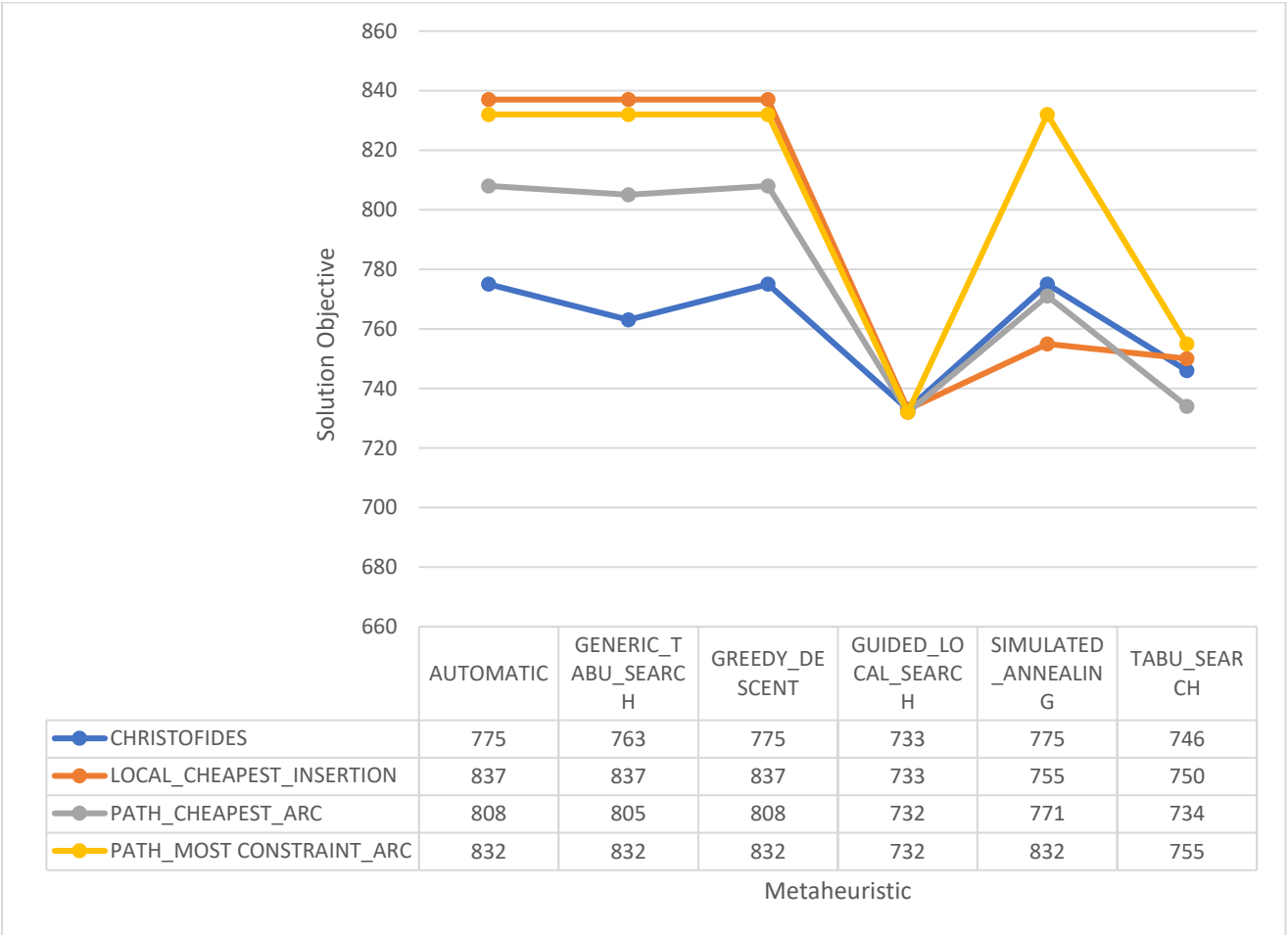
Metaheuristic

` Σε αυτό το πείραμα ο αλγόριθμος Savings, όχι μόνο δεν απέτυχε, αλλά ήταν και αυτός που έδωσε την καλύτερη λύση στο πρόβλημα(689 λεπτά), στην οποία ήταν σταθερός, με οποιοδήποτε metaheuristic και αν συνδυάστηκε. Την χειρότερη λύση(809 λεπτά) την παρήγαγε ο αλγόριθμος Local Cheapest Arc όταν συνδυάστηκε με τα metaheuristic Generic Tabu Search,Greedy Descent και Simulated Annealing, ενώ με τη βοήθεια του Guided Local Search έφτασε και αυτός στο επίπεδο της καλύτερης λύσης (690 λεπτά). Τέλος, το metaheuristic Guided Local Search ήταν σταθερό στην λύση που έδινε για ακόμη μια φορά και ήταν αυτό που βοήθησε όλους τους αλγόριθμους να επιτύχουν την καλύτερη λύση.

