

Σχολή Θετικών Επιστημών και Τεχνολογίας
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα

Διπλωματική Εργασία

Εφαρμογή σύγχρονων τεχνικών υπολογιστικής νοημοσύνης
βασισμένες στον Pelican Optimization Algorithm για την
αποδοτική επίλυση του Inventory Routing Problem

Δερδεμέζης Μάριος

Επιβλέπων καθηγητής: Γρηγόριος Μπεληγιάννης

Πάτρα, Σεπτέμβριος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή (Δερδεμέζης Μάριος) που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.



Εφαρμογή σύγχρονων τεχνικών υπολογιστικής νοημοσύνης
βασισμένες στον Pelican Optimization Algorithm για την
αποδοτική επίλυση του Inventory Routing Problem

Δερδεμέζης Μάριος

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Γρηγόριος Μπεληγιάννης

Καθηγητής Πανεπιστημίου Πατρών

Συν-Επιβλέπων Καθηγητής:

Δημήτριος Καραπιέρης

Καθηγητής Διεθνούς Πανεπιστημίου της
Ελλάδος

Πάτρα, Σεπτέμβριος 2026

Περίληψη

Το Πρόβλημα Δρομολόγησης Αποθέματος (Inventory Routing Problem - IRP) συνιστά μια από τις πλέον σύνθετες προκλήσεις στη διαχείριση της σύγχρονης εφοδιαστικής αλυσίδας, ενοποιώντας τον έλεγχο των αποθεμάτων με τη δρομολόγηση του στόλου οχημάτων. Λόγω της NP-hard φύσης του, οι υφιστάμενες μέθοδοι βελτίωσης αντιμετωπίζουν σοβαρούς υπολογιστικούς περιορισμούς, αδυνατώντας συχνά να επιλύσουν δίκτυα μεσαίας και μεγάλης κλίμακας σε πρακτικούς χρόνους.

Η παρούσα διπλωματική εργασία προτείνει την εφαρμογή ενός σύγχρονου μετα-ευρετικού αλγορίθμου, του Pelican Optimization Algorithm (POA), για την αποδοτική επίλυση του IRP Πολλαπλών Οχημάτων. Προκειμένου να γεφυρωθεί ο συνεχής χώρος αναζήτησης του αλγορίθμου με τη διακριτή φύση του προβλήματος, δοκιμάστηκε ένας τρόπος εφαρμογής μέσω εικονικών κόμβων. Η συνάρτηση καταλληλότητας ενσωματώνει μηχανισμούς όπως τον αλγόριθμο τοπικής αναζήτησης 2-opt για την ταχεία δρομολόγηση, την λογική φόρτωσης οχημάτων, τη στρατηγική του κυλιόμενου ορίζοντα για την πρόβλεψη μελλοντικής ζήτησης, καθώς και δυναμικές ποινές για τη διαχείριση των ελλείψεων μέσω της στοχευμένης χαλάρωσης των περιορισμών που επιβάλλει το πρόβλημα.

Η πειραματική αξιολόγηση πραγματοποιήθηκε στο ευρέως διαδεδομένο σύνολο δεδομένων αναφοράς (instances) που υπάρχουν διαθέσιμα στην διεθνή βιβλιογραφία σε δίκτυα 10, 25 και 50 πελατών. Τα υπολογιστικά αποτελέσματα καταδεικνύουν την εξαιρετική ταχύτητα και ομοιογένεια του POA. Σε αντίθεση με τις υπόλοιπες μεθόδους επίλυσης (Branch & Cut, Matheuristic) οι οποίες τερματίζουν λόγω εξάντλησης του χρονικού ορίου εκτέλεσης που έχουν στην διάθεση τους, ο POA εξάγει λύσεις σε χαμηλότερα χρονικά όρια.

Αν και τα παραγόμενα αποτελέσματα που προκύπτουν παράγουν σχετικά μεγάλο ποσοστό χάσματος στο τελικό κόστους έναντι των υπολοίπων μεθόδων, η ταχύτητα εξαγωγής και η ικανότητα εύρεσης λύσεων ίσως να καθιστά τον αλγόριθμο πρακτικά εφαρμόσιμο.

Λέξεις – Κλειδιά

Πρόβλημα Δρομολόγησης Αποθεμάτων, Pelican Optimization Algorithm, Multi Vehicle IRP.

Abstract

Application of modern artificial intelligence techniques based on the Pelican Optimization Algorithm for the efficient solution of the Inventory Routing Problem

The Inventory Routing Problem (IRP) is one of the most complex challenges in modern supply chain management, combining inventory control with vehicle fleet routing. Due to its NP-hard nature, existing optimization methods face serious computational constraints, often failing to solve medium- and large-scale networks within a practical timeframe.

This thesis proposes the application of a modern meta-heuristic algorithm, the Pelican Optimization Algorithm (POA), for the efficient solution of the Multi-Vehicle IRP. In order to bridge the algorithm's continuous search space with the discrete nature of the problem, an implementation method using virtual nodes was tested. The fitness function incorporates mechanisms such as the 2-opt local search algorithm for rapid routing, vehicle loading logic, the rolling horizon strategy for forecasting future demand, as well as dynamic penalties for managing shortages through the targeted relaxation of the constraints imposed by the problem.

The experimental evaluation was conducted on the widely used reference dataset (instances) available in the international literature for networks with 10, 25, and 50 clients. The computational results demonstrate the exceptional speed and consistency of POA. Unlike other solution methods (Branch & Cut, Matheuristic), which terminate due to exhaustion of their allotted execution time limit, POA produces solutions within shorter time limits.

Although the results obtained show a relatively large cost gap compared to other methods, the speed of computation and the ability to find solutions may make the algorithm practically applicable.

Keywords

Inventory Routing Problem, Pelican Optimization Algorithm, Multi Vehicle IRP.

Περιεχόμενα

Περίληψη.....	5
Abstract	6
Περιεχόμενα	7
Κατάλογος Εικόνων / Σχημάτων	9
Κατάλογος Πινάκων	10
Συντομογραφίες & Ακρωνύμια.....	11
Εισαγωγή.....	1
1. Το Inventory Routing Problem	3
1.1 Η απλή λύση.....	4
1.1 Η καλύτερη ή βέλτιστη λύση	5
1.3 Διαχείριση δικτύων διανομής.....	6
1.4 Χρόνοι παράδοσης και χρονικός ορίζοντας.....	6
1.5 Δομή της πολιτικής διανομής.....	7
1.6 Στόχος της πολιτικής διανομής	8
1.7 Αποφάσεις δρομολόγησης	8
1.7.1 Αποφάσεις ως προς τον χρόνο: Το πρόβλημα αποστολής από έναν προμηθευτή σε ένα πελάτη (Single Link Shipping Problem).....	9
1.7.2 Αποφάσεις ως προς τον χρόνο και τον χώρο (Decisions over time and space)	12
1.7.2.1 Το IRP της οχήματος (single-vehicle IRP)	12
1.7.2.2 Το IRP με στόλο οχημάτων (multi-vehicle IRP)	15
2. Ο Pelican Optimization Algorithm και η Συνάρτηση Καταλληλότητας (Fitness Function)	17
2.1 Pelican Optimization Algorithm	17
2.2 Το μαθηματικό μοντέλο του POA	17
2.3 Η Φάση Εξερεύνησης (Φάση 1)	19
2.4 Φάση Εκμετάλλευσης (Φάση 2)	20
2.5 Ψευδοκώδικας POA	20
2.7 Οι Εικονικοί Κόμβοι (Virtual Nodes).....	23
2.8 Ο Αλγόριθμος της Συνάρτησης Καταλληλότητας (Fitness Function).....	24
2.9 Ο ψευδοκώδικας της Συνάρτησης Καταλληλότητας	26
2.10 Χαλάρωση Περιορισμών	27
2.11 Η Στρατηγική του Κυλιόμενου Ορίζοντα	28
3. Επίλυση και Βελτιστοποίηση του IRP	30
3.1 Δεδομένα Εισόδου	30
3.2 Προγραμματιστικό Περιβάλλον και Υλοποίηση στην Python	32
3.3 Τεχνικές Προδιαγραφές Υπολογιστικού Συστήματος	33
3.5 Ρύθμιση Παραμέτρων του POA.....	34
3.6 Συλλογή Στατιστικών Δεδομένων.....	36
3.7 Εμφάνιση Λύσης	36
4. Παρουσίαση και Ανάλυση Αποτελεσμάτων	42

4.2 Αποτελέσματα για τα δίκτυα Μεσαίας Κλίμακας ($N=25$)	47
4.3 Αποτελέσματα για τα δίκτυα Μεγάλης Κλίμακας ($N \geq 50$).....	50
5. Συγκριτική Αξιολόγηση	53
5.1 Σύγκριση με τον αλγόριθμο Branch & Cut.....	53
5.2 Σύγκριση με τον αλγόριθμο Matheuristic	55
6. Συμπεράσματα και Προτάσεις	56
6.1 Συμπεράσματα	56
6.2 Προτάσεις για μελλοντική έρευνα και εφαρμογές.....	56
Βιβλιογραφία.....	58
Παράρτημα.....	61
Συλλογή Στοιχείων από τα instances (load_data.py)	61
Υπολογισμός εγγύτητας εικονικών και πραγματικών κόμβων (shapely_utils.py)	63
Η συνάρτηση καταλληλότητας (new_fitness_hard.py)	64
Η κλάση Pelican Optimization Algorithm (Pelican.py).....	67
Η tsp2opt για την δημιουργία βέλτιστων διαδρομών (tsp_2opt.py)	69
Η συνάρτηση που εξάγει την τελική λύση (αναφορά) του instance (new_improved_print_fitness.py)	69
Το κυρίο πρόγραμμα εκτέλεσης (multiprocessed_run_poa.py).....	74

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 1-1: Το παράδειγμα των Bell κ.α.....	4
Εικόνα 1-2: Η απλή λύση.....	5
Εικόνα 3-1: Παράδειγμα περίπτωσης IRP	30
Εικόνα 3-2: Στιγμιότυπο Αναφοράς από το αρχείο με $N=25$, $Q=38$, $T=6$: Ημέρα 1 ^η (Αγροτική και Τυχαία Κατανομή)	38
Εικόνα 3-3: Στιγμιότυπο Αναφοράς από το αρχείο με $N=25$, $Q=38$, $T=6$: Ημέρα 6η (Αγροτική και Τυχαία Κατανομή)	39
Εικόνα 3-4: Στιγμιότυπο Αναφοράς από το αρχείο με $N=25$, $Q=38$, $T=6$: Ημέρα 3η (Αγροτική και Ομαδοποιημένη Κατανομή), με την προειδοποιητική ένδειξη έλλειψης ...	40
Εικόνα 3-5: Διάγραμμα σύγκλισης από το αρχείο με $N=25$, $Q=38$, $T=6$ (Αγροτική και Τυχαία Κατανομή). Οι καμπύλες απεικονίζουν 10 διαδοχικές εκτελέσεις, όπου αυτή που παρουσιάζει το καλύτερο κόστος επισημαίνεται με κόκκινο χρώμα.....	41

Κατάλογος Πινάκων

Πίνακας 3-1: Τα χαρακτηριστικά γνωρίσματα των παραδειγμάτων IRP	32
Πίνακας 4-1: Αποτελέσματα για Δίκτυα Μικρής Κλίμακας (N=10) - Μέρος 1ο	44
Πίνακας 4-2: Αποτελέσματα για Δίκτυα Μικρής Κλίμακας (N=10) - Μέρος 2ο	45
Πίνακας 5-1: Συγκριτική Αξιολόγηση Επιδόσεων για Αντιπροσωπευτικά Δίκτυα (N=10) - Μέρος 1ο	53
Πίνακας 5-2: Συγκριτική Αξιολόγηση Επιδόσεων για Αντιπροσωπευτικά Δίκτυα (N=10) – Μέρος 2ο	54
Πίνακας 5-3: Συγκριτική Αξιολόγηση Επιδόσεων για Αντιπροσωπευτικά Δίκτυα (N=25)	54
Πίνακας 5-4: Ενδεικτική Σύγκριση Επιδόσεων έναντι της Matheuristic	55

Συντομογραφίες & Ακρωνύμια

IRP	Inventory Routing Problem
POA	Pelican Oprimization Algorithm
TSP	Travel Salesman Problem
CPU	Central Processing Unit
ML	Maximum Level Policy

Εισαγωγή

Η διαχείριση της σύγχρονης εφοδιαστικής αλυσίδας απαιτεί την ταυτόχρονη λήψη αποφάσεων σε πολλαπλά επίπεδα. Το Πρόβλημα Δρομολόγησης Αποθέματος (Inventory Routing Problem - IRP) συνιστά ένα από τα πλέον σύνθετα πεδία της επιχειρησιακής έρευνας, καθώς ενοποιεί τον προγραμματισμό των αποθεμάτων με τη δρομολόγηση του στόλου οχημάτων, αποσκοπώντας στην ελαχιστοποίηση του συνολικού κόστους διανομής δίχως να προκληθούν ελλείψεις (stockouts) στους τελικούς πελάτες.

Το πρόβλημα IRP ανήκει στην κατηγορία των NP-Hard προβλημάτων, που σημαίνει ότι ο χώρος των πιθανών λύσεων κινείται σε χώρους αναζήτησης μεγάλης έκτασης. Οι περισσότερες μέθοδοι επίλυσης είναι μεν ικανές να εντοπίσουν βέλτιστες λύσεις αλλά απαιτούν σημαντικά μεγάλο χρόνο επεξεργασίας ακόμα και σε μικρής κλίμακας προβλήματα δρομολόγησης.

Ο σκοπός εκπόνησης αυτής της εργασίας είναι να εξετάσει αν ο Pelican Optimization Algorithm μπορεί να λύσει αποδοτικά σε χρόνο και με το ελάχιστο δυνατό κόστος το πρόβλημα IRP, μέσω της πολιτικής διανομής που προτείνει. Η ικανότητα του στην εύρεση και βελτιστοποίηση τιμών συναρτήσεων, αξιοποιείται και προσαρμόζεται κατάλληλα στους περιορισμούς που θέτουν το πρόβλημα αυτό. Μέσω μιας γενικής ανασκόπησης της βιβλιογραφίας και αξιοποίησης διαφόρων τεχνικών επίλυσης, ο αλγόριθμος εξοπλίζεται με τις κατάλληλες παραμετροποιήσεις ώστε να αντιμετωπίσει δυναμικά το πρόβλημα δρομολόγησης.

Η παρούσα διπλωματική εργασία αποτελείται από τα εξής μέρη:

Στο 1^ο Κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο του Προβλήματος Δρομολόγησης Αποθέματος και πραγματοποιείται εκτενής βιβλιογραφική ανασκόπηση των μεθόδων επίλυσης, που στηρίζονται στην ελαχιστοποίηση συναρτήσεων με αυστηρούς περιορισμούς. Στην συνέχεια το 2^ο Κεφάλαιο αναλύεται ο αλγόριθμος Pelican Optimization Algorithm (POA), περιγράφοντας τον τρόπο σύλληψής του, τα μαθηματικά του μοντέλα και τον τρόπο που μεταβαίνει από τη φάση της εξερεύνησης (exploration) στη φάση της εκμετάλλευσης (exploitation) στην εύρεση βέλτιστων λύσεων, ενώ αναλύονται οι τεχνικές προσαρμογής και παραμετροποίησης του για να επιλύσει το πρόβλημα. Στο 3^ο Κεφάλαιο

αναπτύσσεται η μεθοδολογία έρευνας και επίλυσης του προβλήματος προγραμματιστικά και στο 4^ο Κεφάλαιο παρουσιάζονται και αναλύονται τα αποτελέσματα της έρευνας αυτής. Στο 5^ο Κεφάλαιο γίνεται μια συγκριτική αξιολόγηση των αποτελεσμάτων με τα ήδη υφιστάμενα αποτελέσματα άλλων αλγορίθμων και τέλος στο 6^ο Κεφάλαιο αναπτύσσονται γενικά κάποιες προτάσεις για τις μελλοντικές προεκτάσεις και βελτιώσεις της προτεινόμενης επίλυσης.

1. Το Inventory Routing Problem

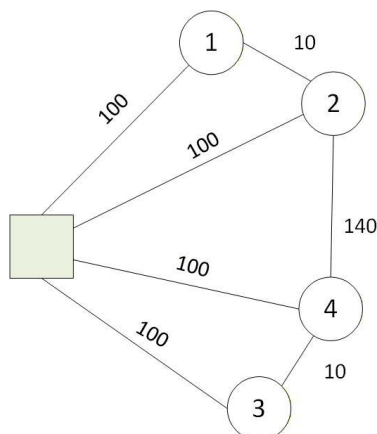
Το Πρόβλημα Δρομολόγησης Αποθέματος ή Inventory Routing Problem (IRP), είναι ο τρόπος βελτιστοποίησης της διανομής ενός συνόλου προϊόντων από τον προμηθευτή προς τους πελάτες μέσω δρομολόγησης οχημάτων. Στόχος της είναι να περιοριστούν στο ελάχιστο το κόστος μεταφοράς των προϊόντων αλλά και το κόστος διατήρησης αποθέματος των προϊόντων αυτών στον εκάστοτε πελάτη ενώ παράλληλα θα πρέπει να εξασφαλιστεί ότι το απόθεμα αυτό δεν θα εξαντληθεί.

Οι κύριοι περιορισμοί της διανομής της είναι [5]:

1. Ποιοι θα πελάτες θα εξυπηρετηθούν,
2. Πόσα προϊόντα θα μεταφερθούν και
3. Ποιες διαδρομές θα ακολουθήσει η διανομή.

Ας δούμε ένα απλό παράδειγμα που αναφέρεται στον Bell [2] και περιγράφει τον αντίκτυπο που θα έχει η οποιαδήποτε απόφαση διανομής, λαμβάνοντας υπόψη τους παραπάνω περιορισμούς:

Έστω ότι θέλουμε να μεταφέρουμε ένα προϊόν σε ένα σύνολο πελατών μέσω οχημάτων με δεδομένη χωρητικότητα και σε συγκεκριμένες ημέρες. Θεωρούμε ότι το πλήθος των πελατών που θέλουμε να εξυπηρετήσουμε είναι 4, ο αριθμός των οχημάτων που διαθέτουμε είναι απερίοριστος ενώ το κόστος διατήρησης αποθέματος είναι αμελητέο. Στο παρακάτω γράφημα (Εικόνα 1) παρουσιάζονται οι διαθέσιμες διαδρομές μεταξύ προμηθευτή και πελατών με τα αντίστοιχα κόστη μεταφοράς για κάθε διαδρομή. Η μέγιστη χωρητικότητα των οχημάτων που διαθέτουμε είναι 5000 μονάδες. Η ημερήσια ζήτηση για κάθε πελάτη είναι 1000, 3000, 2000 και 1500 μονάδες, ενώ αντίστοιχα το μέγιστο όριο αποθέματος για τον καθέναν είναι 5000, 3000, 2000 και 4000 μονάδες. Θεωρούμε ότι το αρχικό απόθεμα του κάθε πελάτη είναι ίσο με το μέγιστο όριο αποθέματός του.



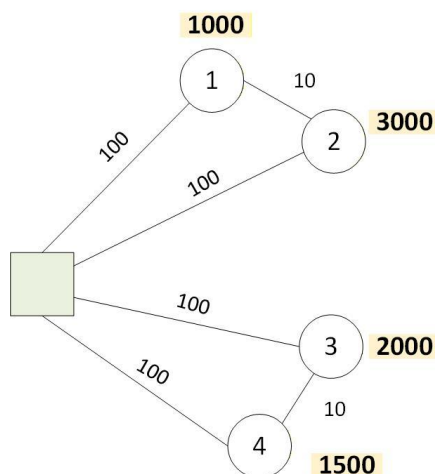
Εικόνα 1-1: Το παράδειγμα των Bell κ.α.

Σκοπός μας τώρα είναι να καταστρώσουμε μια πολιτική διανομής ανά ημέρα όπου λαμβάνονται υπόψη το ποιος πελάτης θα εξυπηρετηθεί, πόσες μονάδες προϊόντων θα μεταφερθούν και ποιες διαδρομές θα ακολουθήσουν τα οχήματα, ενώ η διανομή αυτή θα πρέπει να επαναλαμβάνεται συχνά και με τέτοιο τρόπο ώστε το κόστος μεταφοράς να είναι το ελάχιστο δυνατό. Η πολιτική αυτή θα φροντίζει ώστε να μην προκληθεί έλλειψη αποθέματος στον πελάτη, να μην ξεπερνιέται το μέγιστο όριο αποθέματός του και τα οχήματα θα πρέπει να ξεκινούν από τον προμηθευτή με πλήρες φορτίο. Στο τέλος της ημερήσιας διανομής θα πρέπει να έχει εξασφαλιστεί ότι το απόθεμα του κάθε πελάτη θα είναι ίσο με το αρχικό απόθεμά του.

Για την επίλυση του προβλήματος αυτού προτείνονται δύο λύσεις: Η απλή και η βέλτιστη.

1.1 Η απλή λύση

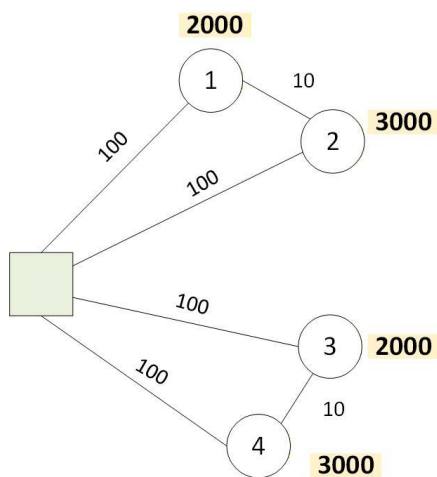
Με βάση τα δεδομένα που διαθέτουμε, παρατηρούμε ότι οι πελάτες 2 και 3 θα πρέπει να εξυπηρετούνται κάθε ημέρα καθώς το μέγιστο απόθεμά τους είναι ίσο με την ημερήσια ζήτηση του προϊόντος, γεγονός το οποίο σημαίνει οι συγκεκριμένοι πελάτες δεν μπορούν να διατηρήσουν περισσότερο απόθεμα από αυτό που πουλάνε. Λαμβάνοντας υπόψη τα κόστη μεταφοράς, η λύση είναι να εξυπηρετηθούν ξεχωριστά τα κοντινότερα ζεύγη πελάτων που είναι τα (1, 2) και (3, 4) με 2 οχήματα αντίστοιχα. Αυτή η πολιτική διανομής μπορεί να επαναλαμβάνεται ημερησίως, ενώ τα οχήματα που εκτελούν το δρομολόγιο της είναι πλήρη σε φορτίο και το κόστος της ανέρχεται σε 420/ημέρα.



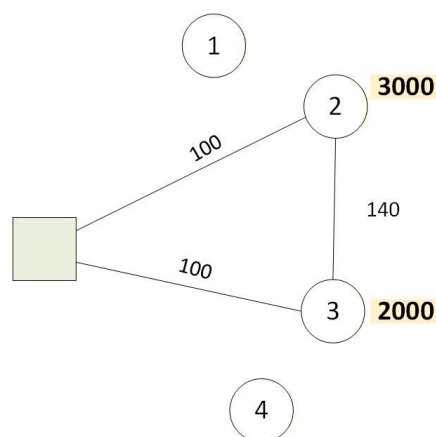
Εικόνα 1-2: Η απλή λύση.

1.1 Η καλύτερη ή βέλτιστη λύση

Μια καλύτερη πρόταση είναι η παραπάνω πολιτική διανομής να υλοποιηθεί σε 2 μέρες. Την πρώτη μέρα μπορεί να εξυπηρετηθεί το ζεύγος (1, 2) με ένα όχημα και το ζεύγος (3, 4) με άλλο όχημα. Οι ποσότητες που θα διανεμηθούν αντίστοιχα είναι 2000 μονάδες (διπλάσια ποσότητα από την ημερήσια ζήτηση) για τον 1, 3000 (ποσότητα ίση με την ημερήσια ζήτηση) για τον 2, 2000 (όσο και η ημερήσια ζήτηση) για τον 3 και 3000 (διπλάσια ποσότητα από την ημερήσια ζήτηση) για τον 4. Σε αυτή την περίπτωση τα οχήματα εκτελούν το δρομολόγιο της με πλήρες φορτίο. Επειδή γνωρίζουμε ότι το ζεύγος πελατών (2, 3) θα πρέπει να εξυπηρετείται ημερησίως, θα εξυπηρετηθούν ξανά με ένα όχημα την δεύτερη μέρα με την τροφοδότηση 3000 μονάδων στον 2 και 2000 στον 3, αντίστοιχα. Το



Εικόνα 1- 3: Βέλτιστη Λύση (Ημέρα 1η).



Εικόνα 1-4: Βέλτιστη Λύση (Ημέρα 2η).

μέσο ημερήσιο κόστος της πολιτικής αυτής ανέρχεται σε 380/ημέρα και είναι σαφές ότι είναι καλύτερη ή βέλτιστη.

1.3 Διαχείριση δικτύων διανομής

Ένα δίκτυο διανομής αποτελείται από έναν προμηθευτή (supplier) και έναν ή περισσότερους πελάτες (customers) που συνήθως είναι καταστήματα λιανικής (retailers). Η αφετηρία του δικτύου είναι ο προμηθευτής και οι ενδιάμεσοι σταθμοί είναι οι πελάτες. Ο προμηθευτής μπορεί να είναι συνήθως μία βιομηχανία, μία αποθήκη ή ένας χώρος στάθμευσης οχημάτων.