Πίνακας 6.2.4 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0158784	4.0002340	0.0153770	4.0000304	4.0001078	4.0000353
GLOBAL_CHEAPEST_ARC	0.0106497	3.9998905	0.0099733	3.9999803	3.9999130	4.0027366
LOCAL_CHEAPEST_ARC	0.0114688	4.0001043	0.0110084	4.0004748	4.0000034	4.0004744
LOCAL_CHEAPEST_INSERT	0.0091650	4.0000642	0.0091620	4.0000873	3.9998969	4.0004984
PARALLEL_CHEAPEST_INSE	0.0068720	4.0002727	0.0068942	4.0001105	3.9999356	4.0038398
PATH_CHEAPEST_ARC	0.0100619	4.0002784	0.0102820	4.0000429	3.9999309	4.0033477
PATH_MOST CONSTRAINT_ARC	0.0124380	3.9999303	0.0109555	3.9999416	3.9999435	4.0008141
SAVINGS	0.0075296	3.9998644	0.0074781	4.0000646	4.0000189	4.0009640

6.2.5 Μελέτη περίπτωσης για 30 πελάτες



Σε αυτό το πείραμα οι αλγόριθμοι Global Cheapest Arc, Local Cheapest Arc, Parallel Cheapest Insertion και Savings απέτυχαν στο να δώσουν λύση. Αντιθέτως, οι

αλγόριθμοι Christofides, Local Cheapest Insertion, Path Cheapest Arc και Path Most Constraint Arc κατάφεραν να λύσουν το πρόβλημα σε σύντομο χρονικό διάστημα. Τις καλύτερες λύσεις συνολικά τις έδωσε ο αλγόριθμος του Christofide, ενώ τις χειρότερες λύσεις τις έδωσε ο αλγόριθμος Local Cheapest Insertion όταν συνδυάστε με τα metaheuristic Generic Tabu Search και Greedy Descent(837 λεπτά). Παρ΄όλα αυτά για ακόμη μια φορά όταν συνδυάστηκε με το metaheuristic Guided Local Search έφτασε στο επίπεδο της καλύτερης λύσης (733 λεπτά).

Πίνακας 6.2.5 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0779620	4.0008309	0.0238133	4.0012931	4.0000343	4.0035744
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0117080	3.9999895	0.0117638	4.0000264	4.0000485	4.0016243
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.0102797	4.0001225	0.0100405	4.0000639	4.0002906	4.0019144
PATH_MOST CONSTRAINT_ARC	0.0122898	4.0000608	0.0125246	3.9998834	4.0004871	4.0044788
SAVINGS	-	-	-	-	-	-

6.2.6 Γενικά συμπεράσματα για την μελέτη διαφορετικού αριθμού Πελατών.

Σύμφωνα με τα πειράματα που πραγματοποιήσαμε, παρατηρήσαμε ότι όσο

πιο μικρό ήταν το μέγεθος των πελατών, τόσο τα αποτελέσματα των αλγορίθμων δεν είχαν διαφορές μεταξύ τους. Έτσι καθώς αρχίσαμε να αυξάνουμε τους πελάτες, τα αποτελέσματα των αλγορίθμων αρχίσαν να έχουν διάφορες μεταξύ τους. Όσο μεγάλωνε ο αριθμός των πελατών, τόσο πιο αισθητές γινόταν οι διαφορές, με χαρακτηριστικό παράδειγμα, το πείραμα 6.2.4, όπου η χειρότερη τιμή ήταν τα 809 λεπτά και η καλύτερη τα 689 λεπτά. Τέλος, σημαντική παρατήρηση αποτελεί το γεγονός ότι όσο μεγαλύτερος ήταν ο αριθμός των πελατών, τόσο λιγότεροι ήταν οι αλγόριθμοι που κατάφερναν να λύνουν το πρόβλημα, με χαρακτηριστικό

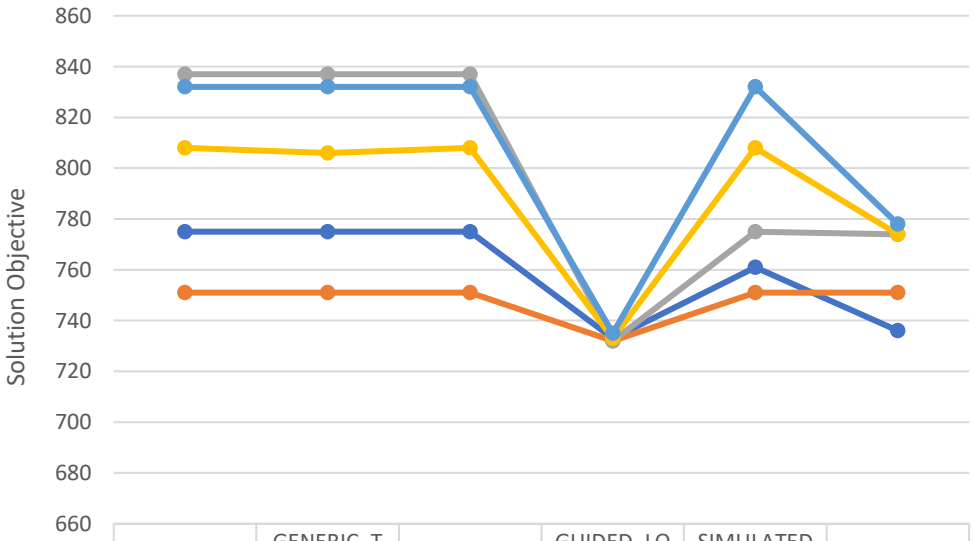
παράδειγμα, το πείραμα 6.2.5, όπου οι 4 από τους 8 αλγόριθμους κατάφεραν να δώσουν λύση.

6.3 Μελέτη περίπτωσης για τα Χρονοπαράθυρα

Σε αυτή την ενότητα μελετήσαμε τη συμπεριφορά των αλγορίθμων και των metaheuristic με βάση τις αλλαγές στα χρονοπαράθυρα. Πιο συγκεκριμένα, κρατήσαμε σταθερούς τους πελάτες, τον αριθμό των φορτηγών και τις χωρητικότητες τους, ενώ αλλάζαμε κάθε φορά τα χρονοπαράθυρα.