Ένα σύνολο προϊόντων θα πρέπει να μεταφερθεί από τον προμηθευτή της της πελάτες προκειμένου να ικανοποιηθεί μία ντετερμινιστική ζήτηση σε έναν χρονικό ορίζοντα ο οποίος είναι πεπερασμένος ή άπειρος. Ο προμηθευτής και οι πελάτες συνήθως επιβαρύνονται με το κόστος διατήρησης αποθέματος, με εξαίρεση κάποιες ειδικές περιπτώσεις που επιβαρύνονται ο ένας από τους δύο με το συγκεκριμένο κόστος. Η διανομή των προϊόντων γίνεται από έναν στόλο οχημάτων με συγκεκριμένη χωρητικότητα ενώ απαιτείται και κάποιο κόστος που θα πρέπει να καταβάλλεται για κάθε διαδρομή που εκτελείται.

Τα κύρια χαρακτηριστικά του IRP είναι τα εξής: χρόνοι παράδοσης και ο χρονικός ορίζοντας, η δομή της πολιτικής διανομής, ο στόχος της πολιτικής της και οι αποφάσεις δρομολόγησης. Μία βέλτιστη πολιτική όπου καθορίζεται το καθένα από τα παραπάνω χαρακτηριστικά καθορίζουν και την βελτιστοποίηση του προβλήματος IRP.

1.4 Χρόνοι παράδοσης και χρονικός ορίζοντας

Σε ένα IRP οι πιθανοί χρόνοι παράδοσης μπορεί να είναι:

- Συνεχείς, όπου η παράδοση μπορεί να πραγματοποιηθεί σε οποιοδήποτε χρονική στιγμή (αρχίζοντας από το 0),
- Συνεχείς με ελάχιστο χρόνο διαμεταφοράς, όπου η παράδοση μπορεί να πραγματοποιηθεί οποιαδήποτε χρονική στιγμή αλλά ο χρόνος μεταξύ δύο διαδοχικών παραδόσεων δεν θα πρέπει να είναι κάτω από κάποιο ελάχιστο χρονικό όριο.
- Ξεχωριστές, όπου γίνονται πολλαπλές παραδόσεις με ελάχιστο χρόνο παράδοσης.

Ο χρονικός ορίζοντας στα όρια του οποίου μπορεί να βελτιωθεί ένα IRP μπορεί να είναι:

- Άπειρος, όπου ο χρόνος για την υλοποίηση της σχεδίου διανομής μπορεί να καθοριστεί από παράγοντες όπως ο αριθμός διαθέσιμων οχημάτων, ο αριθμός διαθέσιμων οδών και οι περιοχές που εξυπηρετεί η διανομή.
- Πεπερασμένος, όπου ο χρόνος για την υλοποίηση του σχεδίου διανομής καθορίζεται αναλόγως της απαίτησης διανομής που προκύπτει. Στην περίπτωση αυτή όσο πιο μικρός είναι ο χρόνος του, τόσο πιο λειτουργικό είναι.

1.5 Δομή της πολιτικής διανομής

Μια βέλτιστη πολιτικής διανομής εξαρτάται από τις ώρες εξυπηρέτησης των πελατών, από τις ποσότητες που πρέπει να διανεμηθούν και την διαδρομή που θα πρέπει να ακολουθήσουν τα οχήματα. Ο σχεδιασμός της είναι εξαιρετικά δύσκολος έως αδύνατος αν δεν υπάρχει συγκεκριμένη δομή που θα πρέπει να ακολουθηθεί και η βελτιστοποίηση του IRP συνήθως επιτυγχάνεται εάν ακολουθηθεί συγκεκριμένη δομή σχεδιασμού.

Παρακάτω αναφέρονται οι συγκεκριμένες δομές σχεδιασμού, που προέκυψαν από υφιστάμενες πολιτικές διανομής:

- Παραγγελία χωρίς απόθεμα (Zero Inventory Ordering). Όλοι οι πελάτες προμηθεύονται μόνο όταν το απόθεμα της είναι μηδενικό.
- Περιοδική (Periodic), όπου οι πελάτες προμηθεύονται αγαθά σε συγκεκριμένη περίοδο.
- Με βάση την συχνότητα (Frequency-based), όπου η προμήθεια των αγαθών γίνεται ανά τακτά και σταθερά διαστήματα.
- Πλήρες φορτίου (Full load). Οι διανομές των αγαθών πραγματοποιούνται όταν στα οχήματα έχει καλυφθεί πλήρως η χωρητικότητά της.
- Απευθείας αποστολή (Direct Shipping). Ο κάθε πελάτης εξυπηρετείται απευθείας από τον προμηθευτή και ανεξάρτητα από της υπόλοιπους και χωρίς ενδιάμεσους σταθμούς.
- Επίπεδο αναπλήρωσης (Order-up-to level). Οι πελάτες εξυπηρετούνται μέχρι το μέγιστο επίπεδο αποθέματος που διαθέτει ο καθένας.

- Μέγιστο επίπεδο αναπλήρωσης (Maximum Level). Είναι η γενικευμένη περίπτωση της πολιτικής Επιπέδου Αναπλήρωσης, όπου οι πελάτες δεν εξυπηρετούνται παραπάνω από το μέγιστο επίπεδο αποθέματος που διαθέτουν.
- Εξυπηρέτηση ανά περιοχή (Fixed Partition). Οι πελάτες εξυπηρετούνται ως σύνολο ανά περιοχή και οι διανομές γίνονται αποκλειστικά στην περιοχή αυτή.
- Εξυπηρέτηση με βάση την περιοχή (Partition-based), η οποία είναι μια γενικευμένη περίπτωση της Εξυπηρέτησης ανά περιοχή, όπου οι πελάτες εξυπηρετούνται ως σύνολο ανά περιοχή είτε ως συνδυασμός συνόλων από διαφορετικές περιοχές.

1.6 Στόχος της πολιτικής διανομής

Ο σκοπός της εκάστοτε πολιτικής διανομής εξαρτάται από την απαίτηση που υπάρχει για την ελαχιστοποίηση του κόστους μεταφοράς ή του κόστους διατήρησης αποθέματος.

Στην περίπτωση της ελαχιστοποίησης του κόστους μεταφοράς (Transportation costs only) κύριος στόχος είναι να μειωθούν τα χιλιόμετρα ή τα δρομολόγια. Η καλύτερη λύση σε αυτήν την περίπτωση είναι να εκτελούνται αραιά δρομολόγια παραδόσεων με γεμάτα οχήματα ώστε να μειωθούν τα έξοδα κίνησης. Εάν ο στόχος είναι να μειωθεί το κόστος διατήρησης αποθέματος, τότε εκτελούνται συχνά δρομολόγια με λιγότερο φορτωμένα οχήματα ώστε να διατηρείται χαμηλά το κόστος αποθέματος στον πελάτη παρόλο που σε αυτήν την περίπτωση το κόστος μεταφοράς αυξάνεται.

Υπάρχει ωστόσο μια ολοκληρωμένη προσέγγιση που αφορά την μείωση του συνολικά του κόστους μεταφοράς και του κόστους διατήρησης αποθέματος στον πελάτη και στον προμηθευτή. Η λύση εδώ είναι να μπορεί ο προμηθευτής να διώξει εμπόρευμα προς τους πελάτες ώστε να μπορεί να ελαχιστοποιεί το δικό του κόστος αποθέματος.

1.7 Αποφάσεις δρομολόγησης

Οι Bertazzi & Speranza στο [6] διαχωρίζουν τις αποφάσεις δρομολόγησης στα IRP σε δύο βασικές κατηγορίες ανάλογα με το πόσο πολύπλοκη είναι η δρομολόγηση:

1. **Αποφάσεις ως προς τον χρόνο (Decisions over time only).** Στην περίπτωση αυτή εξετάζεται το πότε (χρόνος) και το πόσο (ποσότητες) θα σταλθούν στον πελάτη μέσω δεδομένων διαδρομών. Στην κατηγορία αυτή ανήκουν οι περιπτώσεις όπου υπάρχει ένας προμηθευτής και ένας πελάτης (Single Link) ή περιπτώσεις απευθείας

αποστολών (Direct Shipping), όπου το όχημα πηγαίνει γεμάτο σε ένα πελάτη και επιστρέφει χωρίς να απαιτείται σχεδιασμός περίπλοκης διαδρομής με επισκέψεις σε διάφορους πελάτες.

2. **Αποφάσεις ως τον χρόνο και τον χώρο (Decisions over time and space).** Οι αποφάσεις λαμβάνονται έχοντας υπόψη το πότε (χρόνος) και από ποια διαδρομή (χώρος) θα ακολουθήσουν τα οχήματα. Η συγκεκριμένη περίπτωση είναι πιο πολύπλοκη καθώς το κάθε όχημα με ένα δρομολόγιο επισκέπτεται πολλούς πελάτες ταυτόχρονα. Επομένως, θα πρέπει να αποφασιστεί η σειρά επίσκεψης των πελατών στην αντίστοιχη ημέρα παράδοσης.

1.7.1 Αποφάσεις ως προς τον χρόνο: Το πρόβλημα αποστολής από έναν προμηθευτή σε ένα πελάτη (Single Link Shipping Problem)

Στην κατηγορία αυτή θα αναλύσουμε της πέντε περιπτώσεις που αποτελούν τα μοντέλα του προβλήματος αποστολής σε έναν πελάτη.

1. Η Συνεχής Περίπτωση (The Continuous Case)

Στην συνεχή περίπτωση η δρομολόγηση αποστολής αποθέματος μπορεί να γίνει οποιαδήποτε στιγμή και στόχος της είναι να βρούμε μια βέλτιστη περιοδική πολιτική αποστολής για αυτό. Η πολιτική αυτή είναι να σταλούν τα προϊόντα στον πελάτη μία φορά και με ένα όχημα με το ελάχιστο δυνατό κόστος.

Το ελάχιστο κόστος του αποθέματος και μεταφοράς προσδιορίζεται από την συνάρτηση:

$$\min h\tau + \frac{c}{\tau} \quad (1)$$

όπου σε ένα σύνολο I με i προϊόντα, το $h = \sum_{i \in I} h_i \cdot q_i$ αφορά το κόστος αποθέματος ανά μονάδα χρόνου και q η ζήτηση του προϊόντος από τον πελάτη, τ είναι η μονάδα χρόνου ή διάρκεια της περιόδου και c το κόστος μεταφοράς. Οι περιορισμοί που υπάρχουν ώστε να υφίσταται το ελάχιστο κόστος είναι $v\tau \leq 1$ και $\tau \geq 0$, όπου $v = \sum_{i \in I} v_i \cdot q_i$ είναι v είναι η ποσότητα του προϊόντος που διαθέτει ο προμηθευτής ανά διάρκεια περιόδου ενώ το 1 είναι το πλήθος των οχημάτων.

2. Η συνεχή περίπτωση με ελάχιστο χρόνο διαμεταφοράς (The continuous case with minimum intershipment time)

Αυτή είναι μια πιο εξειδικευμένη περίπτωση της συνεχούς περίπτωσης, όπου τα προϊόντα στέλνονται με διαδοχικές αποστολές και μεταξύ των αποστολών αυτών παρεμβάλλεται ένα χρονικό διάστημα (διαμεταφορά). Στην περίπτωση αυτή για να έχουμε ελάχιστο δυνατό κόστος, θα πρέπει να ελαχιστοποιηθεί αυτό το χρονικό διάστημα που επιτυγχάνεται με το να γίνονται οι αποστολές με περισσότερα του ενός οχήματα ανά μονάδα χρόνου.

Η αντικειμενική συνάρτηση που εκφράζει το ελάχιστο κόστος είναι η εξής:

$$\min h\tau + \frac{c}{\tau}y \quad (2)$$

όπου η επιπλέον μεταβλητή y εκφράζει τον αριθμό των διαθέσιμων οχημάτων. Οι περιορισμοί αυτήν την φορά είναι ότι $v\tau \leq y$, $\tau \geq 0$ και $y \geq 1$.

3. Η διακριτή περίπτωση με δεδομένες συχνότητες (*The discrete case with given frequencies*)

Αυτή η περίπτωση είναι πολύ κοντά στην πραγματικότητα που επικρατεί στις μεταφορές. Οι υπηρεσίες μεταφοράς παρέχονται με ημερήσια ή εβδομαδιαία ή μηνιαία συχνότητας δρομολόγια. Η ελαχιστοποίηση του κόστους επιτυγχάνεται με την εύρεση της συχνότητας ή των συχνοτήτων που θα εκτελούνται τα δρομολόγια.

Από το σύνολο των μελετών που έχουν δημοσιευθεί [4, 5, 27, 28] η αντικειμενική συνάρτηση που προσδιορίζει το ελάχιστο κόστος στο σύνολο του μέσου κόστους μεταφοράς και αποθέματος ανά μονάδα χρόνου είναι η εξής:

$$\min \sum_{i \in I} \sum_{j \in J} h_i q_i t_j x_{ij} + \sum_j \frac{c}{t_j} y_j \quad (3)$$

Όπου t_j είναι ο χρόνος διαμεταφοράς που ισούται με $1/f_j$ ($t_j = 1/f_j$) με την μεταβλητή f_j να εκφράζει την συχνότητα εκτέλεσης των δρομολογίων από ένα σύνολο ημερήσιων, μηνιαίων κοκ. δρομολογίων, χ_{ij} είναι το ποσοστό των προϊόντων που πρέπει να μεταφερθούν στην συγκεκριμένη συχνότητα και y_j είναι ο αριθμός των οχημάτων που χρησιμοποιούνται στην συγκεκριμένη συχνότητα. Οι περιορισμοί που ισχύουν για αυτήν συνάρτηση είναι $\sum_{i \in I} \chi_{ij} = 1$, $\sum_{i \in I} h_i q_i t_j x_{ij} \leq y_j$, με $\chi_{ij} \geq 0$ και y_j ακέραιος ≥ 0 με j να αφορά την συχνότητα εκτέλεσης δρομολογίων.

4. Η διακριτή περίπτωση με πεπερασμένο χρονικό ορίζοντα (The discrete case with finite time horizon)

Η συγκεκριμένη περίπτωση είναι η πιο ρεαλιστική περίπτωση στις μεταφορές και αφορά τον προγραμματισμό παραδόσεων ακολουθώντας ένα πλάνο H περιόδων (όπου κάθε περίοδος μπορεί να είναι ένα σύνολο κάποιων ημερών πχ. 7 ημέρες). Σκοπός σε αυτήν την περίπτωση είναι να ελαχιστοποιηθεί το κόστος αποθέματος και μεταφοράς με τον προσδιορισμό της ποσότητας των προϊόντων που θα μεταφέρεται κάθε φορά από τον προμηθευτή προς τον πελάτη και πόσα οχήματα θα χρησιμοποιηθούν την χρονική στιγμή που θα γίνει η μεταφορά. Η ελαχιστοποίηση αυτή επιτυγχάνεται με την χρήση της παρακάτω συνάρτησης:

$$\min \sum_{i \in I} h_i (d_i^A + d_i^B) + \sum_{t \in T} \frac{c}{H} y_t \quad (4)$$

Όπου d_i^A είναι η ποσότητα που είναι διαθέσιμη στον προμηθευτή, d_i^B είναι η ποσότητα που πρέπει να είναι διαθέσιμη στον πελάτη, H είναι το πλάνο περιόδων και y_t είναι ο αριθμός των οχημάτων που χρησιμοποιούνται την t χρονική στιγμή.

Οι περιορισμοί που ισχύουν είναι οι εξής: $\min \sum_{t \in T} s_{it} = q_i H$, όπου s_{it} είναι η ποσότητα των προϊόντων που μεταφέρονται από τον προμηθευτή στον πελάτη κάθε χρονική στιγμή t και $\min \sum_{t \in T} v_i s_{it} \leq y_t$, όπου εξασφαλίζεται ότι ο αριθμός των οχημάτων για κάθε χρονική στιγμή είναι επαρκής. Επίσης, θα πρέπει να εξασφαλιστεί το ότι δεν εξαντλούνται τα αποθέματα στον προμηθευτή και στον πελάτη. Αυτό επιτυγχάνεται με την χρήση των περιορισμών $d_i^A + q_i t - \sum_{k=0}^t s_{ik} \geq 0$ και $d_i^B + \sum_{k=0}^t s_{ik} - q_i(t+1) \geq 0$. Για όλους περιορισμούς θα πρέπει να ισχύει $d_i^A \geq 0, d_i^B \geq 0, s_{it} \geq 0$ και y_t ακέραιος ≥ 0 .

5. Η περίπτωση της απευθείας αποστολής (the Inventory Routing Problem with Direct Shipping)

Η περίπτωση αυτή γενικά αναφέρεται στις περιπτώσεις που ο προμηθευτής έχει να παραδώσει σε περισσότερους από έναν πελάτη και αυτό θα πρέπει να γίνεται με μεμονωμένες απευθείας αποστολές, δηλαδή παράδοση κάθε φορά στον κάθε πελάτη ξεχωριστά. Αυτό δίνει άλλη διάσταση στο πρόβλημα του IRP καθώς για τη

βελτιστοποίηση ή ελαχιστοποίηση του κόστους αποθέματος και μεταφοράς θα πρέπει να ληφθούν και άλλοι παράγοντες.

Οι Bertazzi & Speranza επανέρχονται με νέα μελέτη [7] και προτείνουν ότι οι αποφάσεις δρομολόγησης πλέον έχουν να κάνουν με αποφάσεις που λαμβάνονται στον χώρο και στον χρόνο.

1.7.2 Αποφάσεις ως τον χρόνο και τον χώρο (Decisions over time and space)

Από την στιγμή που θα πρέπει σε κάθε δρομολόγιο να εξυπηρετηθούν πολλοί πελάτες ταυτόχρονα, θα πρέπει πλέον να σχεδιάζονται διαδρομές όπου θα πρέπει να λαμβάνονται υπόψη οι εξής παραδοχές:

- Οι μέρες εξυπηρέτησης και πόση ποσότητα προϊόντος θα πρέπει να μεταφερθούν τις μέρες αυτές χωρίς να προκληθεί έλλειμα στους πελάτες.
- Κατά τις μέρες εξυπηρέτησης θα πρέπει να είναι γνωστή τόσο η αρχική ποσότητα που μπορεί να δεχθεί ο πελάτης αλλά και η ποσότητα που καταναλώνεται.
- Η χωρητικότητα του οχήματος να είναι τέτοια ώστε να του επιτρέπει να έχει επαρκή χρόνο να καλύψει οποιαδήποτε διαδρομή.
- Όλα τα διαθέσιμα οχήματα να είναι επαρκή ώστε να μπορούν να καλύψουν όλες τις διαδρομές μέσα στην ημέρα.

Με τις παραπάνω παραδοχές οι αποφάσεις δρομολόγησης λαμβάνουν υπόψη τον τρόπο αναπλήρωσης αποθεμάτων, όπου θα πρέπει να γίνεται τις ημέρες εξυπηρέτησης με συγκεκριμένες ποσότητες και τις διαδρομές που θα ακολουθούν τα οχήματα με βάση το κόστος μεταφοράς που θα απαιτείται ανά ημέρα εξυπηρέτησης.

Το IRP πλέον διαχωρίζεται σε δύο κατηγορίες:

1. Το IRP ενός οχήματος (single-vehicle IRP), όπου η δρομολόγηση παραδόσεων γίνεται αποκλειστικά με την χρήση ενός οχήματος.
2. Το IRP με στόλο οχημάτων (multi-vehicle IRP), όπου η δρομολόγηση παραδόσεων πραγματοποιείται με την χρήση στόλου οχημάτων.

1.7.2.1 Το IRP της οχήματος (single-vehicle IRP)

Η μοντελοποίηση του IRP ενός οχήματος ακολουθεί το παρακάτω σκεπτικό:

Ας θεωρήσουμε δεδομένο έναν χρονικό ορίζοντα T με H χρονικές στιγμές (πχ. Ημέρες t), δηλαδή $T = \{1, \dots, H\}$. Ο γράφος $G = (N, E)$ αναπαριστά ένα δίκτυο διανομής όπου ο

κόμβος 0 είναι ο προμηθευτής και οι κόμβοι $N' = \{1, \dots, n\}$ αναπαριστούν τους πελάτες. Για κάθε απόσταση μεταξύ δύο κόμβων i και j υπάρχει ένα κόστος c_{ij} . Κάθε πελάτης έχει ένα μέγιστο επίπεδο αποθέματος U_i . Ο προμηθευτής κάθε ημέρα t διαθέτει r_{0t} μονάδες προϊόντων και ο κάθε πελάτης καταναλώνει r_{it} μονάδες αντίστοιχα.

Το διαθέσιμο όχημα έχει χωρητικότητα Q και ο κάθε πελάτης έχει ένα αρχικό απόθεμα I_{i0} και αντίστοιχα προκύπτει ένα κόστος αποθέματος h_i . Η πολιτική διανομής θα πρέπει να είναι τέτοια ώστε την κάθε ημέρα t η ποσότητα που θα παραδίδεται δεν θα δημιουργεί έλλειψη τόσο στον πελάτη όσο και στον προμηθευτή.

Η ουσιαστική λύση του IRP είναι ότι η αντικειμενική συνάρτηση λαμβάνει τους παραπάνω παράγοντες, έτσι ώστε η διαδρομή παράδοσης θα πρέπει συνδυαστικά να γίνει με το ελάχιστο μεταφορικό και αποθεματικό κόστος.

Η μεταβλητή z_{it} καθορίζει αν έχει γίνει επίσκεψη στον πελάτη i την ημέρα t παίρνοντας την τιμή 0 ή 1 (όχι ή ναι, αντίστοιχα), η μεταβλητή x_{ij} αφορά την ποσότητα που παραδόθηκε στον πελάτη και η μεταβλητή y'_{ij} καθορίζει αν διασχίστηκε η ακμή μεταξύ των κόμβων παίρνοντας τιμή 0 ή 1 αντίστοιχα ή 2 στην περίπτωση που πρέπει η ακμή να διασχισθεί ξανά επειδή ο κόμβος που επισκέπτεται είναι ο τελευταίος. Επίσης, η μεταβλητή I_{it} καθορίζει το ύψος αποθέματος στον κάθε πελάτη. Όλα τα παραπάνω είναι απαραίτητα ώστε η δρομολόγηση να λάβει υπόψη της το πότε θα γίνουν οι παραδόσεις στους πελάτες, πόση ποσότητα θα μεταφερθεί και τέλος πς θα σχεδιαστεί η διαδρομή που θα κάνει το όχημα.

Σε γενικές γραμμές η δρομολόγηση θα πρέπει να ακολουθεί το τρίπτυχο «παραγωγή – κατανάλωση – υπολογισμός αποθέματος», δηλαδή θα πρέπει να είναι γνωστό για κάθε ημέρα t τι ποσότητα παραδίδεται, τι ποσότητα καταναλώνεται και τέλος θα πρέπει να υπολογίζεται το απόθεμα που είχε ο πελάτης πριν την παράδοση, μετά την παράδοση και τι ποσότητα καταναλώθηκε.

Κατά τον σχεδιασμό της δρομολόγησης θα πρέπει να ληφθούν υπόψη επιπλέον και κάποιοι περιορισμοί που αφορούν την πολιτική αναπλήρωσης αποθεμάτων (Maximum Level – ML Policy):

1. Η ποσότητα που θα πρέπει να μεταφέρεται ανά ημέρα δεν θα πρέπει να δημιουργεί έλλειμμα στον προμηθευτή.
2. Η ποσότητα που λαμβάνει ο κάθε πελάτης θα πρέπει να είναι ίδια με την ημερήσια του κατανάλωση.

3. Η ποσότητα που παραδίδεται στον πελάτη δεν θα πρέπει να ξεπερνάει το μέγιστο απόθεμα που μπορεί να διατηρήσει.
4. Το όχημα που εκτελεί τις παραδώσεις θα πρέπει να είναι γεμάτο.
5. Θα πρέπει κάθε ημέρα να γίνεται τουλάχιστον μια επίσκεψη σε ένα πελάτη.

Οπότε καταλήγουμε στην παρακάτω αντικειμενική συνάρτηση που εκφράζει την ελαχιστοποίηση του κόστους διατήρησης αποθέματος στον προμηθευτή, της πελάτες καθώς και του κόστους διανομής:

$$\min \sum_{t \in T} \sum_{i \in N} h_i I_{it} + \sum_{t \in T} \sum_{i \in N} \sum_{j \in N, j > i} c_{ij} y_{ij}^t \quad (5)$$

Οι περιορισμοί που ισχύουν είναι οι εξής :

- $I_{0t} = I_{0,t-1} + r_{0t} - \sum_{i \in N} \chi_{it}$, που καθορίζει το επίπεδο αποθέματος στον προμηθευτή και
- $I_{it} = I_{i,t-1} + \chi_{it} + r_{0t} - \sum_{i \in N} \chi_{it}$, που καθορίζει το επίπεδο αποθέματος στον πελάτη.
- $I_{it} \geq 0$, δεν επιτρέπει την έλλειψη αποθέματος στον πελάτη
- $I_{it} \geq U_i$, το επίπεδο αποθέματος δεν περνάει το μέγιστο απόθεμα που μπορεί ο πελάτης να διατηρήσει.
- $\sum_{i \in N} \chi_{it} \leq Q z_{0t}$, η συνολική ποσότητα που παραλαμβάνεται από τον προμηθευτή δεν ξεπερνάει την διαθέσιμη χωρητικότητά του. Η μεταβλητή z_{0t} δηλώνει την επίσκεψη στον προμηθευτή ($z_{0t} = 1$).
- $\chi_{it} \leq U_i z_{it}$, η συνολική ποσότητα που παραδίδεται στον πελάτη δεν ξεπερνάει το ύψος του μέγιστου αποθέματος που μπορεί να διατηρήσει και η μεταβλητή z_{it} παίρνει την τιμή 1 στην περίπτωση που γίνει επίσκεψη στον πελάτη ($z_{it} = 1$)
- $\sum_{j \in N, j > i} y_{ij}^t + \sum_{j \in N, j > i} y_{ji}^t = 2 z_{it}$, ότι στην περίπτωση που δύο ακμές προσπίπτουν σε ένα κόμβο (πελάτης) τότε αυτές διασχίστηκαν στην περίπτωση επίσκεψης.
- $\sum_{j \in S} \sum_{j \in S, j > i} y_{ji}^t \leq \sum_{j \in S} z_{it} - z_{st}$, όπου διασφαλίζεται ότι η διαδρομή που εκτελείται περιλαμβάνει ως αφετηρία και τερματισμό την αποθήκη και στην

διαδρομή αυτή επισκέπτεται κάθε πελάτη s ανά ημέρα t . Στην περίπτωση που πραγματοποιείται επίσκεψη έχουμε $z_{st} = 1$.

Για όλους τους παραπάνω περιορισμούς ισχύει ότι:

- $t \in T, i \in N', i \in N, S \subseteq N', s \in S, j \in N'$.
- $\chi_{it} \geq 0, i \in N', t \in T$.
- $y_{ij}^t \in \{0, 1\}, i \in N', j \in N', t \in T$.
- $y_{0j}^t \in \{0, 1, 2\}, j \in N', t \in T$.
- $z_{it} \in \{0, 1\}, j \in N, t \in T$.

1.7.2.2 Το IRP με στόλο οχημάτων (multi-vehicle IRP)

Πρόκειται για την επέκταση του single-vehicle IRP και μοντελοποιείται με τις παρακάτω παραμετροποιήσεις:

Θεωρούμε ένα σύνολο K με k οχήματα που το καθένα έχει χωρητικότητα Q . Η μεταβλητή z_{it}^k δείχνει εάν το όχημα k έχει επισκεφθεί τον πελάτη i την ημέρα t και παίρνει τιμές 0 ή 1 (όχι ή ναι). Η μεταβλητή χ_{it}^k καθορίζει την ποσότητα που παραδόθηκε στον πελάτη από το όχημα k την ημέρα t και η y_{ij}^{kt} δείχνει αν διασχίστηκε η ακμή (i, j) από το όχημα k την ημέρα t , όπου παίρνει τις τιμές 0 ή 1 για τον αν διασχίστηκε η όχι και την τιμή 2 αν διασχίστηκε 2 φορές (στην περίπτωση που ο πελάτης είναι η τελευταία επίσκεψη).

Η αντίστοιχη συνάρτηση ελαχιστοποίησης είναι:

$$\min \sum_{t \in T} \sum_{i \in N} h_i I_{it} + \sum_{k \in K} \sum_{t \in T} \sum_{i \in N} \sum_{j \in N, j > i} c_{ij} y_{ij}^{kt} \quad (6)$$

και υπόκειται στους παρακάτω περιορισμούς:

- $I_{0t} = I_{0,t-1} + r_{0t} - \sum_{k \in K} \sum_{i \in N'} \chi_{ij}^{kt}$, όπου καθορίζει το επίπεδο αποθέματος στον προμηθευτή και
- $I_{it} = I_{i,t-1} + \chi_{it} + r_{0t} - \sum_{k \in K} \sum_{i \in N'} \chi_{it}$ που καθορίζει το επίπεδο αποθέματος στον πελάτη.
- $I_{it} \geq 0$, δεν επιτρέπει την έλλειψη αποθέματος στον πελάτη.
- $I_{it} \geq U_i$, το επίπεδο αποθέματος δεν περνάει το μέγιστο απόθεμα που μπορεί ο πελάτης να διατηρήσει.

- $\sum_{i \in N'} \chi_{it}^k \leq Qz_{0t}^k$, η συνολική ποσότητα που παραλαμβάνεται από τον προμηθευτή με το όχημα k δεν ξεπερνάει την διαθέσιμη χωρητικότητά του. Η μεταβλητή z_{0t}^k δηλώνει την επίσκεψη του οχήματος k στον προμηθευτή ($z_{0t}^k = 1$).
- $\chi_{it}^k \leq U_i z_{it}^k$, η συνολική ποσότητα που παραδίδεται στον πελάτη με το όχημα k δεν ξεπερνάει το ύψος του μέγιστου αποθέματος που μπορεί να διατηρήσει και η μεταβλητή z_{it}^k παίρνει την τιμή 1 στην περίπτωση που γίνει επίσκεψη στον πελάτη ($z_{it}^k = 1$).
- $\sum_{j \in N, j > i} y_{ij}^{kt} + \sum_{j \in N, j > i} y_{ji}^{kt} = 2z_{it}^k$, ότι στην περίπτωση που δύο ακμές προσπίπτουν σε ένα κόμβο (πελάτης) τότε αυτές διασχίστηκαν στην περίπτωση επίσκεψης.
- $\sum_{j \in S} \sum_{j \in S, j > i} y_{ji}^{kt} \leq \sum_{j \in S} z_{ji}^{kt} - z_{st}^k$, όπου διασφαλίζεται ότι η διαδρομή που εκτελείται από τα οχήματα περιλαμβάνει ως αφετηρία και τερματισμό την αποθήκη και στην διαδρομή αυτή επισκέπτεται κάθε πελάτη s ανά ημέρα t . Στην περίπτωση που πραγματοποιείται επίσκεψη έχουμε $z_{st}^k = 1$.

Για όλους τους παραπάνω περιορισμούς ισχύει ότι:

- $t \in T, i \in N', i \in N, S \subseteq N', s \in S, j \in N', k \in K$
- $\chi_{it_{it}}^k \geq 0, i \in N', t \in T$.
- $y_{ij}^{kt} \in \{0, 1\}, i \in N', j \in N', t \in T$.
- $y_{0j}^{kt} \in \{0, 1, 2\}, j \in N', t \in T$.
- $z_{it}^k \in \{0, 1\}, j \in N', t \in T$.

2. Ο Pelican Optimization Algorithm και η Συνάρτηση Καταλληλότητας (Fitness Function)

2.1 Pelican Optimization Algorithm

Ο Pelican Optimization Algorithm (POA) είναι ένας από τους πιο σύγχρονους αλγόριθμους βελτιστοποίησης και παρουσιάστηκε από τους Pavel Tojovsky και Mohammad Dehghani (2022) [30]. Η έμπνευση του προήλθε από τον τρόπο με τον οποίο οι πελεκάνοι κυνηγούν τα θηράματά τους. Οι πελεκάνοι είναι μεγαλόσωμα πτηνά που ζούνε σε μεγάλα σμήνη και διαθέτουν μακρύ ράμφος με μάρσιππο που τους επιτρέπει να πιάνουν την τροφή τους που συνήθως είναι ψάρια. Έχουν βάρος περίπου 15 κιλά, φτάνουν σε ύψος τα 1,83 μέτρα και το άνοιγμα των φτερών τους κυμαίνεται από 0,5 έως 3 μέτρα [23]. Συνεργάζονται συχνά μαζί για να κυνηγήσουν με τον εξής τρόπο: Πετάνε σε ύψος 10-20 μέτρων πάνω από υδάτινες επιφάνειες και όταν εντοπίσουν την τροφή τους, βουτάνε προς το σημείο που βρίσκεται το θήραμα και όταν το προσεγγίζουν ανοίγουν τα φτερά τους, αναγκάζοντας το θήραμα να πάει σε αβαθή νερά ώστε στην συνέχεια να μπορέσουν να το πιάσουν.

2.2 Το μαθηματικό μοντέλο του POA

Ο POA είναι αλγόριθμος πληθυσμιακής μεθόδου όπου ο κάθε πελεκάνος είναι μέλος ενός πληθυσμού. Στους πληθυσμιακούς αλγόριθμους το κάθε μέλος του πληθυσμού είναι μια υποψήφια λύση που προτείνει τιμές για μεταβλητές προβλημάτων βελτιστοποίησης αναλόγως με το σημείο που βρίσκεται στο χώρο που καταλαμβάνει ο πληθυσμός. Κάθε πρόβλημα βελτιστοποίησης έχει ένα ελάχιστο και ένα μέγιστο όριο. Ο πληθυσμός αρχικοποιείται ανάμεσα στα όρια αυτά και προσδιορίζεται από την παρακάτω εξίσωση:

$$x_{i,j} = l_j + \text{rand}(u_j - l_j) \quad (7)$$

Όπου $x_{i,j}$ είναι η τιμή της $j = 1, 2, \dots, m$ μεταβλητής του προβλήματος βελτιστοποίησης που καθορίζεται από την $i = 1, 2, \dots, N$ υποψήφια λύση, m είναι το πλήθος των μεταβλητών του προβλήματος, N είναι ο αριθμός του πληθυσμού, rand είναι ένας τυχαίος αριθμός μεταξύ του διαστήματος $[0, 1]$, l_j είναι το ελάχιστο όριο της και u_j είναι το μέγιστο όριο της μεταβλητής j .

Ο πληθυσμός αναπαριστάται σε πίνακες όπου σε κάθε γραμμή του πίνακα είναι οι υποψήφιες λύσεις και σε κάθε στήλη οι προτεινόμενες τιμές για τις μεταβλητές του προβλήματος βελτιστοποίησης.

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_i \\ \vdots \\ X_N \end{bmatrix}_{N \cdot m} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,j} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots & & \vdots \\ x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,m} \\ \vdots & & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,j} & \cdots & x_{N,m} \end{bmatrix}_{N \cdot m} \quad (8)$$

Στην παραπάνω απεικόνιση όπου X είναι ο πίνακας του πληθυσμού των πελεκάνων και X_i ο i -οστός πελεκάνος ή υποψήφια λύση.

Εφόσον κάθε πελεκάνος είναι μια υποψήφια λύση για το πρόβλημα βελτιστοποίησης που έχουμε, τότε η κάθε υποψήφια λύση μπορεί να είναι η αντικειμενική τιμή της συνάρτησης του προβλήματος. Οι τιμές που λαμβάνονται για την αντικειμενική συνάρτηση καθορίζονται από το παρακάτω διάνυσμα αντικειμενικής συνάρτησης:

$$F = \begin{bmatrix} F_1 \\ \vdots \\ F_i \\ \vdots \\ F_N \end{bmatrix}_{N \cdot m} = \begin{bmatrix} F(X_1) \\ \vdots \\ F(X_i) \\ \vdots \\ F(X_N) \end{bmatrix}_{N \cdot m} \quad (9)$$

όπου F είναι η αντικειμενική συνάρτηση και F_i είναι η τιμή της αντικειμενικής συνάρτησης της i -οστής υποψήφιας λύσης.

Ο αλγόριθμος εξομοιώνει την συμπεριφορά και την στρατηγική των πελεκάνων όταν επιτίθενται και κυνηγάνε το θήραμα τους, ανανεώνοντας υποψήφιες λύσεις. Η στρατηγική αυτή γίνεται σε δύο φάσεις:

1. Φάση Εξερεύνησης (Exploration Phase): όπου ο πελεκάνος κινείται απευθείας στο θήραμα.

2. Φάση Εκμετάλλευσης (Exploitation Phase): όπου ο πελεκάνος πετάει πάνω από την επιφάνεια του νερού για να οδηγήσει το θήραμα στα ρηχά.

2.3 Η Φάση Εξερεύνησης (Φάση 1)

Στην φάση της εξερεύνησης ο αλγόριθμος προσομοιώνει την ικανότητα των πελεκάνων να κινούνται σε ευρείες περιοχές ώστε να αναζητήσουν πιθανή τροφή ή καλύτερη λύση στον χώρο αναζήτησης. Κύριο στοιχείο του αλγορίθμου είναι ότι η θέση του θηράματος εμφανίζεται τυχαία στον χώρο αυτό με αποτέλεσμα ο πελεκάνος (υποψήφια λύση) αποφασίζει να κινηθεί προς αυτό αναλόγως της θέσης του. Εάν η θέση του θηράματος είναι καλύτερη από την τρέχουσα θέση του, τότε επιτίθεται στο θήραμα και η νέα θέση του πελεκάνου είναι και η καλύτερη υποψήφια λύση στο πρόβλημα βελτιστοποίησης. Αντίθετα, αν ο πελεκάνος βρίσκεται ήδη σε καλύτερη θέση συγκριτικά με αυτήν του θηράματος, τότε δεν επιτίθεται και συνεπώς η υποψήφια λύση θεωρείται ήδη βέλτιστη.

Η παραπάνω λογική μπορεί να αναπαρασταθεί με την παρακάτω εξίσωση:

$$\chi_{i,j}^{p_1} = \begin{cases} \chi_{i,j} + rand(p_j - I \cdot \chi_{i,j}), & F_p < F_i \\ \chi_{i,j} + rand(\chi_{i,j} - p_j), & \text{αλλιώς,} \end{cases} \quad (10)$$

Όπου, $\chi_{i,j}^{p_1}$ είναι η νέα τιμή του i -οστού πελεκάνου στην j -οστή διάσταση που προκύπτει μετά την πρώτη φάση, $rand$ είναι ένας τυχαίος αριθμός μεταξύ του διαστήματος $[0, 1]$, p_j Η θέση του θηράματος στη διάσταση j , F_p η τιμή της αντικειμενικής συνάρτησης που προκύπτει μετά την πρώτη φάση και I είναι μια ειδική παράμετρος όπου παίρνει τυχαία την τιμή 1 ή 2 που έχει να κάνει με το βήμα προσέγγισης προς το θήραμα (1 ή 2 φορές από την υφιστάμενη θέση του). Η ύπαρξη της παραμέτρου I βοηθάει τον αλγόριθμο να σαρώνει τον χώρο αναζήτησης καλύτερα.

Εάν μετά την πρώτη φάση προκύψει ότι η καινούργια θέση είναι καλύτερη τότε η τιμή της αντικειμενικής συνάρτησης παίρνει την τιμή αυτή. Κατά την διαδικασία αυτή ο αλγόριθμος ελέγχει αν η νέα τιμή είναι μεταξύ του ελάχιστου και μέγιστου ορίου στον χώρο αναζήτησης. Εάν δεν βγει εκτός των ορίων τότε η τιμή ανανεώνεται αλλιώς επανέρχεται στην αρχική της. Η νέα θέση του πελεκάνου προσδιορίζεται από την παρακάτω εξίσωση:

$$X_i = \begin{cases} X_i^{P_1}, & F_i^{P_1} < F_i \\ X_i, & \text{αλλιώς,} \end{cases} \quad (11)$$

Όπου $X_i^{P_1}$ είναι η νέα θέση i -οστού πελεκάνου και $F_i^{P_1}$ η νέα τιμή της αντικειμενικής συνάρτησης που προκύπτει μετά την Φάση 1.

2.4 Φάση Εκμετάλλευσης (Φάση 2)

Αφού ο πελεκάνος εντόπισε την ευρύτερη περιοχή όπου βρίσκεται το θήραμα, τώρα πρέπει να γίνει πιο ακριβής για να το πιάσει. Όταν φτάσει στην επιφάνεια του νερού, ο πελεκάνος απλώνει τα φτερά του και κάνει μικρές κινήσεις με αυτά ώστε να οδηγήσει το θήραμα στην επιφάνεια του νερού και να το φάει. Αυτός ο τρόπος αναγκάζει τον αλγόριθμο να συγκλίνει σε καλύτερα σημεία στο χώρο αναζήτησης ενώ ενισχύει την δυναμική του να εξερευνήσει τοπικά στη περιοχή της επίθεσης. Στη συνέχεια θα πρέπει να εξετάσει τα σημεία αυτά συγκριτικά με την θέση του και να συγκλίνει σε μια ακόμη καλύτερη λύση. Η νέα θέση που προκύπτει από την Φάση 2 είναι η εξής:

$$X_{i,j}^{P_2} = X_{i,j} + R \left(1 - \frac{t}{T} \right) (2rand - 1) X_{i,j} \quad (12)$$

Όπου $X_{i,j}^{P_2}$ είναι η νέα θέση είναι η νέα τιμή του i -οστού πελεκάνου στην j -οστή διάσταση που προκύπτει μετά την δεύτερη φάση, $X_{i,j}$ η υφιστάμενη θέση του πελεκάνου και $rand$ είναι ένας τυχαίος αριθμός μεταξύ του διαστήματος $[0, 1]$. Ο συντελεστής $R \left(1 - \frac{t}{T} \right)$ αναπαριστά την ακτίνα της περιοχής στην οποία ο πελεκάνος ψάχνει γύρω από τον εαυτό του, ώστε σε κάθε κίνηση ή επανάληψη της αναζήτησης του συγκλίνει σε καλύτερη λύση. Στον συντελεστή αυτόν όπου R είναι μια σταθερά και είναι ίση με 0,2 ($R = 0,2$), t είναι ο αριθμός της επανάληψης και T ο συνολικός αριθμός των επαναλήψεων.

2.5 Ψευδοκώδικας POA

Παρακάτω παρουσιάζεται ο ψευδοκώδικας του POA όπως ακριβώς περιγράφεται στο [30]:

Είσοδος:

- Αντικειμενική Συνάρτηση $F(X)$ της ελαχιστοποίησης.
- Όρια Μεταβλητών $l[1...m]$, $u[1...m]$.

- Μέγεθος Πληθυσμού N .
- Μέγιστες Επαναλήψεις T .
- Σταθερά $R = 0.2$.

Έξοδος

- Καλύτερη Λύση x_best που βρέθηκε.

Έναρξη POA:

```

1. Αρχικοποίηση Πληθυσμού  $X[1..N]$  τυχαία στα Όρια Μεταβλητών:
   Για  $i = 1..N$ :
       Για  $j = 1..m$ :
            $X[i][j] = l[j] + \text{rand}(0,1) * (u[j] - l[j])$ 
2. Υπολογισμός fitness  $F_i = F(X[i])$  για κάθε  $i$ .
3. Ορισμός ( $x\_best, f\_best$ ) =  $\text{argmin}_i F_i$ 
4. Για  $t = 1..T$  κάνε:
5. // Δημιουργία θηράματος ( $p$ ) τυχαία στον χώρο αναζήτησης. Στην
   τρέχουσα επανάληψη το //θήραμα είναι κοινό για όλους της πελεκάνους.
6.   Για  $j = 1..m$ :
7.      $p[j] = l[j] + \text{rand}(0,1) * (u[j] - l[j])$ 
8.      $F_p = f(p)$ 
9. Για  $i = 1..N$  κάνε:
10.    //Φάση 1: Εξερεύνηση
11.    Επέλεξε  $I$  τυχαία από το {1, 2}
12.    // Μπορεί να επαναδειγματοληφθεί ανά πελεκάνο (η/και ανά
    επανάληψη)
13.    Δημιουργία υποψήφιας λύσης  $Xp1 = X[i]$ 
14.    Για  $j = 1..m$  κάνε:
15.      Εάν  $F_p < F_i$  τότε:
16.         $Xp1[j] = X[i][j] + \text{rand}(0,1) * (p[j] - I * X[i][j])$ 
17.      Αλλιώς:
18.         $Xp1[j] = X[i][j] + \text{rand}(0,1) * (X[i][j] - p[j])$ 
19.      Τέλος Εάν
20.    Τέλος Επανάληψης
21.    Εφαρμογή ορίων στο  $Xp1$  (περιορισμός κάθε διάστασης στο [ $l[j],$ 
     $u[j]$ ])
22.     $Fp1 = f(Xp1)$ 
23.    // Ενημέρωση στην περίπτωση βελτίωσης
24.    Εάν  $Fp1 < F_i$  τότε:
25.       $X[i] = Xp1$ 
26.       $F_i = Fp1$ 
27.    Τέλος Εάν
28.    //Φάση 2: Εκμετάλλευση
29.    Δημιουργία υποψήφιας λύσης  $Xp2 = X[i]$ 
30.     $\text{radius} = R * (1 - t / T)$ 
31.    Για  $j = 1..m$  κάνε:
32.       $Xp2[j] = X[i][j] + \text{radius} * (2 * \text{rand}(0,1) - 1) * X[i][j]$ 
33.    Τέλος Για
34.    Εφαρμογή ορίων στο  $Xp2$  (περιορισμός κάθε διάστασης στο [ $l[j],$ 
     $u[j]$ ])
35.     $Fp2 = f(Xp2)$ 
36.    // Ενημέρωση στην περίπτωση βελτίωσης
37.    Εάν  $Fp2 < F_i$  τότε:
38.       $X[i] = Xp2$ 
39.       $F_i = Fp2$ 
40.    Τέλος Εάν
41.    Τέλος Για //Τέλος επανάληψης για πελεκάνους
42.    Ενημέρωση ( $x\_best, f\_best$ ) =  $\text{argmin}(F_i)$  εάν βελτιώθηκε

```

43.	Τέλος Για
44.	Επιστροφή x_best
Τέλος POA.	

Ψευδοκώδικας 2 - 1: Ο αλγόριθμος POA

Ο αλγόριθμος ξεκινάει δημιουργώντας τον αρχικό πληθυσμό των πελεκάνων έχοντας ορίσει τις παραμέτρους N (Πληθυσμός) και Μέγιστες Επαναλήψεις (T). Οι θέσεις των πελεκάνων (X) επιλέγονται τυχαία μέσα στα όρια αναζήτησης $l[1 \dots m]$ και $u[1 \dots m]$. Υπολογίζεται το fitness για κάθε θέση X_i του πελεκάνου και στην συνέχεια ορίζεται η καλύτερη λύση που βρέθηκε (x_best, f_best).

Στη συνέχεια ξεκινάει ο βρόγχος επαναλήψεων (από $t = 1$ έως T) όπου επαναλαμβάνονται τα παρακάτω βήματα:

1. Δημιουργείται μία τυχαία θέση Θηράματος (p) μέσα στον χώρο αναζήτησης και υπολογίζεται η τιμή της $f(p)$.
2. Εκτελείται η Φάση 1 (Εξερεύνηση) όπου κάθε πελεκάνος θα αποφασίσει πώς θα κινηθεί σε σχέση με το Θήραμα. Συγκρίνεται η τιμή της Fp και Fi και εάν το $Fp < Fi$ τότε ο πελεκάνος κινείται προς το θήραμα και υπολογίζεται η νέα θέση $Xp[j]$, αλλιώς εάν $Fp \geq Fi$ υπολογίζεται η νέα θέση $Xp[j]$ με την αντίστοιχη εξίσωση. Στη συνέχεια ελέγχεται αν η νέα θέση είναι εντός των ορίων και στη συνέχεια ενημερώνεται η τιμή της Fi στην περίπτωση είναι βελτιωμένη ($Fp1 < Fi$).
3. Ο αλγόριθμος στην συνέχεια περνάει στην Φάση 2 (Εκμετάλλευση) όπου ο πελεκάνος θα βελτιώσει εκ νέου την θέση του βάσει της αντίστοιχης εξίσωσης. Εάν η νέα θέση είναι εντός των ορίων αναζήτησης και υπάρξει βελτίωση ($Fp2 < Fi$) τότε κρατάει τη θέση αυτή αλλιώς την απορρίπτει.
4. Μόλις τελειώσει η τρέχουσα επανάληψη ενημερώνεται η τιμή (x_best, f_best).

Μόλις τελειώσει και το σύνολο των επαναλήψεων (T) τότε ο αλγόριθμος σταματά και επιστρέφει την τιμή x_best .

2.6 Εφαρμογή του POA στο IRP

Το μαθηματικό μοντέλο του Pelican Optimization Algorithm (POA) σχεδιάστηκε πρωτίστως για την επίλυση προβλημάτων σε συνεχή χώρο, όπου οι μεταβλητές μπορούν να λάβουν μια οποιαδήποτε δεκαδική τιμή εντός των επιτρεπτών ορίων. Αντιθέτως, το Πρόβλημα Δρομολόγησης Αποθέματος (IRP) ανήκει στην κατηγορία των διακριτών

προβλημάτων βελτιστοποίησης (*discrete optimization*), καθώς απαιτεί αποφάσεις που αφορούν ακέραιες μεταβλητές της συνάρτησής της, όπως η επιλογή του πελάτη που θα εξυπηρετηθεί, η σειρά επίσκεψης και η κατανομή του στόλου.

Για να μπορέσει ο POA να επιλύσει αποτελεσματικά το Multi-Vehicle IRP, χρειάστηκε να αναπτυχθεί ένας μηχανισμός ο οποίος θα γεφυρώσει το χάσμα μεταξύ του συνεχούς και του διακριτού χώρου. Η διαδικασία αυτή λαμβάνει τη συνεχή θέση του πελεκάνου (η οποία αποτελεί μια υποψήφια λύση) και τη μετασχηματίζει σε ένα πλήρως λειτουργικό, διακριτό πλάνο δρομολόγησης. Η αξιολόγηση του πλάνου αυτού γίνεται μέσω της συνάρτησης καταλληλότητας (*fitness function*).