Μελετήσαμε πέντε διαφορετικές περιπτώσεις. Στην πρώτη περίπτωση μελετήσαμε το πρόβλημα χωρίς χρονοπαράθυρα. Στην δεύτερη περίπτωση προσθέσαμε τα χρονοπαράθυρα που μας έδωσε η εταιρία, ενώ στις επόμενες περιπτώσεις κάναμε όλο και πιο αυστηρά τα υπάρχων χρονοπαράθυρα. Συνεπώς, η πρώτη περίπτωση ξεκινάει με βαθμό δυσκολίας το 0 και σε κάθε περίπτωση που προχωράμε ο βαθμός δυσκολίας αυξάνεται, με μέγιστο βαθμό δυσκολίας το 4.

6.3.1 Μελέτη περίπτωσης χωρίς Χρονοπαράθυρα



	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	775	775	775	733	761	736
LOCAL_CHEAPEST_ARC	751	751	751	732	751	751
LOCAL_CHEAPEST_INSERTION	837	837	837	732	775	774
PATH_CHEAPEST_ARC	808	806	808	733	808	774
PATH_MOST CONSTRAINT_ARC	832	832	832	735	832	778

Metaheuristic

Σε αυτό το πείραμα οι αλγόριθμοι Global Cheapest Arc,Parallel Cheapest Arc και Savings απέτυχαν στο να δώσουν λύση στο πρόβλημα. Αντιθέτως, ο αλγόριθμος Local Cheapest Arc έδωσε συνολικά τα καλύτερα αποτελέσματα και όταν συνδυάστηκε με το metaheuristic Guided Local Search έδωσε την καλύτερη λύση στο πείραμα (732 λεπτά). Την χειρότερη λύση την έδωσε ο αλγόριθμος Local Cheapest Insertion όταν συνδυάστηκε με τα metaheuristic Generic Tabu Search και Greedy Descent (837 λεπτά). Παρόλα αυτά όταν τον συνδυάσαμε με το metaheuristic Guided Local Search βελτίωσε σημαντικά την λύση του με αποτέλεσμα να φτάσει στο επίπεδο της καλύτερης λύσης σε αυτό το πείραμα (732 λεπτά).

Το metaheuristic Guided Local Search έδωσε τις καλύτερες λύσεις σε αυτό το πείραμα με όποιον αλγόριθμο και αν το συνδυάσαμε, ενώ τα metaheuristic Generic Tabu Search και Greedy Descent ήταν αυτά που έδιναν τις χειρότερες λύσεις.

Πίνακας 6.3.1 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0256975	4.0006038	0.0251558	4.0004724	4.0001493	4.0001642
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	2.8367090	4.0003420	2.8071555	4.0003039	4.0004014	4.0004490
LOCAL_CHEAPEST_INSERT	0.0117796	4.0009187	0.0119677	3.9994878	4.0001347	4.0007262
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.0137690	4.0002046	0.0139667	4.0001742	4.0001705	4.0015445
PATH_MOST CONSTRAINT_ARC	0.0675828	3.9993781	0.0130961	4.0002145	4.0002611	4.0001383
SAVINGS	-	-	-	-	-	-

6.3.2 Μελέτη περίπτωσης για τα Χρονοπαράθυρα(βαθμού δυσκολίας 1)

	AUTOMATIC	GENERIC_TABU_SEARCH	GREEDY_DESCENT	GUIDED_LOCAL_SEARCH	SIMULATED_ANNEALING	TABU_SEARCH
CHRISTOFIDES	775	763	775	733	775	746
LOCAL_CHEAPEST_INSERTION	837	837	837	733	755	750
PATH_CHEAPEST_ARC	808	805	808	732	771	734
PATH_MOST CONSTRAINT_ARC	832	832	832	732	832	755

Metaheuristic

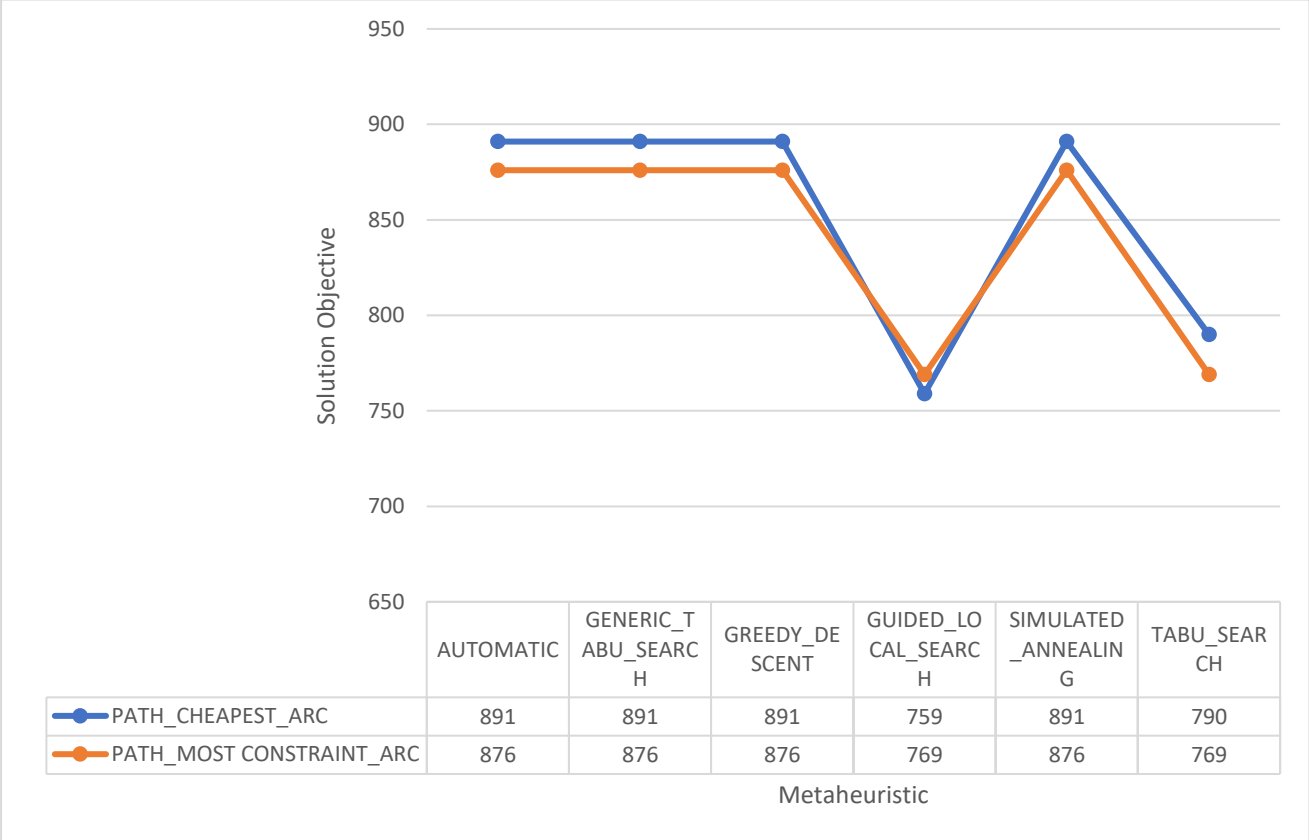
Με την εισαγωγή των χρονοπαραθύρων παρατηρούμε ότι ακόμη λιγότεροι αλγόριθμοι καταφέρνουν να λύσουν το πρόβλημα. Τις καλύτερες λύσεις συνολικά τις έδωσε ο αλγόριθμος του Christofide, ενώ τις χειρότερες λύσεις τις έδωσε ο αλγόριθμος Local Cheapest Insertion όταν συνδυάστε με τα metaheuristic Generic Tabu Search και Greedy Descent(837 λεπτά). Παρ ΄όλα αυτά για ακόμη μια φορά όταν συνδυάστηκε με το metaheuristic Guided Local Search έφτασε στο επίπεδο της καλύτερης λύσης (733 λεπτά).