2.7 Οι Εικονικοί Κόμβοι (Virtual Nodes)

Η θέση του κάθε πελεκάνου αναπαρίσταται ως ένα μονοδιάστατο διάνυσμα συνεχών τιμών. Το πρώτο πράγμα που πρέπει να κάνει αυτή η συνάρτηση είναι να μετασχηματιστεί το διάνυσμα αυτό σε έναν τρισδιάστατο πίνακα διαστάσεων $T \times C \times 2$, όπου:

- T : Ο συνολικός χρονικός ορίζοντας (π.χ. ημέρες),
- C : Το πλήθος των Εικονικών Κόμβων (Virtual Nodes), το οποίο ισούται με το πλήθος των πελατών,
- 2: Οι γεωγραφικές συντεταγμένες (X, Y) στον χώρο αναζήτησης. Στην ουσία είναι οι δύο θέσεις που κρατάνε μία τιμή για την κάθε συντεταγμένη στον χώρο αναζήτησης ($X = T \times C$ και $Y = T \times C$).

Οι συντεταγμένες αυτές περιορίζονται αυστηρά εντός του ορίου $[0, 500]$, εναρμονιζόμενες πλήρως με τις γεωμετρικές διαστάσεις του δικτύου των παραδειγμάτων που θα επεξεργαστούμε στην έρευνά μας [26]. Με έναν τρόπο λοιπόν θα πρέπει για κάθε ημέρα t να δημιουργηθούν C εικονικοί κόμβοι στο συγκεκριμένο επίπεδο ($[0, 500]^2$). Στην συνέχεια αυτοί οι εικονικοί κόμβοι θα πρέπει να ταυτοποιηθούν με τους πραγματικούς κόμβους (πελάτες) του δικτύου (που εκτείνονται στον ίδιο γεωγραφικό χώρο), που έχουν πραγματική ανάγκη αναπλήρωσης σε αποθέματα. Αυτό επιτυγχάνεται μέσω της εγγύτητας των εικονικών κόμβων με τους πραγματικούς χρησιμοποιώντας την μέθοδο της Ευκλείδειας Απόστασης:

$$E_{i,j} = \sqrt{(y_j - y_i)^2 + (x_j - x_i)^2}, \quad i, j \in N \quad (13)$$

Όταν προσδιοριστεί η εγγύτητα των κόμβων, τότε δημιουργείται μία σειρά κατάταξης ή αλληλουχία επισκέψεων (*daily sequence*). Με αυτόν τον τρόπο οι μεταβλητές του ΡΟΑ μεταφράζονται σε μία διακριτή σειρά εξυπηρέτησης.

2.8 Ο Αλγόριθμος της Συνάρτησης Καταλληλότητας (Fitness Function)

Για να μετατρέψουμε αυτήν την αλληλουχία επισκέψεων σε ένα σχέδιο διανομής, δημιουργήθηκε ένας αλγόριθμος ο οποίος θα πρέπει να εκτελείται επαναληπτικά για κάθε χρονική περίοδο $t \in T$, να προσομοιώνει τη διανομή και να υπολογίζει το συνολικό κόστος. Τα βήματα του αλγορίθμου είναι τα εξής:

Βήμα 1: Ενημέρωση Αποθέματος Προμηθευτή

Στην αρχή κάθε ημέρας, το απόθεμα του προμηθευτή αυξάνεται κατά τον ρυθμό παραγωγής, χωρίς ωστόσο να ξεπερνά την μέγιστη αποθηκευτική του ικανότητα.

Βήμα 2: Υπολογισμός Απαιτούμενη Ποσότητας

Για κάθε πελάτη της ημερήσιας λίστας (αλληλουχία επισκέψεων) υπολογίζεται η επιθυμητή ποσότητα παράδοσης. Ο αλγόριθμος δεν εστιάζει σε μόνο στην τρέχουσα ζήτηση αλλά ενσωματώνει μια στρατηγική «κυλιόμενου παραθύρου/ ορίζοντα (*Sliding Window – Rolling Horizon*)» όπου συνυπολογίζονται και οι μελλοντικές απαιτήσεις ώστε να προληφθούν πιθανές ελλείψεις. Η στρατηγική αυτή έχει χρησιμοποιηθεί στη βιβλιογραφία [18, 29, 10] όπου επιτρέπει να ελέγχονται τα αποθέματα σε τμηματικά μικρότερα χρονικά διαστήματα, αντί σε ολόκληρο τον χρονικό ορίζοντα. Η ποσότητα αυτή οριοθετείται τόσο από τον διαθέσιμο χώρο του πελάτη όσο και από το διαθέσιμο απόθεμα του προμηθευτή. Ακόμη η χρήση της παραπάνω στρατηγικής έχει θετική επίδραση στο συνολικό κόστος καθώς επιτυγχάνεται σημαντική εξοικονόμηση πόρων.

Βήμα 3: Κατανομή Στόλου και Δρομολόγηση

Δεδομένου ότι διαθέτουμε πολλαπλά οχήματα χωρητικότητας Q , τα οχήματα φορτώνονται στο μέγιστο της χωρητικότητάς τους λόγω της πολιτικής αναπλήρωσης αποθεμάτων, ενώ εάν οι απαιτήσεις ενός πελάτη υπερβαίνουν την χωρητικότητα του οχήματος τότε επιτρέπεται η τμηματική παράδοση (*split deliveries*). Με την τμηματική παράδοση, το υπόλοιπο της ζήτησης καλύπτεται με το επόμενο διαθέσιμο όχημα.

Βήμα 4: Βελτιστοποίηση Διαδρομών (Αλγόριθμος 2-opt)

Αφού έχουν οριστικοποιηθεί οι πελάτες που ανατίθενται σε κάθε όχημα K , κατασκευάζονται οι διαδρομές που θα ακολουθήσουν τα οχήματα (από και προς τον προμηθευτή). Για την επίλυση του προβλήματος της βέλτιστης δρομολόγησης (*Travel Sales Man Problem – TSP*) [14] εφαρμόζεται ο αλγόριθμος 2-opt που είναι το πιο σύνηθες που χρησιμοποιείται στην βιβλιογραφία [13]. Το άθροισμα των αποστάσεων όλων των οχημάτων προστίθεται στο συνολικό κόστος.

Βήμα 5: Αξιολόγηση Αποθεμάτων και Επιβολή Δυναμικών Ποινών

Στο τέλος της ημέρας, η ζήτηση του πελάτη αφαιρείται από το απόθεμά του. Εάν το τελικό απόθεμα είναι θετικό, υπολογίζεται το κόστος διατήρησης αποθέματος (*holding cost*).

Ωστόσο, εάν προκύψει *stockout*, δηλαδή το απόθεμα του πελάτη είναι αρνητικό, τότε στο σημείο αυτό αντί να απορριφθεί η λύση του προβλήματος ως μη εφικτή, εφαρμόζεται μία δυναμική ποινή με την χρήση ενός συντελεστή λ [22, 24].

Προκειμένου οι ποινή αυτή να προσαρμόζεται σε κάθε είδος προβλήματος δρομολόγησης, ανεξαρτήτως μέγεθος δικτύου, τα όριά της υπολογίζονται δυναμικά με βάση το μέγιστο δυνατό κόστος δρομολόγησης.

Η συνολική ποινή υπολογίζεται ως το γινόμενο της ποσότητας έλλειψης επί τον συντελεστή $\lambda_{stockout}$ ο οποίος αυξάνεται γραμμικά με την πρόοδο των επαναλήψεων:

$$\lambda_{stockout} = P_{start} + \left(\frac{iter}{iter_{max}} \right) \cdot (P_{end} - P_{start}) \quad (14)$$

Όπου P_{start} η αρχική ποινή, P_{end} η μέγιστη τελική ποινή, $iter$ αριθμός επανάληψης και $iter_{max}$: η μέγιστη επανάληψη. Η αρχική ποινή, προσδιορίστηκε με την προσαύξηση κατά 20% του χειρότερου κόστους δρομολόγησης ($P_{start} = worst_case_route_cost \times 1,2$), ενώ η τελική ποινή είναι τριπλάσια της αρχικής ποινής ($P_{end} \times 3$).

Με τον τρόπο αυτό ο αλγόριθμος POA αποκτά μια ευελιξία στο τρόπο που εξερευνά τον χώρο αναζήτησης, όπου αποφεύγει τις μη εφικτές λύσεις οι οποίες κοστίζουν πολύ. Καθώς οι επαναλήψεις προχωρούν και το κόστος γίνεται υπερβολικά υψηλό, ο POA θα αναγκαστεί

να αποφύγει τις λύσεις αυτές και να οδηγηθεί αυστηρά σε εφικτές και βέλτιστες λύσεις ενώ ταυτόχρονα να μην εγκλωβίζεται σε τοπικά βέλτιστα [19, 25].

Στο τέλος το συνολικό κόστος που εξάγεται από την συνάρτηση περιλαμβάνει το κόστος μεταφοράς, το κόστος διατήρησης αποθεμάτων και τις ενδεχόμενες τιμές. Στην συνέχεια αυτό το κόστος είναι που αξιολογείται από τον POA μέχρι να καταλήξει σε αυτό που είναι θεωρεί βέλτιστο.

2.9 Ο ψευδοκώδικας της Συνάρτησης Καταλληλότητας

Παρακάτω παρουσιάζεται ο ψευδοκώδικας της fitness function όπου αποτυπώνονται τα βήματα του αλγορίθμου όπως περιεγράφηκαν στην προηγούμενη ενότητα:

Είσοδος:

- Δεδομένα: N, T, K, Q, Πίνακας Αποστάσεων, Απόθεμα Προμηθευτή, Απόθεμα Πελάτη, Μέγιστο Απόθεμα Προμηθευτή, Μέγιστο Απόθεμα Πελάτη, Ημερήσια Παραγωγή, Κόστος Διατήρησης Αποθέματος, Κόστος Αποθέματος Προμηθευτή.
- Pelican_position
- Current_iteration, Max_iteration

Εξοδος:

- Συνολικό Κόστος

Εναρξη Fitness:

1. $C = N$ //Εικονικοί κόμβοι ανά ημέρα
2. Pelican position = reshape(T, C, 2)
3. Υπολογισμός Δυναμικής ποινής λ:
4. Progress = Current_iteration / Max_iteration
5. Χειρότερο κόστος δρομολόγησης = max(Πίνακας Αποστάσεων)
6. Αρχική Ποινή = Χειρότερο κόστος δρομολόγησης * 1.2
7. Τελική Ποινή = Αρχική Ποινή * 3.0
8. Δυναμική ποινή λ = Αρχική Ποινή + Progress * (Τελική Ποινή - Αρχική Ποινή)
9. Συνολικό Κόστος = 0.0
10. Για t = 1...T:
11. // Ενημέρωση Αποθέματος Προμηθευτή
12. Απόθεμα Προμηθευτή = min(Απόθεμα Προμηθευτή + Ημερήσια Παραγωγή, Μέγιστο Απόθεμα Προμηθευτή)
13. //Ταυτοποίηση υποψήφιων πελατών
14. Υποψήφιοι Πελάτες = {Πελάτες[i] με διαθέσιμο χώρο < 0}
15. // Δημιουργία σειράς επίσκεψης
16. Σειρά επίσκεψης = []
17. Εάν Απόθεμα Πελάτη > 0 και Υποψήφιοι Πελάτες τότε:
18. Για v = 1...C:
19. Συντεταγμένες = determineNearestNodes(v, Υποψήφιοι Πελάτες)
20. Σειρά Επίσκεψης = append(Πελάτες[i])
21. // Φόρτωση και Δρομολόγηση οχήματος
22. Διαδρομές = []
23. Φορτία = []
24. Τρέχον Όχημα = 1

```

25. Για Υποψήφιοι Πελάτες στη Σειρά Επίσκεψης:
26. Εάν Απόθεμα Προμηθευτή >= 0 τότε:
27. ΔΙΑΚΟΠΗ
28. Κυλιόμενος Ορίζοντας = min(t+ημέρες,T) // ημέρες = 1, 2, 3
29. Μελλοντική Ζήτηση = sum(Συνολική Ζήτηση[t, Κυλιόμενος
    Ορίζοντας])
30. Διαθέσιμος Χώρος = max(0, Μελλοντική Ζήτηση - Απόθεμα
    Πελάτη[Υποψήφιοι Πελάτες])
31. Επιθυμητή Ποσότητα = min(Ελεύθερος Χώρος, Διαθέσιμος Χώρος,
    Απόθεμα Προμηθευτή)
32. Όσο Επιθυμητή Ποσότητα > 0:
33. Αν Τρέχον Όχημα > K:
34. ΔΙΑΚΟΠΗ //Δεν υπάρχει διαθέσιμο όχημα
35. Αν Φορτία[Τρέχον Όχημα] >= Q:
36. Τρέχον Όχημα = Τρέχον Όχημα + 1
37. ΣΥΝΕΧΕΙΑ
38. Ποσότητα = min(Επιθυμητή Ποσότητα, Q - Φορτία[Τρέχον Όχημα])
39. Αν Ποσότητα > 0:
40. Διαδρομές[Τρέχον Όχημα] = append(Υποψήφιοι Πελάτες[i])
41. Φορτία[Τρέχον Όχημα] = Φορτία[Τρέχον Όχημα + Ποσότητα]
42. Απόθεμα Προμηθευτή = Απόθεμα Προμηθευτή - Ποσότητα
43. Απόθεμα Πελάτη[Υποψήφιοι Πελάτες[i]] = Απόθεμα
    Πελάτη[Υποψήφιοι Πελάτες[i]] + Ποσότητα
44. Επιθυμητή Ποσότητα = Επιθυμητή Ποσότητα - Ποσότητα
45. // Δρομολόγηση
46. Για k = 1...K:
47. Αν Διαδρομές[k]:
48. Αποστάσεις = tsp2opt(Διαδρομές[k], Πίνακας Αποστάσεων)
49. Τελικό Κόστος = Τελικό Κόστος + Αποστάσεις
50. Για i = 1...N:
51. Αν Απόθεμα Πελάτη [i] >= 0:
52. Συνολικό Κόστος = Συνολικό Κόστος + (Απόθεμα Πελάτη[i] *
    Κόστος Αποθέματος[i])
53. Αλλιώς:
54. Συνολικό Κόστος = Συνολικό Κόστος + (Απόθεμα Πελάτη[i] *
    Δυναμική ποινή λ)
55. Αν Απόθεμα Προμηθευτή >= 0:
56. Συνολικό Κόστος = Συνολικό Κόστος + (|Απόθεμα Προμηθευτή[i]|
    * Κόστος Αποθέματος Προμηθευτή[i])
57. Αλλιώς:
58. Συνολικό Κόστος = Συνολικό Κόστος + (|Απόθεμα Πελάτη[i]| *
    Δυναμική ποινή λ)
59. Τέλος Αν
60. Επιστροφή Τελικό Κόστος

```

Ψευδοκώδικας 2 -2: Η συνάρτηση καταλληλότητας (fitness function)

Όπως αναφέρθηκε προηγουμένως η επιλογή για τις τμηματικές παραδώσεις, η επιβολή δυναμικής ποινής αλλά και η περίπτωση στοχευμένης έλλειψης είναι λύσεις που έχουν καταγραφεί στην πρόσφατη βιβλιογραφία.

Η έρευνα των Coelho και Laporte (2012, 2013) [11, 12] πάνω στην επίλυση διαφόρων προβλημάτων IRP αναφέρει πως οι αυστηροί περιορισμοί οδηγούν σε ακραία υπολογιστική

πολυπλοκότητα, που ακόμα και οι πιο βελτιωμένοι αλγόριθμοι επίλυσης αδυνατούν να τα επιλύσουν. Η προσωρινή ανοχή σε μη εφικτές λύσεις (πχ. ελλείψεις) την καθιστά απαραίτητη προϋπόθεση για την επιτυχία των μετα-ευρετικών αλγορίθμων.

Επιπλέον οι Bertazzi et al. (2019) [3] για την επίλυση πιο σύνθετων προβλημάτων (όπως πχ. Multi-Depot IRP), επιβεβαιώνουν ότι η χαλάρωση μεταβλητών δρομολόγησης και αποθεμάτων επιτρέπει στους αλγόριθμους επίλυσης να πλοηγηθούν πιο αποτελεσματικά σε χώρους αναζήτησης εφικτών λύσεων. Όπως θα δούμε και παρακάτω στην έρευνά μας, έχουμε αναιρέσει τον περιορισμό για τα οχήματα να ταξιδεύουν γεμάτα. Εάν υπάρχει υπόλοιπο ποσότητας, μικρότερο της χωρητικότητας του οχήματος, που πρέπει να διανεμηθεί και υπάρχει διαθέσιμο όχημα, τότε επιτρέπεται το όχημα αυτό να μεταφέρει την υπόλοιπη ποσότητα.

Επίσης, για τα προβλήματα που αφορούν την εφοδιαστική αλυσίδα γενικότερα (που σε αυτά ανήκει και το IRP) η ενσωμάτωση μηχανισμών ποινής σε πληθυσμιακούς αλγορίθμους αποτελεί την πλέον ενδεδειγμένη μέθοδο [21].

2.11 Η Στρατηγική του Κυλιόμενου Ορίζοντα

Στην περιγραφή των βημάτων του αλγορίθμου της fitness function κάναμε μια αναφορά για την στρατηγική του κυλιόμενου ορίζοντα/ παραθύρου. Η υιοθέτηση αυτή επιτρέπει στον αλγόριθμο να εστιάσει σε ένα μικρότερο χρονικό διάστημα, αντί σε ολόκληρη την περίοδο T , και να λάβει υπόψη την ζήτηση στο διάστημα αυτό. Όπως φαίνεται και στον αντίστοιχο ψευδοκώδικα του αλγορίθμου λαμβάνεται η ζήτηση της τρέχουσας περιόδου t καθώς και των επόμενων περιόδων $t+1$, $t+2$, $t+3$. Ο τρόπος αυτός έχει αποδειχθεί ότι μπορεί να προσφέρει ακριβείς και εφικτές λύσεις [10].

Η εστίαση αυστηρά στην κάλυψη της ζήτησης στην τρέχουσα περίοδο, ειδικά σε περιβάλλοντα με διαφορετικούς ρυθμούς κατανάλωσης, είναι βέβαιο ότι θα οδηγήσει σε ελλείψεις στις αμέσως επόμενες περιόδους [20].

Ένα άλλο γεγονός που συμβαίνει είναι ότι χρησιμοποιώντας αυτήν την στρατηγική για τον προγραμματισμό των παραδόσεων, δεν αποσκοπεί απλώς στην αποφυγή της άμεσης εξάντλησης του αποθέματος, αλλά και στην μεγιστοποίηση της αποδοτικότητας των

δρομολογίων. Με αυτόν τον τρόπο η προκαταβολική μεταφορά μεγαλύτερων ποσοτήτων (εφόσον υπάρχει διαθέσιμος χώρος) εξοικονομεί μελλοντικά δρομολόγια [1].

3. Επίλυση και Βελτιστοποίηση του IRP

3.1 Δεδομένα Εισόδου

Οι Skalnaes et al. [26] το 2023 δημοσίευσαν μια συλλογή από περιπτώσεις (*instances*) που απεικονίζουν σχεδόν πραγματικά IRPs. Σε αυτά τα 270 παραδείγματα παρουσιάζονται περιπτώσεις που περιλαμβάνουν δίκτυα διανομής που αποτελούνται από 10 έως 200 πελάτες με τοπική έως διευρυμένη γεωγραφική κατανομή, με δομημένο πλάνο ζήτησης ανά χρονική περίοδο και εξυπηρετούνται από οχήματα με διάφορες και δεδομένες χωρητικότητες. Ενδεικτικά, ένα από τα παραδείγματα παρουσιάζεται ως εξής:

nVehicles: 10 nDepots: 1 nCustomers: 10 nPeriods: 6																									
VEHICLE_TYPE		FLEET_SIZE				CAPACITY																			
1		10				8																			
DEPOT	0	X	195	471	Y	INV_COST	0	87	INITIAL_INV	MIN_LEVEL_INV	0	MAX_LEVEL_INV	394	PRODUCTION_0	67	PRODUCTION_1	67	PRODUCTION_2	67	PRODUCTION_3	67	PRODUCTION_4	67	PRODUCTION_5	67
CUSTOMER	1	X	297	286	Y	INV_COST	0.07	8	INITIAL_INV	MIN_LEVEL_INV	0	MAX_LEVEL_INV	25	DEMAND_0	4	DEMAND_1	5	DEMAND_2	6	DEMAND_3	8	DEMAND_4	6	DEMAND_5	6
	2		337	334			0.3	7			0		22	6		7		11		9		9		7	
	3		342	280			0.25	7			0		17	3		5		5		7		5		4	
	4		320	275			0.23	1			0		35	5		5		6		7		9		7	
	5		323	276			0.17	12			0		35	6		7		10		13		9		8	
	6		230	500			0.03	2			0		8	3		4		4		6		6		5	
	7		258	494			0.22	3			0		18	5		9		9		10		9		8	
	8		235	485			0.03	4			0		18	4		4		5		7		8		7	
	9		231	482			0.18	3			0		20	4		5		6		7		8		5	
	10		217	499			0.06	4			0		15	4		5		8		8		7		8	
COST_MATRIX		0	1	2	3	4	5	6	7	8	9	10													
	0																								
	1	106																							
	2	99	32																						
	3	121	23	28																					
	4	117	13	31	12																				
	5	117	14	30	10	2																			
	6	23	113	99	124	122	122																		
	7	34	106	90	115	114	114	15																	
	8	22	105	92	116	114	114	8	13																
	9	19	104	92	116	113	113	10	15	3															
	10	18	114	103	127	124	124	7	21	12	12														

Εικόνα 3-1: Παράδειγμα περίπτωσης IRP

Από τα παραπάνω μπορούν να αντληθούν δεδομένα όπως:

- Πλήθος Οχημάτων (*nVehicles*),
- Πλήθος Σημείων Αναχώρησης (Αμαξοστάσια) ή Προμηθευτών (*nDepots*),
- Πλήθος Πελατών (*nCustomers*),
- Χρονικός Ορίζοντας σε ημέρες (*nPeriods*),
- Τύπος Οχήματος (*VEHICLE_TYPE*), Μέγεθος Στόλου (*FLEET_SIZE*) και Χωρητικότητα Οχήματος (*CAPACITY*),
- Συντεταγμένες Προμηθευτή (*DEPOT X Y*), Κόστος Διατήρησης Αποθέματος στον προμηθευτή (*INV_COST*), Αρχικό Απόθεμα Προμηθευτή (*INITIAL_INV*), Ελάχιστο (*MIN_LEVEL_INV*) και Μέγιστο Ύψος Αποθέματος στον Προμηθευτή (*MAX_LEVEL_INV*) και Ημερήσια Παραγωγή στον Προμηθευτή (*PRODUCTION_t*).

- Συντεταγμένες Πελατών (*CUSTOMER X Y*), Αρχικό Απόθεμα Πελάτη (*INITIAL_INV*), Κόστος Διατήρησης Αποθέματος στον Πελάτη (*INV_COST*), Ελάχιστο (*MIN_LEVEL_INV*) και Μέγιστο Ύψος Αποθέματος στον Πελάτη (*MAX_LEVEL_INV*), Ημερήσια Ζήτηση του Πελάτη (*DEMAND_t*).
- Διάγραμμα Κόστους (*COST_MATRIX*) όπου παρουσιάζεται το κόστος των αποστάσεων μεταξύ Προμηθευτή ($0, 0$) και Πελάτων (i, j) που έχουν υπολογιστεί με την μέθοδο της Ευκλείδειας Απόστασης.

Τα παραδείγματα αυτά έχουν κάποια χαρακτηριστικά γνωρίσματα, όπως:

- Οι περιοχές που γίνονται οι διανομές χωρίζονται σε Αστικές και Αγροτικές με αντίστοιχη τροποποίηση στο μήκος διαδρομής (κάθε μονάδα απόστασης ισούται με 0,1 χλμ. στις Αστικές περιοχές και 0,5 χλμ. στις Αγροτικές). Για λόγους απλότητας όμως κάθε μονάδα απόστασης έχει κόστος 1 μονάδα.
- Η γεωγραφική κατανομή των πελατών είναι Τυχαία, Ομαδοποιημένη και Τυχαία-Ομαδοποιημένη.
- Οι πελάτες κατανέμονται σε σύνολα των 10, 25, 50, 100, 200 ανά περίπτωση.
- Οι τύποι των οχημάτων χωρίζονται σε Μικρά Φορτηγά (Τύπου 1), Φορτηγά (Τύπου 2), Επικαθήμενα (Τύπου 3) με αντίστοιχες χωρητικότητες των 8, 18 και 38 τεμαχίων, όπου ως μονάδα μέτρησης των τεμαχίων θεωρείται η Ευρωπαϊκή Παλέτα.

Ο χρονικός ορίζοντας ορίζεται σε 6, 9, 12 χρονικές περιόδους, όπου θεωρούμε πως είναι ημερήσιες.

Τα χαρακτηριστικά γνωρίσματα των παραδειγμάτων συνοψίζονται στον παρακάτω πίνακα:

Είδος περιοχής	Κατανομή	Πλήθος Πελατών	Τύποι Οχημάτων (Τύπος)	Χωρητικότητες Οχημάτων (Τύπος)	Χρονικές Περίοδοι
Αστική	Τυχαία	10	Μικρά Φορτηγά (1)	8 (1)	6
	Ομαδοποιημένη	25	Φορτηγά (2)	18 (2)	9
		50			
Αγροτική	Τυχαία-	100	Επικαθήμενα (3)	38 (3)	12
	Ομαδοποιημένη	200			

Πίνακας 3-1: Τα χαρακτηριστικά γνωρίσματα των παραδειγμάτων IRP

3.2 Προγραμματιστικό Περιβάλλον και Υλοποίηση στην Python

Για την επίλυση του προβλήματος IRP κρίθηκε ως καλύτερη επιλογή η προγραμματιστική υλοποίηση του στη γλώσσα προγραμματισμού Python. Η συγκεκριμένη γλώσσα προγραμματισμού υπερτερεί σε πληθώρα χαρακτηριστικών που την καθιστούν ιδανική για την ανάπτυξη μετα-ευρετικών αλγορίθμων και την επίλυση σύγχρονων προβλημάτων βελτιστοποίησης.

Η απλότητα και η καθαρότητα της σύνταξής της επιτρέπουν την αποτύπωση των περιορισμών και των μεθόδων που παρουσιάζονται στους αλγορίθμους με πολύ απλή λεκτική περιγραφή. Συνεπώς η αποτύπωση του POA αλλά και της fitness function έγινε με ακρίβεια και χωρίς την περιττή πολυπλοκότητα που συναντάται σε άλλες γλώσσες προγραμματισμού χαμηλότερου επιπέδου.

Ο τρόπος διαχείρισης των περιορισμών και των μεθόδων γίνεται με την χρήση συναρτήσεων (*def*) όπου εκεί μπορούμε να αναπαραστήσουμε με εξαιρετική ευελιξία τα ορίσματα και τις μεταβλητές που τροφοδοτούν τους αλγορίθμους επίλυσης και βελτιστοποίησης του IRP. Επίσης με αυτόν τον τρόπο μπορούμε να επαληθεύσουμε την ορθότητα της λογικής του κώδικα σε κάθε επιμέρους βήμα των αλγορίθμων.

Ένας ακόμα σημαντικός παράγοντας που λειτούργησε ως κριτήριο επιλογής της συγκεκριμένης γλώσσας προγραμματισμού είναι ότι η Python διαθέτει τεράστιο πλήθος πακέτων και βιβλιοθηκών που μπορούν να συνεισφέρουν σε μεγάλο βαθμό στην επίλυση πολύπλοκων προβλημάτων, όπως είναι το IRP. Για παράδειγμα στον κώδικά μας κάνουμε

την χρήση της *NumPy* όπου με τα εργαλεία που προσφέρει μπορούμε να κάνουμε πράξεις πινάκων και γρήγορους διανυσματικούς υπολογισμούς αποστάσεων. Επίσης με την χρήση της βιβλιοθήκης *Shapely* ενσωματώσαμε στην *fitness function* την ιδιότητα να συγκρίνει την εγγύτητα των εικονικών με τους πραγματικούς κόμβους του δικτύου.

Τέλος, μια ακόμη ιδιότητα που διαθέτει η Python είναι η δυνατότητα να διαβάζει και να αντλεί πολύ εύκολα αρχεία κειμένου. Δεδομένου ότι τα παραδείγματα που χρησιμοποιήσαμε ως δεδομένα είναι απλά αρχεία κειμένου (.txt), η Python μας επέτρεψε να δημιουργήσουμε μηχανισμούς εισαγωγής (*parsers*) για να αντληθούν δομημένα και αυτοματοποιημένα όλα τα κρίσιμα δεδομένα που απαιτήθηκαν για την έρευνά μας.

3.3 Τεχνικές Προδιαγραφές Υπολογιστικού Συστήματος

Για την εκτέλεση του προγράμματος επίλυσης και την διεξαγωγή των υπολογιστικών πειραμάτων, χρησιμοποιήθηκε ένα σύγχρονο σταθερό υπολογιστικό σύστημα ώστε να διασφαλιστεί η ταχύτητα και η απρόσκοπτη αξιολόγηση των χιλιάδων υποψήφιων λύσεων σε λογικούς χρόνους εκτέλεσης.

Τα τεχνικά χαρακτηριστικά του συστήματος είναι τα παρακάτω:

- Επεξεργαστής: Intel Core i5-10600KF CPU,
- Ταχύτητα επεξεργαστή: 4.10GHz,
- Μνήμη: 32 GB,
- Λειτουργικό Σύστημα: Windows 11 Pro Edition, Version 25H2.

3.4 Διαχείριση Υπολογιστικού Φόρτου και Ελεγχόμενη Παράλληλη Επεξεργασία

Για να μπορέσουμε να αξιολογήσουμε τον μεγάλο αριθμό λύσεων (πελεκάνων) ανά επανάληψη αλλά και να περιορίσουμε την δυνατότητα των εσωτερικών βιβλιοθηκών (όπως είναι η *NumPy*) της Python, οι οποίες εκτελούν αυτόματα τις πράξεις τους πολυνηματικά (*multi-threaded*) [17], χρειάστηκε να γίνει μια αυστηρή παραμετροποίηση σε επίπεδο λειτουργικού συστήματος.

Η Python επιτρέπει την παραμετροποίηση αυτή μέσω των μεταβλητών περιβάλλοντος (*os.environ*) ώστε να αναγκάσουμε όλες τις υποκείμενες βιβλιοθήκες υπολογισμού (*OpenBLAS*, *MKL*, *NumExpr* κ.ά.) να χρησιμοποιήσουν ένα νήμα ανά επεξεργασία. Η

παραμετροποίηση φαίνεται στο παρακάτω απόσπασμα από το πρόγραμμα που εκτελέσαμε για την έρευνά μας:

```
import os
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["VECLIB_MAXIMUM_THREADS"] = "1"
os.environ["NUMEXPR_NUM_THREADS"] = "1"
```

Κώδικας 3 - 1: Απόσπασμα του κώδικα που αφορά τις παραμετροποιήσεις επεξεργασίας.

Με αυτόν τον τρόπο εξασφαλίστηκε η θερμική και η υπολογιστική σταθερότητα σε επίπεδο νημάτων αλλά παράλληλα και η προστασία του ίδιου του επεξεργαστή, καθώς κληθήκαμε να διαχειριστούμε την τεράστια υπολογιστική πολυπλοκότητα της ίδιας της συνάρτησης καταλληλότητας.

Με δεδομένο το γεγονός ότι η αξιολόγηση κάθε μεμονωμένης υποψήφιας λύσης απαιτεί τη διαχείριση εικονικών κόμβων, πολύπλοκες γεωμετρικές συσχετίσεις και επαναληπτικές κλήσεις του ευρετικού αλγορίθμου 2-opt για επίλυση πολλαπλών υπο-προβλημάτων TSP, ο αθροιστικός φόρτος για έναν ολόκληρο πληθυσμό επί χιλιάδες επαναλήψεις είναι δυσθεώρητος [16].

Για να είναι εφικτή η επίλυση αυτής της πολυπλοκότητας σε ρεαλιστικούς χρόνους, χρησιμοποιήθηκε η κλάση *ProcessPoolExecutor* από τη βιβλιοθήκη *concurrent.futures* όπου τέθηκε ένα αυστηρό όριο στον μέγιστο αριθμό πυρήνων ή εργατών (*Μέγιστοι Εργάτες – Max Workers*) που θα χρησιμοποιηθεί για κάθε επανάληψη εκτέλεσης. Με αυτόν τον τρόπο η βαριά υπολογιστική διαδικασία αξιολόγησης των υποψήφιων λύσεων κατανέμεται με ασφάλεια στους διαθέσιμους πυρήνες, αποκομίζοντας σημαντικά οφέλη όπως η δραστική επιτάχυνση του αλγορίθμου POA αλλά και η διατήρηση της θερμοκρασίας της CPU σε ασφαλή επίπεδα.

3.5 Ρύθμιση Παραμέτρων του POA

Στα περισσότερα πειράματα για την βελτιστοποίηση λύσεων σε υπαρκτά προβλήματα που έχουν γίνει μέχρι στιγμής [30], αλλά και στις δοκιμές επιδόσεων του POA, προτείνονται διαδοχικά τα εξής:

- Ορισμός πληθυσμού 20, 30, 50 και 100 πελεκάνοι (*agents*).
- Ορισμός επαναλήψεων αλγορίθμου 100, 500, 800, 1000 φορές.

Κατά την αρχική επεξεργασία των instances, ειδικά εκείνων με τα μεγάλα χαρακτηριστικά γνωρίσματα και με την μέγιστη παραμετροποίηση διαπιστώθηκε ότι χρειαζόταν πολύ μεγάλος χρόνος εκτέλεσης. Ωστόσο ο αλγόριθμος είχε καταλήξει στην βέλτιστη λύση από τις μισές σχεδόν επαναλήψεις, ενώ ένα άλλο γεγονός που διαπιστώθηκε είναι ότι η χρήση 100 agents δεν διαφοροποιείται στατιστικά από τους υπόλοιπους πληθυσμούς. Οπότε ο υπολειπόμενος υπολογιστικός χρόνος ήταν χαμένος χρόνος [15].

Για να μπορέσουμε να ρυθμίσουμε πιο αποδοτικά τις παραμέτρους του αλγορίθμου, καταγράφηκαν σε ποιες επαναλήψεις και μετά ο αλγόριθμος παρουσιάζει απόλυτη σύγκλιση (δηλαδή το κόστος παραμένει σταθερό). Διαπιστώθηκε ότι μέχρι τις 200 επαναλήψεις ο αλγόριθμος είχε τη μέγιστη βελτίωση ενώ το σημείο σύγκλισης ήταν μεταξύ 350 και 450 επαναλήψεων. Συνεπώς η συνέχιση εκτέλεσης σε περισσότερες των 500 επαναλήψεων (πχ. 800, 1000) δεν απέφερε στατιστικά σημαντική βελτίωση, ενώ ο αλγόριθμος εγκλωβιζόταν σε τοπικά βέλτιστα [8].

Μια ακόμα παρατήρηση ήταν ότι με την χρήση μεγάλων αριθμών πληθυσμού (πχ. 80, 100) δεν αναβάθμισε ουσιαστικά την αποδοτικότητα του αλγορίθμου. Η χρήση όμως ενός μεσαίου πλήθους πελεκάνων (30 και 50) εξασφάλισε επαρκή ποικιλομορφία στα αποτελέσματα, ειδικά στην φάση της εξερεύνησης [30].

Ένα ακόμη γεγονός είναι ότι επειδή ο αλγόριθμος εκτελείται στο προαναφερόμενο υπολογιστικό σύστημα η χρήση παραμετροποίησης με μεγάλα μεγέθη πληθυσμού και επαναλήψεων έκανε τον κώδικα να τρέχει πιο αργά, χωρίς να υπάρχει το αντίστοιχο όφελος στην βελτίωση των αποτελεσμάτων. Το αποτέλεσμα ήταν ότι ο υπολογιστικός χρόνος ουσιαστικά ήταν χαμένος χρόνος και να «καταπονεί» στην ουσία το σύστημα, παρόλη την διαχείριση της παράλληλης επεξεργασίας που αναφέρθηκε προηγουμένως.

Ο αλγόριθμος παρουσίασε τα πιο βελτιωμένα αποτελέσματα όταν ο πληθυσμός ορίστηκε σε 30, με μέγιστο αριθμό επαναλήψεων τις 500, για τα instances με αριθμό πελατών μικρότερο του 50. Για τα instances που αφορούν αριθμό πελάτων μεγαλύτερο από το 50 τα βελτιωμένα αποτελέσματα προκύπτουν όταν ο πληθυσμός ορίστηκε σε 50, με μέγιστο

αριθμό επαναλήψεων τις 800. Αυτές είναι και οι παράμετροι για τις οποίες αρχικοποιήθηκε ο POA για την εκτέλεση των μετρήσεων στα αντίστοιχα instances.

3.6 Συλλογή Στατιστικών Δεδομένων

Επειδή η επεξεργασία των instances από το πρόγραμμα θα γίνεται επαναλαμβανόμενα, για την ολοκληρωμένη αξιολόγηση της απόδοσης του POA στο τέλος των επαναλήψεων, καταγράφονται τα εξής μεγέθη: το Καλύτερο Κόστος (*Best Cost*) που αντιπροσωπεύει τη βέλτιστη λύση που ανακαλύφθηκε, το Χειρότερο Κόστος (*Worst Cost*), το Μέσο Κόστος (*Mean Cost*) όλων των εκτελέσεων, η Τυπική Απόκλιση (*Standard Deviation*) από το μέσο κόστος, η Διάμεσος (*Median*) του κόστους, καθώς και η μέση Χρονική Διάρκεια της εκτέλεσης.

Επιπλέον, για την αρτιότερη στατιστική τεκμηρίωση των αποτελεσμάτων του αλγορίθμου POA, καταγράφονται δύο επιπλέον κρίσιμοι δείκτες απόδοσης:

- Συντελεστής Μεταβλητότητας (*Coefficient of Variation - CV%*): Ένα χαμηλό ποσοστό *CV%* υποδεικνύει ότι ο αλγόριθμος είναι εξαιρετικά αξιόπιστος, παράγοντας σταθερά λύσεις γύρω από το μέσο κόστος ανεξάρτητα από την τυχαία αρχικοποίησή του.
- Ποσοστιαίο Χάσμα (*Gap %*): Μέσω αυτού του μεγέθους αξιολογείται η ποσοστιαία απόσταση της εκάστοτε λύσης από τον μέσο όρο (ή από την καλύτερη γνωστή λύση) [26]. Ο υπολογισμός του *Gap%* προσφέρει μια σαφή και συγκρίσιμη εικόνα για την ποιότητα της παραγόμενης λύσης σε σχέση με τα γενικά πρότυπα απόδοσης.

Τα δεδομένα αυτά με την αντίστοιχη χρήση της βιβλιοθήκης της Python (*pandas*) εξάγονται σε υπολογιστικά φύλλα Excel. Με αυτόν τον τρόπο η συγκέντρωση των αποτελεσμάτων εξυπηρετεί στο να είναι προσβάσιμα τα αποτελέσματα για αξιολόγηση και περαιτέρω έρευνα.

3.7 Εμφάνιση Λύσης

Επιπλέον της στατιστικής καταγραφής, το πρόγραμμα κατά την ολοκλήρωση της επεξεργασίας εξάγει την τελική (βέλτιστη) λύση μέσω μια λεπτομερούς αναφοράς κειμένου σε αρχείο της μορφής .txt. Σε αυτά τα αρχεία παρουσιάζονται σε διακριτά μέρη, χωρισμένα ανά ημέρα, τα εξής στοιχεία:

- Επισκόπηση Προμηθευτή και Σειρά Προτεραιότητας: Στην αρχή αποτυπώνεται το τρέχον διαθέσιμο απόθεμα του προμηθευτή (*Depot Inventory*) σε σχέση με τη μέγιστη χωρητικότητά του. Στην συνέχεια παρουσιάζεται η προτεινόμενη Σειρά Προτεραιότητας (*Suggested Priority Sequence*), η οποία έχει προκύψει από την διαδικασία εύρεσης της βέλτιστης λύσης από τον POA.
- Αναλυτική Δρομολόγηση Οχημάτων (*Routes*): Για κάθε όχημα που χρησιμοποιήθηκε καταγράφεται η διαδρομή του από το σημείο αφετηρίας (*Depot, Κόμβος 0*), τους πελάτες που θα επισκεφθεί και τέλος την επιστροφή του στην αφετηρία (πχ. Path: 0 -> 25 -> 2 -> 12 -> 4 -> 0). Επίσης αναγράφεται το φορτίο σε σχέση με το ωφέλιμο φορτίο του κάθε οχήματος, το κόστος της διαδρομής, καθώς και τις ακριβείς ποσότητες που παραδόθηκαν στον πελάτη. Τα οχήματα που δεν χρησιμοποιήθηκαν επισημαίνονται ως «Δεν χρησιμοποιήθηκαν (*Not Used*)».
- Κατάσταση Αποθεμάτων Πελατών: Στο τέλος της κάθε ημέρας, εμφανίζεται ένας συγκεντρωτικός πίνακας που αποτυπώνει την κατάσταση των αποθεμάτων του κάθε πελάτη. Αναγράφεται το αρχικό ημερήσιο απόθεμα (*Initial*), η ημερήσια ζήτηση (*Demand*), η ποσότητα που παραδόθηκε (*Delivered*), το εναπομείναν απόθεμα (*New Stock*) και το ανάλογο κόστος διατήρησης αποθέματος (*Inv. Cost*).
- Σύστημα Προειδοποίησης Έλλειψης: Σε περίπτωση που υπάρχει κίνδυνος έλλειψης ή έχει προκύψει έλλειψη σε πελάτη κατά την διάρκεια της χρονική περιόδου, τότε αυτό επισημαίνεται με την αντίστοιχη ένδειξη (<!=> WARNING: Infeasible delivery for Customer 21 at Day 3 / STOCKOUT!). Η ένδειξη αυτή αποδεικνύει παράλληλα την ενεργοποίηση του μηχανισμού επιβολής δυναμικής ποινής σε περίπτωση έλλειψης, που αναλύθηκε σε προηγούμενο κεφάλαιο.
- Τελική Σύνοψη Κόστους: Μετά την τελευταία ημέρα του χρονικού ορίζοντα, εμφανίζονται διαδοχικά το συνολικό κόστος μεταφοράς (*Total Routing Cost*), το συνολικό κόστος διατήρησης αποθέματος μαζί με το συνολικό κόστος των ποινών, εάν έχει προκύψει (*Total Holding Cost & Penalties*). Στο τέλος αναγράφεται το συνολικό κόστος που αντιπροσωπεύει το τελικό και βέλτιστο κόστος της λύσης (*TOTAL COST*).

Παρακάτω παρουσιάζονται δυο ενδεικτικά στιγμιότυπα της παραπάνω αναφοράς που αφορούν την πρώτη και την τελευταία ημέρα ενός αρχείου που επεξεργαστήκαμε (με χαρακτηριστικά $N=25$, $Q=38$, $T=6$, Αγροτική και Τυχαία Κατανομή) και ένα στιγμιότυπο που αφορά την 3^η ημέρα ενός αρχείου (με χαρακτηριστικά $N=25$, $Q=38$, $T=6$, Αγροτική και Ομαδοποιημένη Κατανομή) με την προειδοποιητική ένδειξη έλλειψης:

```
=====
DAY 1:
=====
Depot Inventory: 313.00/904.0
Suggested Priority Sequence: [1, 7, 4, 20, 22, 13, 15, 14, 11, 19, 24, 6, 3, 10, 17, 8, 23, 2, 18, 21, 16, 5, 9, 12]]

** ROUTES **
Vehicle 1:
|--> Path: 0 -> 1 -> 7 -> 4 -> 20 -> 22 -> 13 -> 15 -> 0
|--> Load: 38.00/38 | Cost: 1097.00
|--> Drops: Customer 1 (11.00), Customer 7 (6.00), Customer 4 (1.00), Customer 20 (7.00), Customer 22 (5.00), Customer 13 (4.00), Customer 15 (8.00)
Vehicle 2:
|--> Path: 0 -> 15 -> 11 -> 19 -> 24 -> 3 -> 10 -> 17 -> 23 -> 18 -> 0
|--> Load: 38.00/38 | Cost: 1279.00
|--> Drops: Customer 15 (8.00), Customer 11 (7.00), Customer 19 (3.00), Customer 24 (4.00), Customer 3 (2.00), Customer 10 (6.00), Customer 17 (6.00), Customer 23 (2.00), Customer 18 (5.00)
Vehicle 3:
|--> Path: 0 -> 18 -> 21 -> 16 -> 5 -> 9 -> 12 -> 0
|--> Load: 27.00/38 | Cost: 639.00
|--> Drops: Customer 18 (5.00), Customer 21 (9.00), Customer 16 (2.00), Customer 5 (5.00), Customer 9 (4.00), Customer 12 (6.00)
Vehicle 4: Not used
Vehicle 5: Not used

** INVENTORY STATUS (End of Day) **
* Customer | Initial | Demand | Delivered | New Stock | Inv.Cost *
-----
* 1 | 1.00 | 5.00 | 11.00 | 7.00 | 0.07 *
* 2 | 6.00 | 3.00 | 0.00 | 3.00 | 0.06 *
* 3 | 5.00 | 3.00 | 2.00 | 4.00 | 0.44 *
* 4 | 8.00 | 4.00 | 1.00 | 5.00 | 0.85 *
* 5 | 8.00 | 6.00 | 5.00 | 7.00 | 1.68 *
* 6 | 11.00 | 5.00 | 0.00 | 6.00 | 1.74 *
* 7 | 5.00 | 6.00 | 6.00 | 5.00 | 0.35 *
* 8 | 7.00 | 4.00 | 0.00 | 3.00 | 0.57 *
* 9 | 4.00 | 4.00 | 4.00 | 4.00 | 0.00 *
* 10 | 3.00 | 4.00 | 6.00 | 5.00 | 0.00 *
* 11 | 1.00 | 3.00 | 7.00 | 5.00 | 0.70 *
* 12 | 4.00 | 5.00 | 6.00 | 5.00 | 0.00 *
* 13 | 7.00 | 6.00 | 4.00 | 5.00 | 0.65 *
* 14 | 14.00 | 7.00 | 0.00 | 7.00 | 1.89 *
* 15 | 6.00 | 7.00 | 8.00 | 7.00 | 1.47 *
* 16 | 8.00 | 5.00 | 2.00 | 5.00 | 1.10 *
* 17 | 3.00 | 5.00 | 6.00 | 4.00 | 0.32 *
* 18 | 5.00 | 5.00 | 5.00 | 5.00 | 1.05 *
* 19 | 5.00 | 4.00 | 3.00 | 4.00 | 0.00 *
* 20 | 8.00 | 8.00 | 7.00 | 7.00 | 0.42 *
* 21 | 6.00 | 7.00 | 9.00 | 8.00 | 1.76 *
* 22 | 2.00 | 4.00 | 5.00 | 3.00 | 0.15 *
* 23 | 7.00 | 4.00 | 2.00 | 5.00 | 1.20 *
* 24 | 4.00 | 4.00 | 4.00 | 4.00 | 0.60 *
* 25 | 11.00 | 5.00 | 0.00 | 6.00 | 1.74 *
```

Εικόνα 3-2: Στιγμιότυπο Αναφοράς από το αρχείο με $N=25$, $Q=38$, $T=6$: Ημέρα 1^η (Αγροτική και Τυχαία Κατανομή)

```
=====
DAY 6:
=====
Depot Inventory: 321.00/904.0
Suggested Priority Sequence: [7, 4, 22, 20, 11, 15, 1, 13, 25, 14, 19, 24, 6, 3, 10, 17, 8, 23, 2, 18, 5, 21, 16, 9, 12]

** ROUTES **
Vehicle 1:
|--> Path: 0 -> 11 -> 25 -> 10 -> 0
|--> Load: 12.00/38 | Cost: 656.00
|--> Drops: Customer 11 (3.00), Customer 25 (7.00), Customer 10 (2.00)
Vehicle 2: Not used
Vehicle 3: Not used
Vehicle 4: Not used
Vehicle 5: Not used

** INVENTORY STATUS (End of Day) **
* Customer | Initial | Demand | Delivered | New Stock | Inv.Cost | *
=====
* 1 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 2 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | *
* 3 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 4 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 5 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 6 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | *
* 7 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | *
* 8 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 9 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 10 | 3.00 | 5.00 | 2.00 | 0.00 | 0.00 | *
* 11 | 5.00 | 8.00 | 3.00 | 0.00 | 0.00 | *
* 12 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | *
* 13 | 9.00 | 9.00 | 0.00 | 0.00 | 0.00 | *
* 14 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 15 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 16 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | *
* 17 | 7.00 | 7.00 | 0.00 | 0.00 | 0.00 | *
* 18 | 4.00 | 4.00 | 0.00 | 0.00 | 0.00 | *
* 19 | 4.00 | 4.00 | 0.00 | 0.00 | 0.00 | *
* 20 | 11.00 | 11.00 | 0.00 | 0.00 | 0.00 | *
* 21 | 8.00 | 8.00 | 0.00 | 0.00 | 0.00 | *
* 22 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 23 | 6.00 | 6.00 | 0.00 | 0.00 | 0.00 | *
* 24 | 5.00 | 5.00 | 0.00 | 0.00 | 0.00 | *
* 25 | 3.00 | 10.00 | 7.00 | 0.00 | 0.00 | *
=====
Total Routing Cost: 18883.00
Total Holding Cost & Penalties: 112.56
TOTAL COST: 18995.56
```

Εικόνα 3-3: Στιγμιότυπο Αναφοράς από το αρχείο με N=25, Q=38, T=6: Ημέρα 6η (Αγροτική και Τυχαία Κατανομή)