Το metaheuristic Guided Local Search ήταν αυτό που βοηθούσε τους αλγόριθμους να επιτύχουν τις καλύτερες λύσεις, σε αντίθεση με τα metaheuristic Generic Tabu Search και Greedy Descent τα οποία για ακόμα μια φορά δεν παρήγαν καλές λύσεις σε σχέση με το metaheuristic Guided Local Search.

Πίνακας 6.3.2 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	0.0779620	4.0008309	0.0238133	4.0012931	4.0000343	4.0035744
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	0.0117080	3.9999895	0.0117638	4.0000264	4.0000485	4.0016243
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.0102797	4.0001225	0.0100405	4.0000639	4.0002906	4.0019144
PATH_MOST CONSTRAINT_ARC	0.0122898	4.0000608	0.0125246	3.9998834	4.0004871	4.0044788
SAVINGS	-	-	-	-	-	-

6.3.3 Μελέτη περίπτωσης για τα Χρονοπαράθυρα(βαθμού δυσκολίας 2)



Σε αυτό το πείραμα κάναμε ακόμα πιο αυστηρά τα χρονοπαράθυρα με αποτέλεσμα μόνο δυο αλγόριθμοι να καταφέρουν να λύσουν το πρόβλημα. Ο αλγόριθμος Path Cheapest Arc σε συνδυασμό με τα metaheuristic Generic Tabu

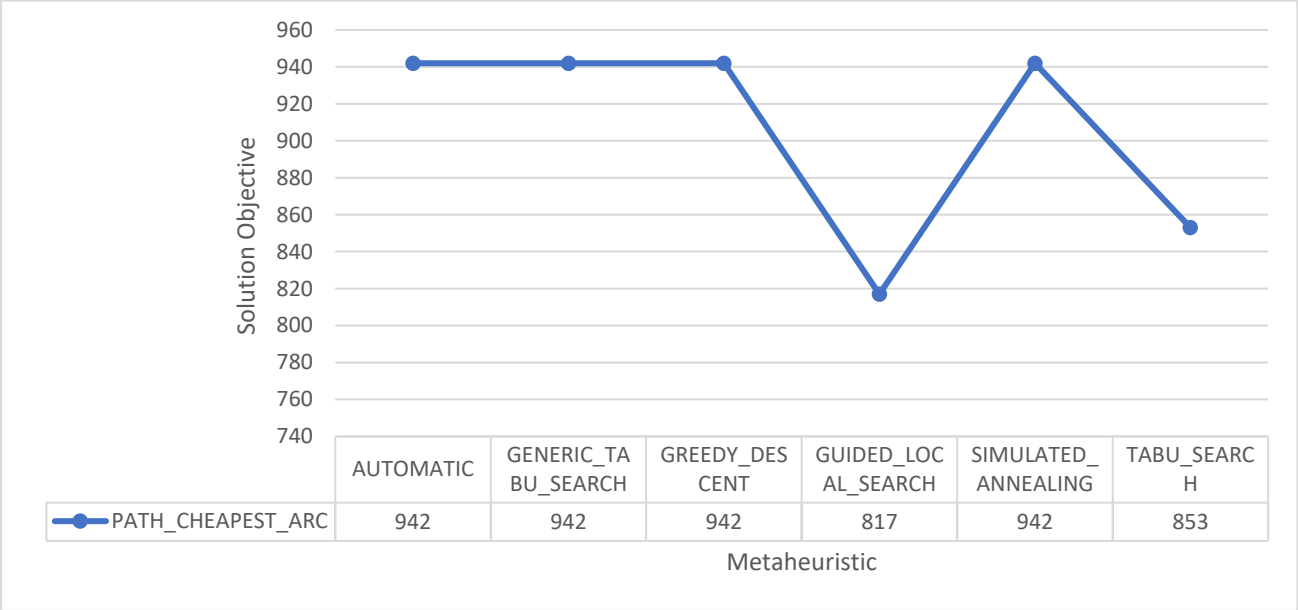
Search, Greedy Descent και Simulated Annealing παρήγαγε τις χειρότερες λύσεις του προβλήματος (891 λεπτά) ,ενώ σε συνδυασμό με το metaheuristic Guided Local Search κατάφερε να δώσει την καλύτερη λύση στο πείραμα (759 λεπτά). Ανάμεσα σε αυτές τις τιμές κινήθηκε και ο αλγόριθμος Path Most Constraint Arc.

Τα metaheuristic Generic Tabu Search, Greedy Descent και Simulated Annealing παρήγαγαν τις χειρότερες σε αντίθεση με το metaheuristic Guided Local Search που βοήθησε τους αλγόριθμούς να επιτύχουν καλές λύσεις.

Πίνακας 6.3.3 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	-	-	-	-	-	-
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	-	-	-	-	-	-
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.0321332	4.0003180	0.0280151	4.0008275	4.0002616	4.0028733
PATH_MOST CONSTRAINT_ARC	0.0312384	4.0003531	0.0304992	4.0002234	3.9995256	4.0009836
SAVINGS	-	-	-	-	-	-

6.3.4 Μελέτη περίπτωσης για τα Χρονοπαράθυρα(βαθμού δυσκολίας 3)

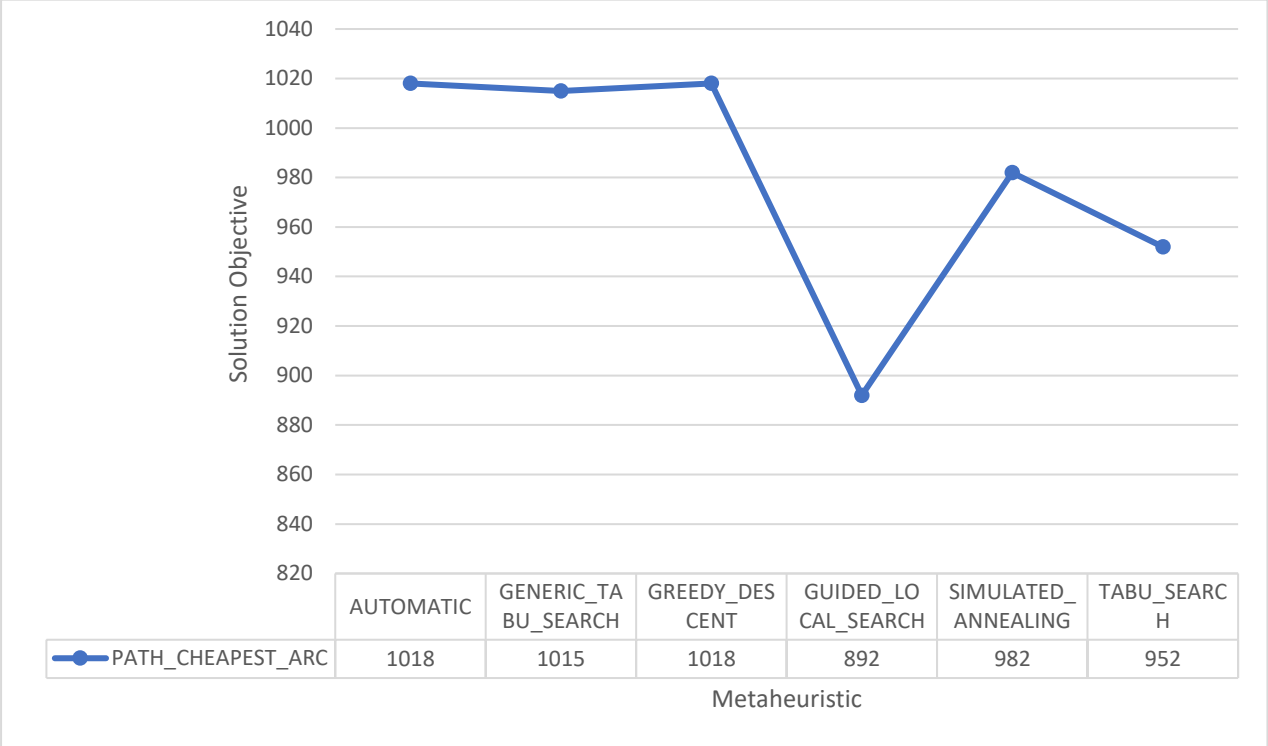


Σε αυτό το πείραμα μονό ένας αλγόριθμος κατάφερε να λύσει το πρόβλημα και αυτός ήταν ο Path Cheapest Arc. Με τη βοήθεια του Guided Local Search κατάφερε να επιτύχει την καλύτερη λύση (817 λεπτά), ενώ σε συνδυασμό με τα metaheuristic Generic Tabu Search, Greedy Descent και Simulated Annealing έδωσε την χειρότερη λύση (942 λεπτά).

Πίνακας 6.3.4 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα.

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	-	-	-	-	-	-
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	-	-	-	-	-	-
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.4653331	4.0002542	0.4225919	3.9999227	4.0003361	4.0001108
PATH_MOST CONSTRAINT_ARC	-	-	-	-	-	-
SAVINGS	-	-	-	-	-	-

6.3.5 Μελέτη περίπτωσης για τα Χρονοπαράθυρα(βαθμού δυσκολίας 4)



Στο πείραμα 6.3.5 με τα πιο αυστηρά χρονοπαράθυρα που μελετήσαμε, μόνο ο αλγόριθμος Path Cheapest Arc κατάφερε να δώσει λύση. Την καλύτερη λύση την έδωσε με τη βοήθεια του Guided Local Search (892 λεπτά), ενώ την χειρότερη λύση την έδωσε σε συνδυασμό με το Greedy Descent (1018 λεπτά).