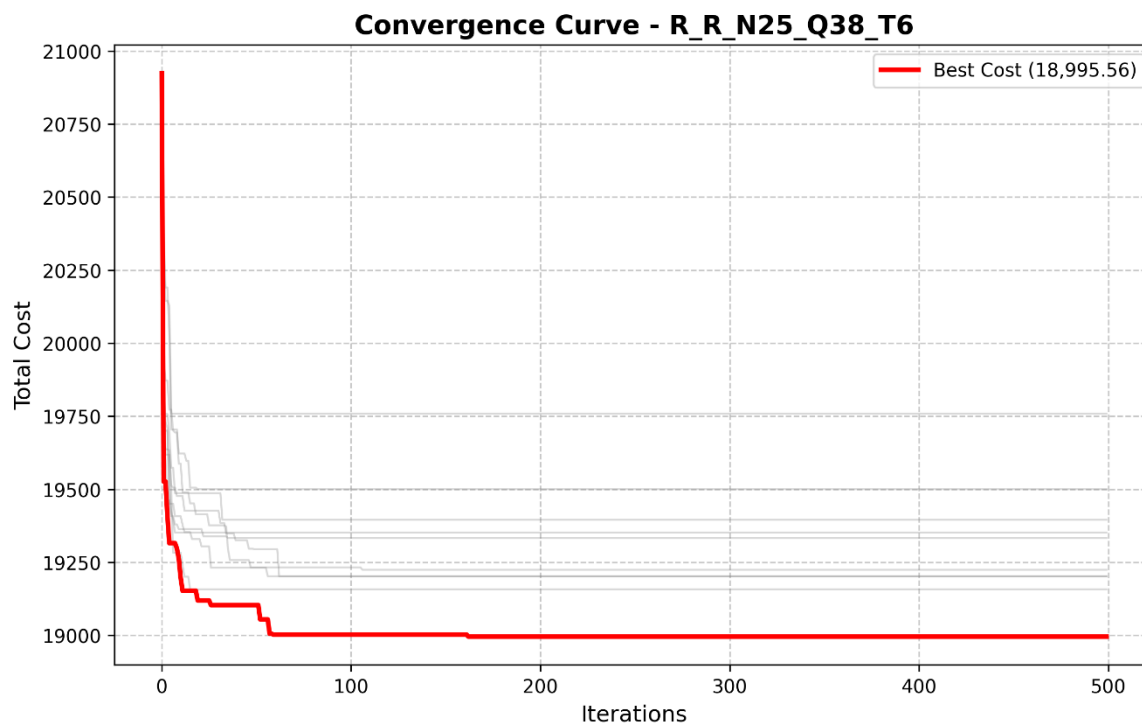
```
=====
DAY 3:
=====
Depot Inventory: 449.00/1006.0
Suggested Priority Sequence: [20, 13, 23, 17, 22, 15, 19, 5, 18, 10, 21, 14, 16, 24, 11, 3, 8, 4, 12, 1, 7, 9, 2, 6, 25]
<!-- WARNING: Infeasible delivery for Customer 25 on Day 3 / STOCKOUT!

** ROUTES **
Vehicle 1:
|--> Path: 0 -> 20 -> 13 -> 23 -> 17 -> 22 -> 15 -> 0
|--> Load: 38.00/38 | Cost: 760.00
|--> Drops: Customer 20 (5.00), Customer 13 (4.00), Customer 23 (8.00), Customer 17 (6.00), Customer 22 (9.00), Customer 15 (10.00)
Vehicle 2:
|--> Path: 0 -> 15 -> 19 -> 5 -> 18 -> 10 -> 21 -> 0
|--> Load: 38.00/38 | Cost: 810.00
|--> Drops: Customer 15 (10.00), Customer 19 (12.00), Customer 5 (6.00), Customer 18 (7.00), Customer 10 (6.00), Customer 21 (12.00)
Vehicle 3:
|--> Path: 0 -> 21 -> 14 -> 16 -> 24 -> 11 -> 0
|--> Load: 38.00/38 | Cost: 847.00
|--> Drops: Customer 21 (12.00), Customer 14 (6.00), Customer 16 (12.00), Customer 24 (9.00), Customer 11 (7.00)
Vehicle 4:
|--> Path: 0 -> 11 -> 3 -> 8 -> 4 -> 12 -> 0
|--> Load: 38.00/38 | Cost: 862.00
|--> Drops: Customer 11 (7.00), Customer 3 (11.00), Customer 8 (6.00), Customer 4 (7.00), Customer 12 (12.00)
Vehicle 5:
|--> Path: 0 -> 12 -> 1 -> 7 -> 9 -> 2 -> 6 -> 0
|--> Load: 38.00/38 | Cost: 897.00
|--> Drops: Customer 12 (12.00), Customer 1 (6.00), Customer 7 (5.00), Customer 9 (10.00), Customer 2 (10.00), Customer 6 (4.00)

** INVENTORY STATUS (End of Day) **
* Customer | Initial | Demand | Delivered | New Stock | Inv.Cost | *
=====
* 1 | 5.00 | 5.00 | 6.00 | 6.00 | 0.60 | *
* 2 | 9.00 | 9.00 | 10.00 | 10.00 | 1.80 | *
* 3 | 9.00 | 9.00 | 11.00 | 11.00 | 0.11 | *
* 4 | 7.00 | 7.00 | 7.00 | 7.00 | 2.10 | *
* 5 | 6.00 | 6.00 | 6.00 | 6.00 | 1.68 | *
* 6 | 7.00 | 7.00 | 4.00 | 4.00 | 1.12 | *
* 7 | 5.00 | 6.00 | 5.00 | 4.00 | 0.08 | *
* 8 | 4.00 | 6.00 | 6.00 | 4.00 | 0.68 | *
* 9 | 8.00 | 8.00 | 10.00 | 10.00 | 0.20 | *
* 10 | 7.00 | 7.00 | 6.00 | 6.00 | 1.08 | *
* 11 | 5.00 | 5.00 | 7.00 | 7.00 | 1.68 | *
* 12 | 12.00 | 12.00 | 12.00 | 12.00 | 2.76 | *
* 13 | 4.00 | 5.00 | 4.00 | 3.00 | 0.81 | *
* 14 | 7.00 | 7.00 | 6.00 | 6.00 | 0.90 | *
* 15 | 9.00 | 9.00 | 10.00 | 10.00 | 2.30 | *
* 16 | 9.00 | 9.00 | 12.00 | 12.00 | 1.56 | *
* 17 | 6.00 | 6.00 | 6.00 | 6.00 | 1.08 | *
* 18 | 8.00 | 9.00 | 7.00 | 6.00 | 0.18 | *
* 19 | 11.00 | 11.00 | 12.00 | 12.00 | 2.40 | *
* 20 | 4.00 | 4.00 | 5.00 | 5.00 | 0.90 | *
* 21 | 12.00 | 12.00 | 12.00 | 12.00 | 1.44 | *
* 22 | 9.00 | 9.00 | 9.00 | 9.00 | 1.53 | *
* 23 | 7.00 | 7.00 | 8.00 | 8.00 | 1.84 | *
* 24 | 9.00 | 9.00 | 9.00 | 9.00 | 2.16 | *
* 25 | 4.00 | 4.00 | 0.00 | 0.00 | 0.00 | *
```

Εικόνα 3-4: Στιγμιότυπο Αναφοράς από το αρχείο με N=25, Q=38, T=6: Ημέρα 3η (Αγροτική και Ομαδοποιημένη Κατανομή), με την προειδοποιητική ένδειξη έλλειψης

Επιπλέον το πρόγραμμα, στο τέλος της επανάληψης εξάγει και μία εικόνα που περιλαμβάνει το γράφημα της σύγκλισης, όπου στον κατακόρυφο άξονα απεικονίζεται το καλύτερο συνολικό κόστος της καλύτερης λύσης που έχει ανακαλυφθεί ανά επανάληψη και στον οριζόντιο άξονα οι αριθμοί των επαναλήψεων. Με αυτόν τον τρόπο απεικονίζεται με γραφικό τρόπο η ομαλή μετάβαση του αλγορίθμου από τη φάση της εξερεύνησης (έντονη και απότομη πτώση του κόστους στις αρχικές επαναλήψεις) στη φάση της εκμετάλλευσης και τελικής σύγκλισης (σταθεροποίηση της καμπύλης σε ευθεία γραμμή). Στην παρακάτω εικόνα παρουσιάζεται ένα ενδεικτικό διάγραμμα σύγκλισης από το αρχείο που αναφέρθηκε στα προηγούμενα στιγμιότυπα:



Εικόνα 3-5: Διάγραμμα σύγκλισης από το αρχείο με $N=25$, $Q=38$, $T=6$ (Αγροτική και Τυχαία Κατανομή). Οι καμπύλες απεικονίζουν 10 διαδοχικές εκτελέσεις, όπου αυτή που παρουσιάζει το καλύτερο κόστος επισημαίνεται με κόκκινο χρώμα.

4. Παρουσίαση και Ανάλυση Αποτελεσμάτων

Στο παρόν κεφάλαιο παρουσιάζονται αναλυτικά τα υπολογιστικά αποτελέσματα που προέκυψαν από την εφαρμογή του POA για την αποδοτική επίλυση του IRP. Για την διεξαγωγή των πειραμάτων αξιολογήθηκε μέρος του συνόλου των αρχείων παραδειγμάτων (instances) που αναφέραμε στο προηγούμενο κεφάλαιο.

Για να ελεγχθεί κλιμακωσιμότητα (scalability) του αλγορίθμου, τα instances κατηγοριοποιήθηκαν βάσει του μεγέθους του δικτύου διανομής (αριθμός πελατών). Συγκεκριμένα επεξεργάστηκαν δίκτυα μικρής ($N = 10$), μεσαίας ($N = 25$) και επιλεγμένα αρχεία μεγάλης ($N \geq 50$) κλίμακας.

Ο κύριος όγκος της έρευνας επικεντρώθηκε συνειδητά σε αρχεία δικτύων μικρής και μεσαίας κλίμακας. Παρά τις εκτεταμένες παραμετροποιήσεις του αλγορίθμου και την εφαρμογή ελεγχόμενης παράλληλης επεξεργασίας που αναλύθηκαν στο προηγούμενο κεφάλαιο, η υπολογιστική πολυπλοκότητα του προβλήματος αποτελεί αξιόπεραστο εμπόδιο για το διαθέσιμο τεχνολογικό εξοπλισμό. Καθώς η έρευνα έγινε στην ουσία σε έναν τυπικό ηλεκτρονικό υπολογιστή, η ταυτόχρονη εκτέλεση χιλιάδων εκτελέσεων αξιολογήσεων βελτιστοποίησης και δρομολόγησης θα απαιτούσε μη ρεαλιστικούς χρόνους εκτέλεσης, υπερβαίνοντας παράλληλα τα φυσικά όρια του συστήματος. Συνεπώς η έρευνα εστιάζει στην μελέτη της λειτουργικότητας και τις σταθερότητας του POA στα πιο διαχειρίσιμα δίκτυα.

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, για την επεξεργασία των αρχείων μικρής κλίμακας ($N \leq 50$) η αρχικοποίηση του POA ορίστηκε με 30 πελεκάνους (agents) και 500 επαναλήψεις. Η στρατηγική του κυλιόμενου ορίζοντα ορίστηκε στις τρεις ημέρες ($t+3$) για τα αρχεία μικρής κλίμακας ($N = 10$), τα μεσαίας κλίμακας στις δύο ημέρες ($t+2$) και στα μεγάλης κλίμακας ($N \geq 50$) στη μία επιπλέον ημέρα ($t+1$).

Η συγκεκριμένη δυναμική προσαρμογή του παραθύρου πρόβλεψης βασίστηκε κυρίως στη διαθέσιμη χωρητική δυνατότητα των οχημάτων αναλογικά με τον αριθμό των πελατών του δικτύου. Όπως χαρακτηριστικά επισημαίνεται στο [26] σε εκτεταμένα δίκτυα η συνολική χωρητικότητα του στόλου αναλώνεται σχεδόν εξ ολοκλήρου για την κάλυψη των άμεσων

αναγκών του μεγάλου πλήθους πελατών. Συνεπώς μια μακροπρόθεσμη πρόβλεψη τριών ημερών ($t+3$) σε ένα δίκτυο τουλάχιστον 50 πελατών είναι πρακτικά ανώφελη, καθώς στα διαθέσιμα οχήματα δεν περισσεύει χώρος για τις προληπτικές παραδόσεις. Αντίθετα, στα μικρά δίκτυα ($N = 10$) η αναλογία χωρητικότητας – ζήτησης είναι ευνοϊκή και επιτρέπει στα οχήματα να μεταφέρουν αποθέματα για πολλαπλές μελλοντικές ημέρες, που δικαιολογημένα αξιοποιείται ο διευρυμένος ορίζοντας των τριών ημερών.

Για κάθε επιμέρους instance πραγματοποιήθηκαν πολλαπλές, ανεξάρτητες εκτελέσεις ώστε να εξαχθούν ασφαλή στατιστικά συμπεράσματα με τα μεγέθη που αναφέρθηκαν στο προηγούμενο κεφάλαιο (Μέσο Κόστος, Τυπική Απόκλιση, Συντελεστής Μεταβλητότητας CV%, και Ποσοστιαίο Χάσμα Gap%).

4.1 Αποτελέσματα για τα δίκτυα Μικρής Κλίμακας ($N=10$)

Στα συγκεκριμένα παραδείγματα το δίκτυο των 10 πελατών το εξυπηρετεί ένας κεντρικός προμηθευτής. Αν και το μέγεθος του δικτύου είναι μικρό, η πολυπλοκότητα της ταυτόχρονης διαχείρισης αποθεμάτων και δρομολόγησης παραμένει σε υψηλό επίπεδο, ειδικά με την χρήση του διευρυμένου ορίζοντα 3 ημερών.

Για την διασφάλιση της στατιστικής αξιοπιστίας, κάθε ένα από τα 36 instances εκτελέστηκε για 10 ανεξάρτητες δοκιμές (runs), με πληθυσμό 30 πελεκάνων και 500 επαναλήψεις. Στον Πίνακα 4-1 έως 4-3 παρουσιάζονται συγκεντρωτικά τα αποτελέσματα αυτών των πειραμάτων.

Για την καλύτερη κατανόηση των πινάκων, στην πρώτη στήλη παρουσιάζονται τα instances και ερμηνεύονται ως εξής: R_C για την Αγροτική και Ομαδοποιημένη κατανομή (Rural Clustered), R_R για την Αγροτική και Τυχαία κατανομή (Rural Random), R_RC για την Αγροτική και Τυχαία Ομαδοποιημένη κατανομή (Rural Random Clustered), U_C για την Αστική και Ομαδοποιημένη κατανομή (Urban Clustered), U_R για την Αστική και Τυχαία κατανομή (Urban Random), U_RC για την Αστική και Τυχαία Ομαδοποιημένη κατανομή (Urban Random Clustered). Η ένδειξη Q αναφέρεται στην χωρητικότητα των οχημάτων, η ένδειξη T στην διάρκεια της χρονικής περιόδου που αναγράφεται αντίστοιχα στην επόμενη στήλη και η ένδειξη K που αναγράφει το πλήθος των διαθέσιμων οχημάτων.

Όνομα Instance	T	K	Best Cost	Mean Cost	Std. Deviation	Mean Time (sec)	Gap %	CV %
R_C_N10_Q8_T6.txt	6	10	12.844,05	13.474,79	609,74	273,70	4,91	4,53
R_C_N10_Q8_T9.txt	9	9	42.707,34	42.796,87	49,87	398,29	0,21	0,12
R_C_N10_Q18_T6.txt	6	4	4.774,55	4.914,03	80,14	266,11	2,92	1,63
R_C_N10_Q18_T9.txt	9	4	12.176,25	12.387,03	160,37	397,53	1,73	1,29
R_C_N10_Q38_T6.txt	6	2	7.172,31	7.461,57	196,36	266,35	4,03	2,63
R_C_N10_Q38_T9.txt	9	2	15.231,47	16.951,75	1.260,59	395,45	11,29	7,44
R_R_N10_Q8_T6.txt	6	10	18.342,22	18.917,22	542,60	266,09	3,13	2,87
R_R_N10_Q8_T9.txt	9	10	37.272,62	38.980,86	764,02	401,47	4,58	1,96
R_R_N10_Q18_T6.txt	6	4	17.025,73	17.522,63	389,77	262,02	2,92	2,22
R_R_N10_Q18_T9.txt	9	4	17.765,23	18.504,00	583,47	400,52	4,16	3,15
R_R_N10_Q38_T6.txt	6	2	24.611,18	39.173,63	5.590,75	264,00	59,17	14,27
R_R_N10_Q38_T9.txt	9	2	16.637,77	17.422,76	965,00	399,80	4,72	5,54
R_RC_N10_Q8_T6.txt	6	10	24.982,00	25.017,10	18,05	265,11	0,14	0,07
R_RC_N10_Q8_T9.txt	9	10	35.844,91	36.413,48	410,57	404,06	1,59	1,13
R_RC_N10_Q18_T6.txt	6	5	10.926,88	11.259,40	324,33	260,99	3,04	2,88
R_RC_N10_Q18_T9.txt	9	5	22.232,18	22.573,18	123,28	397,40	1,53	0,55

Πίνακας 4-1: Αποτελέσματα για Δίκτυα Μικρής Κλίμακας (N=10) - Μέρος 1ο

Όνομα Instance	T	K	Best Cost	Mean Cost	Std. Deviation	Mean Time (sec)	Gap %	CV %
R_RC_N10_Q38_T6.txt	6	2	24.456,13	28.795,50	2.989,10	263,19	17,74	10,38
R_RC_N10_Q38_T9.txt	9	2	10.082,33	10.449,42	211,98	394,95	3,64	2,03
U_C_N10_Q8_T6.txt	6	10	28.237,36	28.251,83	6,61	451,15	0,05	0,02
U_C_N10_Q8_T9.txt	9	10	10.897,15	12.277,14	551,89	700,44	12,66	4,50
U_C_N10_Q18_T6.txt	6	5	15.535,33	15.604,32	138,08	459,84	0,44	0,88
U_C_N10_Q18_T9.txt	9	4	167.125,78	184.020,54	9.525,97	696,61	10,11	5,18
U_C_N10_Q38_T6.txt	6	2	4.852,14	5.294,89	351,12	454,14	9,12	6,63
U_C_N10_Q38_T9.txt	9	2	55.698,04	64.878,39	5.553,61	698,87	16,48	8,56
U_R_N10_Q8_T6.txt	6	10	18.531,84	19.539,40	525,28	461,55	5,44	2,69
U_R_N10_Q8_T9.txt	9	11	32.443,70	33.350,50	507,48	705,07	2,79	1,52
U_R_N10_Q18_T6.txt	6	4	23.735,14	28.510,29	3.188,87	462,21	20,12	11,18
U_R_N10_Q18_T9.txt	9	4	56.672,09	65.645,34	6.878,39	698,76	15,83	10,48
U_R_N10_Q38_T6.txt	6	2	8.958,02	9.326,63	202,94	461,96	4,11	2,18
U_R_N10_Q38_T9.txt	9	2	19.166,43	25.486,19	3.946,63	694,50	32,97	15,49
U_RC_N10_Q8_T6.txt	6	10	14.364,80	14.823,76	260,46	456,98	3,20	1,76
U_RC_N10_Q8_T9.txt	9	10	54.429,92	55.947,26	874,38	711,03	2,79	1,56

Πίνακας 4-2: Αποτελέσματα για Δίκτυα Μικρής Κλίμακας (N=10) - Μέρος 2ο

Όνομα Instance	T	K	Best Cost	Mean Cost	Std. Deviation	Mean Time (sec)	Gap %	CV %
U_RC_N10_Q18_T6.txt	6	5	14.240,72	14.849,79	429,68	465,10	4,28	2,89
U_RC_N10_Q18_T9.txt	9	4	19.652,02	24.165,16	2.570,80	695,83	22,97	10,64
U_RC_N10_Q38_T6.txt	6	2	10.239,95	10.239,95	0,00	420,29	0,00	0,00
U_RC_N10_Q38_T9.txt	9	2	16.754,82	23.945,69	4.350,74	583,83	42,92	18,17

Πίνακας 4 - 1: Αποτελέσματα για Δίκτυα Μικρής Κλίμακας (N=10) - Μέρος 3ο

Από τα παραπάνω αποτελέσματα παρατηρούμε ότι ο μέσος χρόνος επεξεργασίας των αρχείων κυμάνθηκε από ≈ 260 δευτερόλεπτα για τα απλούστερα προβλήματα έως ≈ 711 δευτερόλεπτα για τα πιο πολύπλοκα, αποδεικνύοντας ότι για τα δίκτυα με 10 πελάτες, η επίλυση ολοκληρώνεται σε λιγότερο από 12 λεπτά ανά εκτέλεση.

Ο Συντελεστής Μεταβλητότητας (CV%) διατηρήθηκε σε εξαιρετικά χαμηλά επίπεδα (κάτω από 3 με 4 %), υποδεικνύοντας την ομοιογένεια των λύσεων μεταξύ των πολλαπλών ανεξάρτητων εκτελέσεων. Ενδεικτικά, σε instances όπως το U_C_N10_Q8_T6 και το R_RC_N10_Q8_T6, ο δείκτης CV% άγγιξε το 0,02% και 0,07% αντίστοιχα, ενώ στο U_RC_N10_Q38_T6 καταγράφηκε απόλυτη σταθερότητα (0,00%). Με την αμελητέα αυτή διακύμανση συμπεραίνουμε πως ο αλγόριθμος POA δεν λειτουργεί με τρόπο τυχαίο, αλλά συγκλίνει αξιόπιστα και συστηματικά προς την ίδια βέλτιστη περιοχή του χώρου αναζήτησης σε κάθε εκτέλεση.

Αντίστοιχα συμπεράσματα εξάγονται και από το Ποσοστιαίο Χάσμα (Gap %). Για τα instances με τυπική χωρητικότητα οχημάτων ($Q = 8$ και $Q = 18$), το χάσμα μεταξύ της καλύτερης λύσης (Best Cost) και του μέσου όρου (Mean Cost) περιορίζεται τις περισσότερες φορές κάτω από το 5%. Αυτό πιστοποιεί την ικανότητα του POA να ανακαλύπτει εξαιρετικά ποιοτικές λύσεις με συνέπεια.

Ένα ενδιαφέρον εύρημα είναι η συμπεριφορά του αλγορίθμου στα instances με την χωρητικότητα των οχημάτων $Q = 38$. Στα αρχεία R_R_N10_Q38_T6 και

U_RC_N10_Q38_T9 για παράδειγμα, η διαθέσιμη χωρητικότητα του στόλου ($K = 2$ Οχήματα) είναι περιορισμένη που καθιστά το πρόβλημα ασφυκτικά αυστηρό. Όπως ήταν αναμενόμενο, υπό τόσα στενά περιθώρια οι πιθανότητες εμφάνισης ελλείψεων (stockouts) αυξήθηκαν, ενεργοποιώντας βαρύτερες δυναμικές ποινές. Η ενεργοποίηση των ποινών σε ορισμένες μόνο εκτελέσεις εξηγεί απόλυτα την παρατηρούμενη αύξηση τόσο στο *Gap %* (έως και 59,17%) όσο και στο *CV %* (έως 18,17%). Αυτό το αποτέλεσμα επιβεβαιώνει στην πράξη την λογική της *fitness function*: σε συνθήκες έλλειψης διαθέσιμων πόρων, ο αλγόριθμος πασχίζει να βρει εφικτές λύσεις εντός του αυστηρού ορίου, επιτρέποντας αναγκαστικές ποινές όταν είναι η μοναδική διέξοδος για την εξυπηρέτηση του δικτύου.

Συγκρίνοντας την μετάβαση από την μία χρονική περίοδο στην άλλη ($T = 6$ και $T = 9$) είναι αναμενόμενο να υπάρχει μια φυσιολογική αύξηση στο συνολικό κόστος. Είναι όμως αξιοσημείωτο ότι παρόλο που αυξάνεται ο χρόνος εκτέλεσης ο αλγόριθμος δεν αποσταθεροποιείται. Χαρακτηριστική είναι η περίπτωση των αρχείων R_C_N10_Q8_T6 και R_C_N10_Q8_T9 όπου με την συγκεκριμένη ικανοποιητική χωρητικότητα ($Q = 8$), ο δείκτης *CV%* παραμένει περισσότερο σταθερός (μεταβολή από 4,53 % σε 0,12%).

Τέλος, το σύνολο των δεδομένων περιλαμβάνει διαφορετικές γεωγραφικές κατανομές και είναι αξιοσημείωτο ότι ο αλγόριθμος συμπεριφέρεται πολύ ικανοποιητικά σε όλες τις κατανομές. Αν και οι πελάτες με την Ομαδοποιημένη κατανομή διευκολύνουν περισσότερο τον αλγόριθμο να εντοπίσει φθηνότερα δρομολόγια, η γενική ικανότητα του POA να ορίζει σωστά την σειρά εξυπηρέτησης φαίνεται να είναι ανεπηρέαστη από την γεωγραφική κατανομή.

4.2 Αποτελέσματα για τα δίκτυα Μεσαίας Κλίμακας ($N=25$)

Είναι προφανές ότι η κλιμάκωση του προβλήματος IRP σε 25 πελάτες επιφέρει μια εκθετική αύξηση στον χώρο αναζήτησης. Για την αντιμετώπιση αυτής της πολυπλοκότητας, αλλά τον περιορισμό του υπολογιστικού χρόνου και της λογικής που αναπτύξαμε προηγουμένως, ο ορίζοντας κυλιόμενου παραθύρου ορίστηκε στις 2 ημέρες ($t+2$). Οι παρακάτω πίνακες συνοψίζουν τα αποτελέσματα για τα 30 αρχεία που επεξεργαστήκαμε.