Πίνακας 6.3.5 Χρόνος επίλυσης των metaheuristic σε δευτερόλεπτα

Όνομα	AUTOMATIC	GENERIC TABU SEARCH	GREEDY DESCENT	GUIDED LOCAL SEARCH	SIMULATED ANNEALING	TABU SEARCH
CHRISTOFIDES	-	-	-	-	-	-
GLOBAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_ARC	-	-	-	-	-	-
LOCAL_CHEAPEST_INSERT	-	-	-	-	-	-
PARALLEL_CHEAPEST_INSE	-	-	-	-	-	-
PATH_CHEAPEST_ARC	0.8654839	4.0001125	0.8727800	3.9997874	4.0001420	4.0003103
PATH_MOST CONSTRAINT_ARC	-	-	-	-	-	-
SAVINGS	-	-	-	-	-	-

6.3.6 Γενικά συμπεράσματα για την μελέτη των Χρονοπαραθύρων

Σύμφωνα με τα πειράματα που πραγματοποιήσαμε, συμπεράναμε ότι όσο πιο αυστηρά γινόταν τα χρονοπαράθυρα, τόσο πιο δύσκολο ήταν για τους αλγόριθμους να λύσουν το πρόβλημα. Χαρακτηριστικό παράδειγμα είναι τα πειράματα 6.3.4 και 6.3.5, όπου μόνο ο αλγόριθμος Path Cheapest Arc κατάφερε να δώσει λύσεις.

Κεφάλαιο 7. Συμπεράσματα για τους αλγορίθμους και τα metaheuristic.

7.1 Συμπεράσματα για τους αλγορίθμους.

7.1.1 Συμπεράσματα για τον αλγόριθμο του Christofide.

Ο αλγόριθμος του Christofide έδωσε αρκετά καλά αποτελέσματα κατά την διάρκεια της μελέτης. Πιο συγκεκριμένα, κατάφερε να λύσει όλα τα πειράματα στις μελέτες περίπτωσης τόσο των φορτηγών, όσο και των πελατών. Αντιθέτως, η επίδοση του δεν καλή στην μελέτη των χρονοπαραθύρων καθώς όταν τα χρονοπαράθυρα άρχισαν να γίνονται πιο αυστηρά, αδυνατούσε να λύσει το πρόβλημα. Τέλος, η λύση του επηρεαζόταν κάθε φορά από το metaheuristic που τον συνδυάζαμε, αφού παρουσίαζε μια μικρή μεταβλητότητα στις λύσεις του σε κάθε πείραμα.

7.1.2. Συμπεράσματα για τον αλγόριθμο Global Cheapest Arc.

Πολύ κακή ήταν απόδοση του αλγορίθμου Global Cheapest Arc, καθώς απέτυχε να δώσει λύση σε όλα τα πειράματα στην μελέτη για τα χρονοπαράθυρα, ενώ κακή ήταν και η απόδοση του στην μελέτη των φορτηγών, αφού κατάφερε να λύσει μόνο τα 2 από τα 5 πειράματα. Παρ΄όλα αυτά, όταν κατάφερνε να λύσει ένα πρόβλημα, έδινε αρκετά καλές λύσεις σε σχέση με τους υπόλοιπους αλγόριθμος, ενώ σε κάθε πείραμα οι λύσεις του δεν είχαν μεγάλες διάφορες με το κάθε metaheuristic που συνδυαζόταν. Αρκετά καλύτερη ήταν η απόδοση του στην μελέτη των πελατών,

7.2.3 Συμπεράσματα για το metaheuristic Simulated Annealing.

Λίγο καλύτερα ήταν τα αποτελέσματα του Simulated Annealing από τα προηγούμενα δυο metaheuristic, ενώ κάποιες φορές παρήγαγε και αυτό τις ίδιες ακριβώς λύσεις με το Generic Tabu Search και Greedy Descent.

7.2.4 Συμπεράσματα για το metaheuristic Tabu Search.

Αρκετά καλή ήταν η επίδοση του metaheuristic Tabu Search, αφού βοηθούσε αρκετά τον κάθε αλγόριθμο να φτάσει πολύ κοντά στην καλύτερη λύση του κάθε πειράματος, ενώ αρκετές φορές βοήθησε τους αλγόριθμους να φτάσουν στην καλύτερη λύση.

7.2.5 Συμπεράσματα για το metaheuristic Guided Local Search.

Το metaheuristic Guided Local Search ήταν αυτό που είχε την καλύτερη επίδοση από όλα τα metaheuristic σε αυτήν μελέτη, καθώς βοήθησε όλους τους αλγόριθμους, όσο κακές λύσεις και να παρήγαν με τα υπόλοιπα metaheuristic να καταφέρνουν πάντα να δίνουν την καλύτερη λύση σε κάθε πείραμα. Τέλος, αξίζει να αναφέρουμε ότι οι λύσεις του ήταν σχεδόν πάντα σταθερές, παρουσιάζοντας πάρα πολύ μικρή μεταβλητότητα.

8. Βιβλιογραφία.

[1] Travelling salesman problem

[https://en.wikipedia.org/wiki/Travelling_salesman_problem ,ανακτήθηκε 25/7/2023]

[2] Vehicle routing problem

αφού κατάφερε να λύσει τα 4 από τα 5 πειράματα και να παράγει αρκετά καλές λύσεις.

7.1.3. Συμπεράσματα για τον αλγόριθμο Local Cheapest Arc.

Ο αλγόριθμος Local Cheapest Arc είχε την χειρότερη επίδοση σε αυτήν την μελέτη. Στην μελέτη για τα φορτηγά δεν κατάφερε να λύσει κανένα πείραμα, ενώ στην μελέτη για τα χρονοπαράθυρα κατάφερε να λύσει μόνο το πείραμα χωρίς χρονοπαράθυρα, στο οποίο παραδόξως έδωσε συνολικά τα καλύτερα αποτελέσματα σε σχέση με τους υπόλοιπους 4 αλγορίθμους που κατάφεραν να λύσουν το πρόβλημα. Λίγο καλύτερη ήταν η απόδοση του στην μελέτη για τους πελάτες, καθώς έλυσε 4 από τα 5 πειράματα, όπου όσο αυξάναμε τους πελάτες, τόσο χειρότερη γινόταν η λύση του σε σχέση με τους υπόλοιπους αλγορίθμους. Αξίζει επίσης να αναφέρουμε ότι, όσο πιο μεγάλος ήταν ο αριθμός των πελατών, τόσο μεγαλύτερη μεταβλητότητα παρουσίαζε στις λύσεις του σε κάθε πείραμα. Τέλος, να επισημάνουμε ότι παρ΄όλο τις κακές λύσεις που παρουσίασε, όταν συνδυαζόταν με το metaheuristic Guided Local Search κατάφερνε σε όλα τα πειράματα να φτάσει το επίπεδο της καλύτερης λύσης κάθε φορά.

7.1.4. Συμπεράσματα για τον αλγόριθμο Global Cheapest Insertion.

Ο αλγόριθμος Global Cheapest Insertion παρήγαγε αρκετά καλά αποτελέσματα σε αυτήν την μελέτη. Συγκεκριμένα, κατάφερε να λύσει όλα τα πειράματα στις μελέτες για τα φορτηγά και τους πελάτες, ενώ δυσκολεύτηκε αρκετά στην μελέτη για τα χρονοπαράθυρα, αφού έλυσε μόνο τα 2 από τα 5 πειράματα. Αξίζει να επισημάνουμε ότι οι λύσεις που κατάφερε να δώσει σε κάθε πείραμα ήταν στην πλειοψηφία τους οι χειρότερες του πειράματος, καθώς επίσης παρουσίαζαν μεγάλη μεταβλητότητα μεταξύ τους, πράγμα που δείχνει ότι η λύση του αλγορίθμου Global Cheapest Insertion επηρεάζετε σημαντικά από το metaheuristic με το οποίο συνδυάζετε. Τέλος, παρόλο τα κακά αποτελέσματα που παρουσίασε στην πλειοψηφία του, όταν συνδυαζόταν με metaheuristic Guided Local Search κατάφερνε

και αυτός σε όλα τα πειράματα να φτάσει το επίπεδο της καλύτερης λύσης κάθε φορά.

7.1.5. Συμπεράσματα για τον αλγόριθμο Parallel Cheapest Insertion.

Κακή ήταν απόδοση του αλγορίθμου Parallel Cheapest Insertion, αφού στην μελέτη για τα χρονοπαράθυρα δεν κατάφερε να λύσει κανένα πείραμα, στην μελέτη για τα φορτηγά κατάφερε να λύσει τα 3 από τα 5 πειράματα, ενώ καλύτερη ήταν η απόδοση του στην μελέτη για τους πελάτες αφού κατάφερε να λύσει 4 από τα 5 πειράματα. Οι λύσεις που παρήγαγε σε κάθε πείραμα ήταν μέτριες σε σχέση με τους υπόλοιπους αλγορίθμους, ενώ σε συνδυασμό με τον Guided Local Search κατάφερνε να φτάσει το επίπεδο της καλύτερης λύσης.

7.1.6. Συμπεράσματα για τον αλγόριθμο Path Cheapest Arc.

Ο αλγόριθμος Path Cheapest Arc ήταν ο αλγόριθμος με την καλύτερη επίδοση σε αυτήν την μελέτη, αφού είναι ο μόνος που κατάφερε να λύσει όλα τα πειράματα και για τις 3 μελέτες. Παρόλα αυτά, οι λύσεις που παρήγαγε ήταν μέτριες σε σχέση με τους υπόλοιπους αλγορίθμους, ενώ όσο πιο δύσκολο γινόταν το πρόβλημα κάθε φορά, τόσο μεγαλύτερες διαφορές είχε στις λύσεις του σε κάθε πείραμα. Τέλος, αξίζει να αναφέρουμε ότι και αυτός ο αλγόριθμος με την βοήθεια του Guided Local Search κατάφερνε να δώσει την καλύτερη λύση στο πείραμα.

7.1.7. Συμπεράσματα για τον αλγόριθμο Path Most Constraint Arc.

Την δεύτερη καλύτερη επίδοση σε αυτήν την μελέτη την είχε ο αλγόριθμος Path Most Constraint Arc. Κατάφερε να λύσει όλα τα πειράματα στην μελέτη των πελατών και των φορτηγών, ενώ στην μελέτη για τα χρονοπαράθυρα κατάφερε να δώσει λύση στα 3 από τα 5 πειράματα. Οι λύσεις που πρόσφερε ήταν μέτριες, ενώ σε 2 πειράματα έδωσε την χειρότερη λύση στο πείραμα, παρόλα αυτά σε συνδυασμό με

τον Guided Local Search κατάφερνε και αυτός ο αλγόριθμος να φτάσει το επίπεδο της καλύτερης λύσης.

7.1.8 Συμπεράσματα για τον αλγόριθμο Savings.