Όνομα Instance	T	K	Best Cost	Mean Cost	Std. Dev	Time (s)	Gap %	CV %
R_C_N25_Q18_T6.txt	6	10	17.771,56	19.361,05	817,90	1.472,86	8,94	4,22
R_C_N25_Q18_T9.txt	9	10	52.771,59	55.321,04	2.344,77	2.250,09	4,83	4,24
R_C_N25_Q38_T6.txt	6	5	22.122,74	22.248,33	79,59	1.511,02	0,57	0,36
R_C_N25_Q38_T9.txt	9	5	20.898,93	20.965,19	33,72	2.219,05	0,32	0,16
R_C_N25_Q8_T6.txt	6	25	74.418,15	74.576,25	119,16	1.492,29	0,21	0,16
R_C_N25_Q8_T9.txt	9	23	74.810,27	75.041,07	123,56	2.242,26	0,31	0,16
R_R_N25_Q18_T6.txt	6	11	39.025,45	39.611,60	325,35	1.469,20	1,50	0,82
R_R_N25_Q18_T9.txt	9	12	63.467,93	64.531,18	615,08	2.195,17	1,68	0,95
R_R_N25_Q38_T6.txt	6	5	18.995,56	19.312,26	199,89	1.440,83	1,67	1,04
R_R_N25_Q38_T9.txt	9	6	49.264,31	50.054,01	651,66	2.185,48	1,60	1,30
R_RC_N25_Q18_T6.txt	6	15	31.742,70	32.169,90	227,48	1.453,93	1,35	0,71
R_RC_N25_Q18_T9.txt	9	14	76.373,56	77.585,66	713,48	2.207,26	1,59	0,92
R_RC_N25_Q38_T6.txt	6	5	17.655,80	24.524,93	3.035,00	2.367,01	38,91	12,38
R_RC_N25_Q38_T9.txt	9	5	287.101,99	326.765,22	22.872,64	3.779,46	13,82	7,00
R_RC_N25_Q8_T6.txt	6	24	39.942,98	44.480,82	3.077,23	2.637,48	11,36	6,92
R_RC_N25_Q8_T6.txt	6	24	71.054,82	72.132,22	651,79	1.457,70	1,52	0,90
R_RC_N25_Q8_T9.txt	9	22	90.096,24	91.166,44	615,51	2.214,76	1,19	0,68
U_C_N25_Q18_T6.txt	6	9	35.258,73	35.667,47	452,41	1.489,40	1,16	1,27
U_C_N25_Q18_T9.txt	9	10	49.150,90	49.442,60	114,47	2.249,33	0,59	0,23

Πίνακας 4 - 2: Αποτελέσματα για Δίκτυα Μεσαίας Κλίμακας (N=25) - Μέρος 1ο

Όνομα Instance	T	K	Best Cost	Mean Cost	Std. Dev	Time (s)	Gap %	CV %
U_C_N25_Q38_T6.txt	6	5	24.530,43	24.784,23	156,92	1.488,38	1,03	0,63
U_C_N25_Q38_T9.txt	9	5	30.051,77	31.453,48	923,74	2.234,91	4,66	2,94
U_C_N25_Q8_T6.txt	6	24	42.376,90	43.321,30	508,72	1.508,59	2,23	1,17
U_C_N25_Q8_T6.txt	6	24	43.050,42	44.414,98	1.035,17	2.809,27	3,17	2,33
U_C_N25_Q8_T9.txt	9	25	116.848,71	117.875,81	1.065,44	2.274,23	0,88	0,90
U_R_N25_Q8_T6.txt	9	25	119.287,28	119.753,58	324,04	4.350,53	0,39	0,27
U_R_N25_Q18_T6.txt	6	13	43.467,54	44.576,04	583,56	1.816,55	2,55	1,31
U_R_N25_Q18_T9.txt	9	10	140.699,23	154.947,21	10.604,30	2.645,71	10,13	6,84
U_R_N25_Q38_T6.txt	6	5	23.339,79	23.897,39	361,19	1.735,18	2,39	1,51
U_R_N25_Q8_T6.txt	6	23	56.956,74	58.275,44	1.024,34	1.794,69	2,32	1,76
U_R_N25_Q8_T9.txt	9	25	103.176,93	105.181,93	1.669,23	2.688,08	1,94	1,59

Πίνακας 4 - 3:Αποτελέσματα για Δίκτυα Μεσαίας Κλίμακας (N=25) - Μέρος 2ο

Η επεξεργασία των instances με τους 25 πελάτες αναδεικνύει κρίσιμα και αξιοσημείωτα συμπεράσματα, τα οποία εδραιώνουν την δυναμική του POA.

Η χρονική διάρκεια των εκτελέσεων ήταν ιδιαιτέρως αυξημένη καθώς ο μέσος όρος επεξεργασίας κυμάνθηκε μεταξύ 1500 και 2800 δευτερολέπτων (25 έως 46 λεπτά), ενώ σε αρχεία με εξαιρετικά δύσκολες συνθήκες διαθέσιμων πόρων (πχ. U_R_N25_Q8_T6) απαιτήθηκε χρόνος 4.350 δευτερόλεπτα περίπου (πάνω από 70 λεπτά). Η παρατήρηση αυτή συνδέεται άμεσα με τον ορισμό του ορίζοντα κυλιόμενου παραθύρου σε $t + 2$, αλλά και της ελεγχόμενης παραμετροποίησης του αλγορίθμου.

Ωστόσο, παρόλο τον αντίκτυπο της αύξησης του υπολογιστικού χρόνου, βάσει των δεικτών CV%, ο αλγόριθμος POA είχε εξαιρετική ομοιογένεια στα αποτελέσματά του. Στην συντριπτική πλειονότητα των instances, ειδικά σε αυτά που δεν υπήρχε περιορισμός ως

προς την χωρητικότητα ($Q = 8$ και $Q = 18$), ο $CV\%$ κατέγραψε τιμές μικρότερες του 1% (π.χ. 0,16% στα $R_C_N25_Q8_T6$ και $R_C_N25_Q8_T9$ και 0,23% στο $U_C_N25_Q18_T9$). Η ικανότητα του αλγορίθμου να παράγει σχεδόν πανομοιότυπες, βέλτιστες λύσεις σε κάθε ανεξάρτητη δοκιμή, αποτελεί το ισχυρότερο επιχείρημα υπέρ της αποτελεσματικότητάς του.

Καταλήγοντας, στα instances με την περιορισμένη συνολική χωρητικότητα ($Q = 38$) και μικρό αριθμό στόλου, επιβεβαιώνεται το γεγονός εμφάνισης επαναλαμβανόμενων ελλείψεων. Η επιβολή δυναμικών ποινών αποτυπώνεται ξεκάθαρα σε περιπτώσεις όπως το $R_RC_N25_Q38_T6$, όπου το $Gap\%$ εκτοξεύτηκε στο 38,91% και το $CV\%$ στο 12,38%. Η συμπεριφορά αυτή είναι απόλυτα φυσιολογική, καθώς το σύστημα αδυνατεί να εξυπηρετήσει βέλτιστα 25 πελάτες με μόλις 5 ($K = 5$) οχήματα χωρητικότητας ($Q = 38$), εγκλωβίζοντας τον αλγόριθμο στην προσπάθεια ελαχιστοποίησης ή αποφυγής των αναπόφευκτων ποινών.

4.3 Αποτελέσματα για τα δίκτυα Μεγάλης Κλίμακας ($N \geq 50$)

Η αξιολόγηση των αρχείων που περιλαμβάνουν τα δίκτυα με ή και περισσότερους από 50 πελάτες είναι το απόγειο του POA. Οι ίδιοι οι δημιουργοί των instances, αναφέρουν χαρακτηριστικά στο [26] ότι η πλειονότητα αυτών των αρχείων παραμένουν πρακτικά «άλυτα», ως προς την εύρεση και μαθηματική απόδειξη της απόλυτα βέλτιστης λύσης.

Οι παραδοσιακές ακριβείς μέθοδοι (exact solvers) αδυνατούν να συγκλίνουν σε εφικτές λύσεις μέσα σε αποδεκτούς υπολογιστικούς χρόνους, λόγω της χαοτικής αύξησης των πιθανών συνδυασμών κάλυψης της ζήτησης, διανομής αποθεμάτων και δρομολόγησης οχημάτων. Συνεπώς, η εφαρμογή του αλγορίθμου POA σε αυτά τα αρχεία δεν στόχευε απλώς στην επαλήθευση της λειτουργικότητάς του, αλλά αποκτά καθαρά ερευνητικό χαρακτήρα. Βασική επιδίωξη ήταν να ανακαλύψει λύσεις σε περιπτώσεις όπου η επιστημονική κοινότητα δεν έχει ακόμη καταλήξει στην τελική, βέλτιστη απάντηση.

Δεδομένου του τεράστιου απαιτούμενου υπολογιστικού χρόνου, επιλέχθηκε ένα αντιπροσωπευτικό δείγμα αρχείων $N=50$ να εκτελεστεί 3 έως 5 φορές με τον ορίζοντα κυλιόμενου παραθύρου ορισμένο σε μία ημέρα ($t + 1$). Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα της δοκιμής αυτής:

Όνομα Instance	T	K	Best Cost	Mean Cost	Std. Dev	Time (s)	Gap %	CV %
R_C_N50_Q8_T6.txt	6	47	73.780,46	75.612,86	1.080,04	5.732,31	2,48	1,43
R_C_N50_Q8_T9.txt	9	50	181.729,22	181.817,22	50,31	8.749,58	0,05	0,03
R_RC_N50_Q38_T6.txt	6	9	363.395,17	363.745,17	358,11	5.841,64	0,10	0,10
U_C_N50_Q18_T6.txt	6	20	255.855,59	256.105,02	180,13	5.883,91	0,10	0,07
U_R_N50_Q8_T9.txt	9	46	197.335,16	199.059,49	1.252,24	8.916,14	0,87	0,63
U_R_N50_Q18_T9.txt	9	22	553.322,14	553.726,34	392,07	8.905,87	0,07	0,07

Πίνακας 4 - 4:Αποτελέσματα για Δίκτυα Μεγάλης Κλίμακας (N=50)

Τα παραπάνω αποτελέσματα αποτυπώνουν με τον πλέον ανάγλυφο τρόπο την ακραία πολυπλοκότητα των προβλημάτων N=50, αλλά και την ικανότητα του αλγορίθμου POA να ανταπεξέλθει σε αυτά. Για τον μικρό ορίζοντα των 6 ημερών ($T = 6$) απαιτήθηκε χρόνος εκτέλεσης τουλάχιστον 5.800 δευτερολέπτων (1,5 ώρα περίπου) ενώ στον ορίζοντα των 9 ημερών 8.900 δευτερόλεπτα (2,5 ώρες). Οι χρόνοι αυτοί ωστόσο είναι αναγκαίοι προκειμένου να αποδώσει ο αλγόριθμος την βέλτιστη λύση σε ένα τεράστιο χώρο πιθανών λύσεων.

Παρά το μέγεθος του προβλήματος, ωστόσο, ο δείκτης *Gap%* κατέγραψε τιμές κάτω του 1% στην πλειοψηφία των δειγμάτων και ο *CV%* κυμάνθηκε μεταξύ των τιμών 0,03% και 1,43% (στα αρχεία R_C_N50_Q8_T9 και R_C_N50_Q8_T6 αντίστοιχα). Τα μεγέθη αυτά δείχνουν ότι οι μηχανισμοί εξερεύνησης και εκμετάλλευσης που διαθέτει ο POA του επιτρέπουν να συγκλίνει με συνέπεια προς την βέλτιστη λύση.

Μια ακόμα παρατήρηση είναι στο αντιπροσωπευτικό δείγμα αυτό, παρόλο τον ισχυρισμό ότι η πλειονότητα των instances αυτών των δειγμάτων είναι «άλυτα», τα αρχεία R_C_N50_Q8_T9.txt και U_R_N50_Q8_T9.txt επιλύθηκαν χωρίς ελλείψεις, ενώ τα υπόλοιπα παρουσίασαν ελλείψεις, κυρίως λόγω της δυσαναλογίας διαθέσιμου συνολικού χώρου οχημάτων, του διαθέσιμου στόλου και την κάλυψη της υφιστάμενης ζήτησης των πελατών.

Σε αυτό το σημείο θα πρέπει να διευκρινιστεί ο λόγος για τον οποίο η παρούσα έρευνα δεν επεκτάθηκε στα διαθέσιμα instances ακραίας κλίμακας των 100 και 200 πελατών ($N=100$ και $N=200$). Το IRP αποτελεί ένα NP-hard πρόβλημα. Καθώς το πλήθος των πελατών αυξάνεται, οι πιθανοί συνδυασμοί δρομολόγησης και διαχείρισης αποθεμάτων αυξάνονται δραματικά. Δεδομένου ότι η πειραματική διαδικασία διεξάγεται αποκλειστικά σε ένα τυπικό οικιακό υπολογιστικό σύστημα, η εκτέλεση του αλγορίθμου για 100 ή 200 πελάτες θα απαιτούσε εξωπραγματικούς υπολογιστικούς χρόνους, πιθανότατα πολύ μεγάλων χρονικών διαστημάτων, εξαντλώντας τα φυσικά όρια του συστήματος. Επομένως, το όριο των $N=50$ πελατών θεωρήθηκε το «ταβάνι» των δυνατοτήτων του συγκεκριμένου συστήματος.

5. Συγκριτική Αξιολόγηση

5.1 Σύγκριση με τον αλγόριθμο Branch & Cut

Το παρόν κεφάλαιο εστιάζει στη συγκριτική αξιολόγηση του προτεινόμενου POA έναντι των καθιερωμένων αποτελεσμάτων της βιβλιογραφίας [26]. Ως απόλυτο μέτρο σύγκρισης χρησιμοποιούνται τα πειραματικά δεδομένα από την πρωτότυπη έρευνα με την εφαρμογή της ακριβής μεθόδου Branch & Cut (B&C) ως προσέγγισή τους για την επίλυση των ίδιων instances.

Στους πίνακες που ακολουθούν, αντιπαραβάλλονται δύο θεμελιώδη μεγέθη: το συνολικό κόστος δρομολόγησης και αποθέματος, και ο απαιτούμενος χρόνος υπολογισμού. Για την αξιολόγηση της μεθόδου B&C, χρησιμοποιείται το Άνω Όριο (Upper Bound), το οποίο αντιπροσωπεύει την καλύτερη εφικτή λύση που ανακάλυψε ο αλγόριθμος πριν τον τερματισμό του. Αξίζει να σημειωθεί ότι, στις περισσότερες περιπτώσεις, η μέθοδος B&C δεν κατόρθωσε να επιβεβαιώσει το απόλυτο βέλτιστο κόστος, τερματίζοντας αναγκαστικά λόγω εξάντλησης του προκαθορισμένου χρονικού ορίου των 2 ωρών (7.200 δευτερόλεπτα).

Ο δείκτης σύγκρισης είναι ο *Gap%*, όπου συγκρίνονται οι βέλτιστες λύσεις των αλγορίθμων και για την ορθή αποτύπωση των επιδόσεων, επιλέχθηκε ένα ευρύ και αντιπροσωπευτικό δείγμα αρχείων που καλύπτει όλες τις γεωγραφικές κατανομές. Τα αποτελέσματα παρουσιάζονται στους παρακάτω πίνακες:

Όνομα Instance	B&C (UB)	POA (Best Cost)	Gap %	B&C Time (s)	POA Time (s)
R_C_N10_Q8_T6	5.970,59	12.844,05	+ 115,12 %	16	273,70
R_C_N10_Q18_T6	2.008,91	4.774,55	+ 137,67 %	7.200 (Όριο)	266,11
R_C_N10_Q38_T9	6.579,44	15.231,47	+ 131,50 %	7.200 (Όριο)	395,45
R_R_N10_Q8_T6	7.219,63	18.342,22	+ 154,06 %	404	266,09
R_R_N10_Q18_T9	5.950,01	17.765,23	+ 198,57 %	7.200 (Όριο)	400,52

Πίνακας 5-1: Συγκριτική Αξιολόγηση Επιδόσεων για Αντιπροσωπευτικά Δίκτυα (N=10) - Μέρος 1ο

Όνομα Instance	B&C (UB)	POA (Best Cost)	Gap %	B&C Time (s)	POA Time (s)
R_RC_N10_Q8_T6	10.763,10	24.982,00	+ 132,11 %	96	265,11
R_RC_N10_Q38_T6	3.516,38	24.456,13	+ 595,49 %	39	263,19
U_C_N10_Q8_T9	760,09	10.897,15	+ 1333,66 %	7.200 (Όριο)	700,44
U_R_N10_Q18_T6	1.321,38	23.735,14	+ 1696,25 %	22	462,21

Πίνακας 5-2: Συγκριτική Αξιολόγηση Επιδόσεων για Αντιπροσωπευτικά Δίκτυα (N=10) – Μέρος 2ο

Όνομα Instance	B&C (UB)	POA (Best Cost)	Gap %	B&C Time (s)	POA Time (s)
R_C_N25_Q8_T6	36.031,90	74.418,15	+ 106,53 %	7.200 (Όριο)	1.492,29
R_C_N25_Q18_T6	7.544,34	17.771,56	+ 135,56 %	7.200 (Όριο)	1.472,86
R_C_N25_Q38_T6	9.939,82	22.122,74	+ 122,56 %	7.198 (Όριο)	1.511,02
R_R_N25_Q18_T6	14.097,90	39.025,45	+ 176,81 %	7.199 (Όριο)	1.469,20
R_R_N25_Q38_T9	16.179,90	49.264,31	+ 204,47 %	7.198 (Όριο)	2.185,48
R_RC_N25_Q8_T9	38.956,40	90.096,24	+ 131,27 %	7.199 (Όριο)	2.214,76
R_RC_N25_Q38_T6	5.665,70	17.655,80	+ 211,62 %	7.199 (Όριο)	2.367,01
U_C_N25_Q18_T9	4.130,00	49.150,90	+ 1089,85 %	7.199 (Όριο)	2.249,33
U_R_N25_Q38_T6	1.761,59	23.339,79	+ 1224,92 %	7.200 (Όριο)	1.735,18

Πίνακας 5-3: Συγκριτική Αξιολόγηση Επιδόσεων για Αντιπροσωπευτικά Δίκτυα (N=25)

Η ανάλυση των συγκριτικών δεδομένων αναδεικνύει μια σαφή και απόλυτα συνεπή συμπεριφορά: η μέθοδος B&C επιτυγχάνει σταθερά λύσεις χαμηλότερου κόστους ωστόσο η επίτευξη αυτών των ορίων συνοδεύεται από ένα πάρα πολύ μεγάλο υπολογιστικό κόστος. Όπως παρατηρείται στον Πίνακα 5.1, ο αλγόριθμος B&C καταφέρνει να βρει τη βέλτιστη λύση σε λιγότερο από ένα λεπτό μόνο για τα πλέον απλοϊκά σενάρια (π.χ. R_C_N10_Q8_T6, με χρόνο 16 δευτερόλεπτα). Ωστόσο, με την παραμικρή αύξηση της

πολυπλοκότητας (όπως η μείωση της χωρητικότητας των οχημάτων στο $Q = 18$ ή η αύξηση του χρονικού ορίζοντα στο $T = 9$), η μαθηματική μέθοδος καταρρέει. Σε προβλήματα μόλις 10 πελατών, ο B&C εξαντλεί συστηματικά το όριο των 2 ωρών (7.200 δευτερόλεπτα) χωρίς να μπορεί να καταλήξει σε μια βέλτιστη λύση. Το φαινόμενο αυτό κορυφώνεται στα δίκτυα των 25 πελατών, όπου σχεδόν το σύνολο των εκτελέσεων του B&C τερματίζεται βίαια λόγω χρονικού ορίου.

5.2 Σύγκριση με τον αλγόριθμο Matheuristic

Πέραν της ακριβούς μεθόδου B&C, στην αρχική έρευνα [26] αναπτύχθηκε και μια ευρετική μέθοδος Matheuristic για να αντιμετωπίσει τους χρονικούς περιορισμούς. Προκειμένου να παρουσιαστεί συνοπτικά η διαφορά επιδόσεων, στον παρακάτω πίνακα παρατίθεται ένα ενδεικτικό δείγμα 4 αρχείων:

Όνομα Instance	Matheuristic (UB)	POA Best Cost	Gap vs Math %	Math Time (s)	POA Time (s)
R_C_N10_Q8_T6	5.974,75	12.844,05	+ 114,97 %	1.800	273,70
R_R_N10_Q18_T9	6.041,41	17.765,23	+ 194,06 %	1.800	400,52
R_C_N25_Q8_T6	36.479,54	74.418,15	+ 104,00 %	1.800	1.492,29
R_RC_N25_Q38_T6	5.909,60	17.655,80	+ 198,76 %	1.800	2.367,01

Πίνακας 5-4: Ενδεικτική Σύγκριση Επιδόσεων έναντι της Matheuristic

Από το παραπάνω δείγμα διαπιστώνεται ότι, μολονότι η Matheuristic βρίσκει λύσεις κοντά στα όρια του B&C, χαρακτηρίζεται από ένα αυστηρό υπολογιστικό πλαίσιο, καθώς η διαδικασία απαιτεί σταθερά 1.800 δευτερόλεπτα (ακριβώς 30 λεπτά) ανά εκτέλεση, ανεξαρτήτως μεγέθους του προβλήματος (είτε πρόκειται για $N=10$ είτε για $N=25$).

Αντιθέτως, ο αλγόριθμος POA επιδεικνύει διαφορετική δυναμική προσαρμογή. Στα προβλήματα μικρής κλίμακας είναι έως και 85% ταχύτερος (ολοκληρώνοντας σε 4-6 λεπτά), ενώ διατηρεί εξαιρετικά ανταγωνιστικούς χρόνους στη μεσαία κλίμακα, αποδεικνύοντας τη στιβαρότητά του σε πρακτικές εφαρμογές σε σχέση με τη σταθερή χρονική αναμονή της μεθόδου των συγγραφέων.

6. Συμπεράσματα και Προτάσεις

6.1 Συμπεράσματα

Η παρούσα ερευνητική εργασία ολοκληρώθηκε με την ανάπτυξη, υλοποίηση και συστηματική αξιολόγηση του Pelican Optimization Algorithm (POA) στο Πρόβλημα Δρομολόγησης Αποθέματος με Πολλαπλά Οχήματα (Multi-Vehicle IRP). Μέσα από τα υπολογιστικά πειράματα και τη συγκριτική ανάλυση με τα αποτελέσματα αναφοράς της βιβλιογραφίας, αναδείχθηκαν τα δομικά χαρακτηριστικά, οι δυνατότητες αλλά και οι εγγενείς περιορισμοί της προτεινόμενης μεθοδολογίας.

Η παρατηρούμενη απόκλιση του POA (υψηλό ποσοστό CV% στο συνολικό κόστος) οφείλεται ενδεχομένως στις συνειδητές απλοποιήσεις που ενσωματώθηκαν στη συνάρτηση καταλληλότητας: η χρήση του 2-opt, η απληστική λογική φόρτωσης και οι δυναμικές ποινές. Οι συγκεκριμένοι ευρετικοί μηχανισμοί ενσωματώθηκαν στρατηγικά στον αλγόριθμο προκειμένου να καταστήσουν την εξερεύνηση του αχανούς χώρου λύσεων υπολογιστικά εφικτή και να διασφαλίσουν τη σταθερή σύγκλιση, αποφεύγοντας τον εγκλωβισμό σε τοπικά ελάχιστα, ειδικά κατά τη μετάβαση σε δίκτυα μεγαλύτερης κλίμακας ($N=25$ και $N=50$).

Ως εκ τούτου, το βασικό συμπέρασμα της έρευνας επικεντρώνεται σε μια σαφή και καθορισμένη διατύπωση: Ο προτεινόμενος αλγόριθμος POA αποδεικνύεται εξαιρετικά γρήγορος ως προς τους χρόνους εκτέλεσης και εξασφαλίζει την ιδιότητα του στη διαδικασία βελτιστοποίησης, όμως οι παραγόμενες λύσεις έχουν αυξημένο αντικειμενικό κόστος σε σύγκριση με τα άνω όρια των ακριβών μεθόδων.

6.2 Προτάσεις για μελλοντική έρευνα και εφαρμογές

Όπως αποδείχθηκε κατά την πειραματική αξιολόγηση, η κύρια πρόκληση του IRP εντοπίζεται στην παραγοντική αύξηση της πολυπλοκότητάς του, η οποία περιόρισε την εκτέλεση του αλγορίθμου POA στο «ταβάνι» των 50 πελατών, λόγω των φυσικών αντοχών του διαθέσιμου τυπικού υπολογιστικού συστήματος. Ως εκ τούτου, η πλέον επιβεβλημένη μελλοντική τεχνολογική κατεύθυνση της παρούσας έρευνας είναι η μεταφορά και εκτέλεση του κώδικα σε καλύτερες τεχνολογικά υποδομές ή σε κατανεμημένα περιβάλλοντα

υπολογιστικού νέφους (Cloud Computing). Μέσω της αξιοποίησης συστοιχιών υπερ-υπολογιστών ή της μαζικής επιτάχυνσης μέσω καρτών γραφικών, ο αλγόριθμος για παράδειγμα θα μπορούσε να αξιολογεί εκατομμύρια διαδρομές 2-opt ταυτόχρονα. Μια τέτοια αναβάθμιση θα εξάλειφε το πρακτικό υπολογιστικό εμπόδιο, επιτρέποντας στον ΡΟΑ να επιλύσει τα «άλυτα» προβλήματα των 100 και 200 πελατών σε ρεαλιστικούς χρόνους, καταγράφοντας ενδεχομένως βελτιωμένα αποτελέσματα.