Μέτρια ήταν η απόδοση του αλγορίθμου Savings, αφού κατάφερε να δώσει λύση σε 3 από τα 5 πειράματα στην μελέτη των φορτηγών και των πελατών, ενώ στην μελέτη για τα χρονοπαράθυρα απέτυχε να δώσει λύση και στα 5 πειράματα. Στα πειράματα που κατάφερε να λύσει, οι λύσεις του ήταν από τις καλύτερες στο κάθε πείραμα, ενώ τις περισσότερες φορές ήταν αρκετά σταθερός στις λύσεις που έδινε.

7.2. Συμπεράσματα για τα metaheuristic.

7.2.1 Συμπεράσματα για το metaheuristic Generic Tabu Search.

Την χειρότερη επίδοση σε αυτή την μελέτη την είχε το metaheuristic Generic Tabu Search, αφού με οποίον αλγόριθμο που συνδυάστηκε έδινε πάντα τα χειρότερα αποτελέσματα σε κάθε πείραμα.

7.2.2 Συμπεράσματα για το metaheuristic Greedy Descent.

Το ίδιο κακά αποτελέσματα με το metaheuristic Generic Tabu Search παρήγαγε και το metaheuristic Greedy Descent, αφού έδιναν σε όλα τα πειράματα ακριβώς την ίδια λύση, με τη μόνη διαφορά να είναι ότι ο Greedy Descent είναι πιο γρήγορος όχι μόνο από το Generic Tabu Search, αλλά και από όλα τα metaheuristic που μελετήσαμε.

[https://en.wikipedia.org/wiki/Vehicle_routing_problem ,ανακτήθηκε 25/7/2023]

[3] Xiao, Y., Zhao, Q., Kaku, I., & Xu, Y. (2012). Development of a fuel consumption optimization model for the capacitated vehicle routing problem. *Computers & operations research*, *39*(7), 1419-1431.

[4] Λιγνός, Γ. Δ. (2012). *Επίλυση προβλημάτων δρομολόγησης φορτηγών οχημάτων με χρονικά παράθυρα και χωρητικότητα (VRPTW)* (Bachelor's thesis).

[5] Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M. M., & Soumis, F. (2002). VRP with Pickup and Delivery. *The vehicle routing problem*, *9*, 225-242.

[6] Despaux, F., & Basterrech, S. (2016). Multi-trip vehicle routing problem with time windows and heterogeneous fleet. *International Journal of Computer Information Systems and Industrial Management Applications*, *8*, 355-363.

[7] Larsen, A., Madsen, O. B., & Solomon, M. M. (2008). Recent developments in dynamic vehicle routing systems. *The vehicle routing problem: Latest advances and new challenges*, 199-218.

[8] Gamayanti, N., Alkaff, A., & Mangatas, R. (2015). Optimisasi Multi Depot Vehicle Routing Problem (MDVRP) dengan Variabel Travel Time Menggunakan Algoritma Particle Swarm Optimization. *JAVA Journal of Electrical and Electronics Engineering*, *13*(1).

[9] Bastian, C., & Kan, A. H. R. (1992). The stochastic vehicle routing problem revisited. *European Journal of Operational Research*, *56*(3), 407-412.

[10] Francis, P., & Smilowitz, K. (2006). Modeling techniques for periodic vehicle routing problems. *Transportation Research Part B: Methodological*, *40*(10), 872-884.

[11] Lysgaard, J. (1997). Clarke & Wright's savings algorithm. *Department of Management Science and Logistics, The Aarhus School of Business*, *44*, 1-7.

[12] Di Bartolomeo, M., Grande, E., Nicosia, G., & Pacifici, A. (2017). Cheapest paths in dynamic networks. *Networks*, *69*(1), 23-32.

[13] Safrina, L., & Kusuma, F. (2020). OPTIMIZATION OF THE DISTRIBUTION OF FISH USING THE METHOD OF THE CHEAPEST INSERTION HEURISTIC AT THE PLACE OF FISH

AUCTION IN PANTAI LABU. *Journal of Mathematics and Scientific Computing With Applications*, *1*(2), 60-68.

[14] Abdirad, M., Krishnan, K., & Gupta, D. (2021). A two-stage metaheuristic algorithm for the dynamic vehicle routing problem in Industry 4.0 approach. *Journal of Management Analytics*, *8*(1), 69-83.

[15] Kindervater, G. A., Lenstra, J. K., & Shmoys, D. B. (1989). The parallel complexity of TSP heuristics. *Journal of Algorithms*, *10*(2), 249-270.

[16] Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, *20*, 255-282.

[17] [https://developers.google.com/optimization/routing/routing_options , ανακτήθηκε 10/10/2023]

[18] Meta-heuristics: several previous problems

[https://acrogenesis.com/or-tools/documentation/user_manual/manual/metaheuristics.html , ανακτήθηκε 22/10/2023]

[19] Sicilia, J. A., Quemada, C., Royo, B., & Escuin, D. (2016). An optimization algorithm for solving the rich vehicle routing problem based on Variable Neighborhood Search and Tabu Search metaheuristics. *Journal of Computational and Applied Mathematics*, *291*, 468-477.

[20] Simulated Annealing Explained

[https://www.baeldung.com/cs/simulated-annealing ,ανακτήθηκε 30/10/2023]

[21] Il'ev, V. P. (2001). An approximation guarantee of the greedy descent algorithm for minimizing a supermodular set function. *Discrete Applied Mathematics*, *114*(1-3), 131-146.

[22] Voudouris, C., & Tsang, E. (1999). Guided local search and its application to the traveling salesman problem. *European journal of operational research*, *113*(2), 469-499.

[23] Kallab, C., Haddad, S., El-Zakhem, I., Sayah, J., Chakroun, M., Turkey, N., ... & Shakir, W. (2022). Generic Tabu Search. *Journal of Software Engineering and Applications*, *15*(7), 262-273.