Μια ακόμα σημαντική επισήμανση είναι ότι η υλοποίηση στην γλώσσα προγραμματισμού Python να μην είναι αποδοτική και να χρειάζεται να πραγματοποιηθεί σε άλλες γλώσσες χαμηλότερου επιπέδου.

Πέραν της έρευνας, ο αλγόριθμος με κάποιες κατάλληλες τροποποιήσεις ή με διαφορετικές προσεγγίσεις, παρουσιάσει εξαιρετικά άμεσες επιχειρηματικές προεκτάσεις για τη σύγχρονη βιομηχανία. Στο πλαίσιο μιας μελλοντικής πρακτικής εφαρμογής, ο κώδικας θα μπορούσε να μετατραπεί σε ενεργό υποσύστημα βελτιστοποίησης, εφόσον βρεθεί ο περεταίρω τρόπος βελτίωσής του, το οποίο θα διασυνδέεται δυναμικά με τα υπάρχοντα Συστήματα Διαχείρισης Επιχειρησιακών Πόρων (ERP) και Συστήματα Διαχείρισης Αποθηκών (WMS) μιας εταιρείας. Σε ένα τέτοιο σενάριο εφαρμογής, αντί να τροφοδοτείται με στατικά αρχεία αναφοράς, ο αλγόριθμος θα λαμβάνει σε πραγματικό χρόνο τα ζωντανά δεδομένα των παραγγελιών, την τηλεματική διαθεσιμότητα του στόλου και τις χωρητικότητες των εκάστοτε πελατών. Αυτό θα επέτρεπε στα τμήματα εφοδιασμού μεταφορών να εγκαταλείψουν τον εμπειρικό προγραμματισμό και να εξάγουν αυτοματοποιημένα, δυναμικά πλάνα δρομολόγησης, μειώνοντας δραστικά το μεταφορικό κόστος και βελτιστοποιώντας την κυκλοφορία των αποθεμάτων τους.

Επίσης μία άλλη πρόταση είναι η υβριδοποίηση του αλγόριθμου, δηλαδή να ενωθεί ο ΡΟΑ με κάποιον άλλο αλγόριθμο (π.χ. γενετικό αλγόριθμο) για να γίνει ακόμα πιο γρήγορος και πιο αποδοτικός και να παρουσιάσει καλύτερες λύσεις με χαμηλότερο κόστος. Όπως αναφέρθηκε και προηγουμένως, ίσως μια διαφορετική προσέγγιση του ΡΟΑ (διαφορετικής από την υφιστάμενη με τους εικονικούς κόμβους) στο πρόβλημα IRP να είχε διαφορετικά αποτελέσματα και καλύτερες επιδόσεις συγκριτικά με τους υφιστάμενους αλγορίθμους αποδοτικής επίλυσης των προβλημάτων δρομολόγησης.

Βιβλιογραφία

1. Archetti C, Bertazzi L, Hertz A, Speranza MG (2012) A Hybrid Heuristic for an Inventory Routing Problem. *INFORMS Journal on Computing*, 2012, 24(1).
2. Bell WJ, Dalberto LM, Fisher ML, Greenfield AJ, Jaikumar R, Kedia P, Mack RG, Prutzman PJ (1983) Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces* 13:4–23.
3. Bertazzi L, Coehlo LC, Maio AD, Lagana D (2019) A matheuristic algorithm for the multi-depot inventory routing problem. *Transportation Research Part E* 122:524–544.
4. Bertazzi L, Speranza MG (2001) Rounding procedures for the discrete version of the capacitated economic order quantity problem. *Ann Oper Res* 107(1):33–49.
5. Bertazzi L, Speranza MG (2002) Continuous and discrete shipping strategies for the single link problem. *Transp Sci* 36(3):314–325.
6. Bertazzi L, Speranza MG (2012) Inventory routing problems: an introduction. *EURO J Transp Logist* 1:307–326.
7. Bertazzi L, Speranza MG (2013) Inventory routing problems with multiple customers. *EURO J Transp Logist* 2:255–275.
8. Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. *Information Sciences* 237:82–117.
9. Campbell A, Clarke L, Kleywert A, Savelsbergh M (1997) *The Inventory Routing Problem*, The Logistics Institute, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205.
10. Coelho LC, Cordeau JF, Laporte G (2014) Heuristics for dynamic and stochastic inventory-routing. *Computers and Operations Research* 52 (2014):55–67.
11. Coelho LC, Laporte G (2012) The exact solution of several classes of inventory-routing problems. *CIRELLT*, 2013, 13.
12. Coelho LC, Laporte G (2013) Improved Solutions for Inventory-Routing Problem through valid Inequalities and Input Ordering. *Computers & Operations Research* 40:558–565.

13. Croes GA, (1958) A method for solving traveling-salesman problems. Operations research 6(6): 791-812.
14. Dantzig G, Fulkerson R, Johnson S (1954), Solution of a large-scale traveling-salesman problem. Journal of the operations research society of America 2(4):393-410.
15. Eiben AE, Smit SK (2011) Parameter tuning for configuring methods for optimization. European Journal of Operational Research 215(1):68-75.
16. Eijkhout V, Chow E, & Gein van de, R (2020) Introduction to High Performance Scientific Computing (3rd edition) [eBook version]. Ανακτήθηκε την 15 Μαΐου, 2026, από <https://www.cs.wustl.edu/~roger/566S.s21/EijkhoutIntroToHPC.pdf>.
17. Gorelick M, & Ozsvald I (2020) High Performance Python: Practical Performant Programming for Humans (2nd edition). Sebastopol:O'Reilly Media.
18. Hanzar P, Kaleta J (2016) Inventory routing problem in rolling horizon planning environment. Transformations in Business and Economics 15(2):373-388.
19. Joines JA, Houck CR (1994) On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GA's. 1st IEEE Conference on Evolutionary Computation: 579-584.
20. Juan AA, Panadero J, Onggo BS, Corlu CG, Agustin A (2019) An Inventory-Routing Problem with Stochastic Demand and Stock-out: A Solution and Risk Analysis using Simheuristics. Winter Simulation Conference:1977-1988.
21. Liu L, Lee L S, Seow H-V, Chen CY (2022) Logistics Center Location-Inventory-Routing Problem Optimization: A Systematic Review Using PRISMA Method. Sustainability 14:15853.
22. Maia F., Figueira G, Neves-Moreira F (2026) Stochastic dynamic inventory-routing: A comprehensive review. Computers and Operations Research 189 (2026):107383.
23. Marchant, S. (1990) Handbook of Australian, New Zealand & Antarctic Birds: Australian Pelican to Ducks; Oxford University Press: Melbourne, Australia.
24. Markov I, Bierlaire M, Cordeau JF, Maknoon Y, Varone S. (2020) Waste collection inventory routing with non-stationary stochastic demands. Computers and Operations Research 113 (2020):10479.

25. Michalewicz Z, (1995) Genetic algorithms, numerical optimization, and constraints. Proceedings of the 6th International Conference on Genetic Algorithms:151-158.
26. Skålnes J, Ahmet MB, Hvatum LM, Stålhane H (2023) New benchmark instances for the inventory routing problem. European Journal of Operational Research 313:992-1014.
27. Speranza MG, Ukovich W (1994) Minimizing transportation and inventory costs for several products on a single link. Oper Res 42:879–894.
28. Speranza MG, Ukovich W (1996) An algorithm for optimal shipments with given frequencies. Naval Res Logist 43(5):655–671.
29. Spinelli A et al. (2025) A rolling horizon heuristic approach for a multi-stage stochastic waste collection problem. European Journal of Operational Research 323 (2025):276–296.
30. Trojovský P, Dehghani M (2022) Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications. Sensors, 22, 855.

Παράρτημα

Ενδεικτικά παρατίθενται οι κώδικες που έχουν γραφθεί στην Python για την επίλυση του προβλήματος.

Συλλογή Στοιχείων από τα instances (load_data.py)

Με τον παρακάτω parser λαμβάνουμε όλα τα απαραίτητα στοιχεία που χρειάζονται ως δεδομένα για την συνάρτηση καταλληλότητας.

```
import numpy as np

def load_data(filepath):
    data = {} # Dictionary with the data from instances

    with open(filepath, 'r') as f:
        lines = [line.strip() for line in f.readlines() if
line.strip()]

    # Parameters
    for line in lines: # According to layout
        parts = line.split()
        if parts[0] == 'nVehicles:':
            data['n_vehicles'] = int(parts[1])
        elif parts[0] == 'nCustomers:':
            data['n_customers'] = int(parts[1])
        elif parts[0] == 'nPeriods:':
            data['n_periods'] = int(parts[1])

    # Vehicle Load
    for i, line in enumerate(lines):
        if 'VEHICLE_TYPE' in line: # According to layout
            veh_data = lines[i + 1].split()
            data['vehicle_capacity'] = int(veh_data[2]) # 3rd column
            break

    # Depot
    coordinates = []
    for i, line in enumerate(lines):
        if 'DEPOT' in line and 'X' in line:
            depot_line = lines[i + 1].split()
            coordinates.append((float(depot_line[1]),
float(depot_line[2])))

            data['supplier_holding_cost'] = float(depot_line[3])
            data['supplier_init_inv'] = float(depot_line[4])
            data['supplier_max_inv'] = float(depot_line[6])

            # Production
            production = [float(x) for x in depot_line[7:]] # After
line 7

            data['production_rate'] = np.array(production)
            break
```

```
# Customer Data
n_customer = data['n_customers']
n_periods = data['n_periods']

n_nodes = n_customer + 1

# Initialization
'''
demands = np.zeros((n_customer, n_periods))
holding_costs = np.zeros(n_customer)
init_inv = np.zeros(n_customer)
max_capacity = np.zeros(n_customer)
'''

demands = np.zeros((n_nodes, n_periods))
holding_costs = np.zeros(n_nodes)
init_inv = np.zeros(n_nodes)
max_capacity = np.zeros(n_nodes)

start_reading = False
customer_count = 0

for i, line in enumerate(lines): # Read line by line and update
data for each customer
    if 'CUSTOMER' in line and 'X' in line:
        start_reading = True
        continue
    if start_reading and customer_count < n_customer:
        parts = line.split()

        coordinates.append((float(parts[1]), float(parts[2])))

        # customer_index = customer_count
        customer_index = customer_count + 1 # 1 for 0 to be stored
to array

        holding_costs[customer_index] = float(parts[3])
        init_inv[customer_index] = float(parts[4])
        max_capacity[customer_index] = float(parts[6]) #
MAX_LEVEL_INV

        demands[customer_index, :] = [float(x) for x in parts[7 :
7 + n_periods]]

        customer_count += 1

data['coordinates'] = np.array(coordinates)
data['demands'] = demands
data['holding_costs'] = holding_costs
data['init_inv'] = init_inv
data['max_capacity'] = max_capacity

# Calculate distance using Euclidean Distance: sqrt((x1-x2)^2
+ (y1-y2)^2)

# n_nodes = n_customer + 1
distance_matrix = np.zeros((n_nodes, n_nodes)) # Initialize to
zero
```

```
for i in range(n_nodes):
    for j in range(n_nodes):
        eu_distance = np.sqrt((coordinates[i][0] -
coordinates[j][0]) ** 2 + (coordinates[i][1] - coordinates[j][1]) **
2)
        distance_matrix[i][j] = int(round(eu_distance))

data['distance_matrix'] = distance_matrix

return data
```

Υπολογισμός εγγύτητας εικονικών και πραγματικών κόμβων (shapely_utils.py)

Με το συγκεκριμένο script υπολογίζουμε την εγγύτητα των εικονικών με τους πραγματικούς κόμβους.

```
from typing import List, Tuple

try:
    from shapely.geometry import Point
except ImportError as e:
    raise ImportError(
        "The 'shapely' package was not found. Install it using: pip
install shapely"
    ) from e

def determine_nearest_nodes(
    point_xy: Tuple[float, float],
    nodes_xy: List[Tuple[float, float]],
) -> List[Tuple[float, float]]:
    """
    Returns the `nodes_xy` list sorted from closest to farthest
    relative to `point_xy`, using Euclidean distance (Shapely
    Point.distance)
    """
    if not isinstance(point_xy, tuple) or len(point_xy) != 2:
        raise ValueError("point_xy must be a tuple (x, y).")

    if nodes_xy is None:
        return []

    p = Point(point_xy[0], point_xy[1])

    # Calculating the distance for each node and sorting
    dist_and_node = []
    for n in nodes_xy:
        if not isinstance(n, tuple) or len(n) != 2:
            raise ValueError(f"Each node must be a tuple (x, y). Found:
{n!r}")
        d = p.distance(Point(n[0], n[1]))
        dist_and_node.append((d, n))

    dist_and_node.sort(key=lambda x: x[0])
    return [n for _, n in dist_and_node]
```

```
if __name__ == "__main__":  
    # Demo  
    point = (0.0, 0.0)  
    nodes = [(2.0, 2.0), (1.0, 1.0), (-1.0, 0.0), (0.0, 0.0)]  
    ordered = determine_nearest_nodes(point, nodes)  
    print("Point:", point)  
    print("Ordered nodes:", ordered)
```

Η συνάρτηση καταλληλότητας (new_fitness_hard.py)

Με το συγκεκριμένο script υπολογίζουμε την εγγύτητα των εικονικών με τους πραγματικούς κόμβους.

```
import numpy as np  
from tsp_2opt import tsp2opt  
from shapely_utils import determine_nearest_nodes  
def v_nodes_fitness(pelican_position, data):  
    # 1. Get Data  
    N = data['n_customers']  
    T = data['n_periods']  
    K = data['n_vehicles']  
    Q = data['vehicle_capacity']  
    C = N # Virtual Nodes (1 per customer)  
  
    distance_matrix = data['distance_matrix']  
    demands = data['demands']  
    holding_costs = data['holding_costs']  
    max_capacity = data['max_capacity']  
    supplier_holding_cost = data['supplier_holding_cost']  
    production_rate = data['production_rate']  
    supplier_max_inventory = float(data['supplier_max_inv'])  
  
    current_inventory = data['init_inv'].copy()  
    supplier_inventory = float(data['supplier_init_inv'])  
    # 2. Search area  
    coords = [tuple(x) for x in data['coordinates']]  
    coord_to_id = {}  
    for i in range(N + 1):  
        if coords[i] not in coord_to_id:  
            coord_to_id[coords[i]] = []  
            coord_to_id[coords[i]].append(i)  
  
    # 3. Decode pelican vectors (T days, C nodes, 2 Dimensions [X, Y])  
    positions = pelican_position.reshape((T, C, 2))  
    # Stock out as a soft constraint - Dynamic Penalties  
    current_iteration = data.get('current_iteration', 0) # First  
    iteration or not found --> 0  
    max_iteration = data.get('max_iterations', 1)  
    progress = current_iteration / max_iteration # Start (0.0) to End  
    (1.0)  
  
    # START_PENALTY = 100  
    # END_PENALTY = 1500
```

```
# LAMBDA STOCKOUT = 100.0
max_distance_from_depot = np.max(distance_matrix[0, :])
worst_case_route_cost = max_distance_from_depot * 2.0

START_PENALTY = worst_case_route_cost * 1.2
END_PENALTY = START_PENALTY * 3.0
LAMBDA_STOCKOUT = START_PENALTY + progress * (END_PENALTY -
START_PENALTY)
total_cost = 0.0

# Main loop
for t in range(T):
    supplier_inventory = min(supplier_inventory +
production_rate[t], supplier_max_inventory)

    # Candidate Customers
    candidates = [i for i in range(1, N + 1) if max_capacity[i] -
current_inventory[i] > 0.01]
    candidate_coords = [coords[i] for i in candidates]

    # Create customer sequence
    daily_sequence = []
    if supplier_inventory > 0.01 and candidates:
        for v in range(C):
            v_node = tuple(positions[t, v])
            ordered_coords = determine_nearest_nodes(v_node,
candidate_coords)

            found = False
            for coord in ordered_coords:
                for customer_id in coord_to_id[coord]:
                    if customer_id not in daily_sequence and
customer_id in candidates:
                        daily_sequence.append(customer_id)
                        found = True
                        break
            if found:
                break

    # Create routes
    routes = [[] for _ in range(K)]
    loads = [0.0] * K
    current_vehicle = 0

    for customer_id in daily_sequence:
        if supplier_inventory <= 0.01:
            break
        # Un-comment what will be used
        lookahead_horizon = min(t + 1, T)
        #lookahead_horizon = min(t + 2, T)
        #lookahead_horizon = min(t + 3, T)

        future_demand = np.sum(demands[customer_id,
t:lookahead_horizon])

        smart_space = max(0.0, future_demand -
current_inventory[customer_id])
```

```
        space = min(max_capacity[customer_id] -
current_inventory[customer_id], smart_space)
        desired_qty = min(space, supplier_inventory)

        # Hard Constraint: Vehicle gets its full capacity (Using
while loop for split deliveries)
        while desired_qty > 0.01:

            if current_vehicle >= K:
                break

            if loads[current_vehicle] >= Q:
                current_vehicle += 1
                if current_vehicle >= K:
                    break

            qty = min(desired_qty, Q - loads[current_vehicle])

            if qty > 0:
                routes[current_vehicle].append(customer_id)
                loads[current_vehicle] += qty
                supplier_inventory -= qty
                current_inventory[customer_id] += qty
                desired_qty -= qty # Reduce the need

                if customer_id in candidates:
                    candidates.remove(customer_id)

        # Calculating route cost (TSP)
        for k in range(K):
            if routes[k]:
                full_route = [0] + routes[k] + [0]
                dist, _ = tsp2opt(full_route, distance_matrix)
                total_cost += dist

        # Update Inventory & Holding/Stockout Costs
        current_inventory -= demands[:, t]

        for i in range(1, N + 1):
            if current_inventory[i] >= 0:
                # Normal inventory cost if customers inventory is OK
                total_cost += current_inventory[i] * holding_costs[i]
            else:
                # Soft Constraint: If a stockout occurred add a penalty
to the total cost
                total_cost += abs(current_inventory[i]) *
LAMBDA_STOCKOUT

        # Update Supplier Inventory
        if supplier_inventory >= 0:
            total_cost += supplier_inventory * supplier_holding_cost
        else:
            total_cost += abs(supplier_inventory) * LAMBDA_STOCKOUT

    return float(total_cost)
```

Η κλάση Pelican Optimization Algorithm (Pelican.py)

Η υλοποίηση του αλγόριθμου στην Python.

```
import numpy as np

class Pelican:
    def __init__(self, data, fitness, population, max_iterations, dim,
lb, ub):
        self.data = data
        self.fitness_function = fitness
        self.population = population
        self.max_iterations = max_iterations

        # Problem Dimensions
        self.problem_dimensions = dim
        self.lb = lb
        self.ub = ub

        # Population Initialization (Random numbers between 0, 500)
        self.x = np.random.uniform(self.lb, self.ub, (self.population,
self.problem_dimensions))

        # Initial Fitness Calculation
        self.init_fitness =
np.array([self.fitness_function(individual, self.data) for individual
in self.x])

        # Finding the best solution (Prey)
        self.best_solution = np.min(self.init_fitness)
        index_best_solution = np.argmin(self.init_fitness)
        self.prey_position = self.x[index_best_solution].copy()

        # Convergence Curve Diagram
        self.convergence_curve = []

    def pelican(self):
        for t in range(self.max_iterations):
            self.data['current_iteration'] = t # Settings for...
            self.data['max_iterations'] = self.max_iterations #
...dynamic penalty

            # self.init_fitness = np.array([self.fitness_function(ind,
self.data) for ind in self.x]) # Re-calculate after dynamic penalty
            # self.best_solution =
self.fitness_function(self.prey_position, self.data) # Keep in memory
the best-solution so far

            # Phase 1 (Exploration):
            for i in range(self.population):
                i_value = np.random.randint(1, 3)

                # Move towards prey or away
                if self.init_fitness[i] > self.best_solution:
                    x_new = self.x[i] + np.random.rand() *
(self.prey_position - i_value * self.x[i])
```

```
else:
    x_new = self.x[i] + np.random.rand() * (self.x[i]
- self.prey_position)

    # Bounds check (0, 500)
    x_new = np.clip(x_new, self.lb, self.ub)

    # New position check
    f_new = self.fitness_function(x_new, self.data)

    if f_new < self.init_fitness[i]:
        self.x[i] = x_new
        self.init_fitness[i] = f_new

    # Phase 2 (Exploitation):
    radius = 0.2 * (1 - t / self.max_iterations)

    # X_new = X + radius * (2*rand - 1) * X
    x_new = self.x[i] + radius * (2 *
np.random.rand(self.problem_dimensions) - 1) * self.x[i]

    # Bound check (0, 500)
    x_new = np.clip(x_new, self.lb, self.ub)

    f_new = self.fitness_function(x_new, self.data)

    # New position check
    if f_new < self.init_fitness[i]:
        self.x[i] = x_new
        self.init_fitness[i] = f_new
# End of iterations

current_best_value = np.min(self.init_fitness)
current_best_position = np.argmin(self.init_fitness)

# Update and keep the best solution
if current_best_value < self.best_solution:
    self.best_solution = current_best_value
    self.prey_position =
self.x[current_best_position].copy()
    print(f"Iteration {t+1}: New Best Found:
{self.best_solution:.2f}")
else:
    if (t+1) % 10 == 0: # Logging per 10 iterations
        print(f"Iteration {t+1}: Best Cost Remains:
{self.best_solution:.2f}")

    self.convergence_curve.append(self.best_solution)

return self.prey_position, self.best_solution,
self.convergence_curve
```

Η `tsp2opt` για την δημιουργία βέλτιστων διαδρομών (`tsp_2opt.py`)

Ο πιο απλός αλγόριθμος για την εύρεση βέλτιστων διαδρομών μεταξύ των κόμβων ενός δικτύου.

```
def calculate_route_distance(route, distance_matrix):
    distance = 0
    for i in range(len(route) - 1):
        distance += distance_matrix[route[i]][route[i + 1]]
    return distance

def tsp2opt(nodes_to_visit, distance_matrix):
    if not nodes_to_visit: # if are not any nodes to visit return to depot (0)
        return 0, []

    # Nearest Neighbor
    current_node = 0 # Depot
    unvisited_nodes = set(nodes_to_visit)
    route = [0]

    while unvisited_nodes:
        next_node = min(unvisited_nodes, key=lambda x:
distance_matrix[current_node][x])
        route.append(next_node)
        unvisited_nodes.remove(next_node)
        current_node = next_node

    route.append(0) # Return to depot

    # 2-opt local search
    best_distance = calculate_route_distance(route, distance_matrix)
    improved_route = True

    while improved_route:
        improved_route = False
        while improved_route:
            improved_route = False
            for i in range(1, len(route) - 2):
                for j in range(i + 1, len(route) - 1):
                    new_route = route[:i] + route[i:j + 1][::-1] +
route[j + 1:]
                    new_distance = calculate_route_distance(new_route,
distance_matrix)

                    if new_distance < best_distance:
                        route = new_route
                        best_distance = new_distance
                        improved_route = True

    return best_distance, route
```

Η συνάρτηση που εξάγει την τελική λύση (αναφορά) του instance (`new_improved_print_fitness.py`)

```
import numpy as np
```

```
from tsp_2opt import tsp2opt
from shapely_utils import determine_nearest_nodes

# CAUTION: Set lookahead the same value as lookahead horizon is set
further below
def critical_customers(initial_inventory, demands, t, lookahead=1):
    T = demands.shape[1]
    horizon = min(lookahead, T - t)
    future_demand = np.sum(demands[:, t:t+horizon], axis=1)
    projected_inventory = initial_inventory - future_demand
    return projected_inventory < 0 # JIT, was <= 0

def routing_costs(routes, distance_matrix):
    total = 0.0
    route_costs = []

    for route in routes:
        if route:
            full_route = [0] + route + [0]
            distance, _ = tsp2opt(full_route, distance_matrix)
            total += distance
            route_costs.append(distance)
        else:
            route_costs.append(0.0)

    return total, route_costs

def show_routes_v_nodes(pelican_position, data, f=None):
    N = data['n_customers']
    T = data['n_periods']
    K = data['n_vehicles']
    Q = data['vehicle_capacity']
    C = N

    distance_matrix = data['distance_matrix']
    demands = data['demands']
    holding_costs = data['holding_costs']
    max_capacity = data['max_capacity']

    supplier_holding_cost = data['supplier_holding_cost']
    production_rate = data['production_rate']
    supplier_max_inventory = float(data['supplier_max_inv'])

    current_inventory = data['init_inv'].copy()
    supplier_inventory = float(data['supplier_init_inv'])

    coords = [tuple(x) for x in data['coordinates']]
    coord_to_id = {}
    for i in range(N + 1):
        if coords[i] not in coord_to_id:
            coord_to_id[coords[i]] = []
        coord_to_id[coords[i]].append(i)

    positions = pelican_position.reshape((T, C, 2))

    # DO NOT USE
```

```
# LAMBDA STOCKOUT = 100.0 # Small penalty for stockouts
# LAMBDA STOCKOUT = 1500.0 # The maximum value for stockouts
according to dynamic

max_distance_from_depot = np.max(distance_matrix[0, :])
worst_case_route_cost = max_distance_from_depot * 2.0

START_PENALTY = worst_case_route_cost * 1.2
END_PENALTY = START_PENALTY * 3.0

LAMBDA_STOCKOUT = END_PENALTY

total_routing_cost = 0.0
total_holding_cost = 0.0

for t in range(T):
    print(f"\n{'='*70}", file=f)
    print(f"DAY {t+1}:", file=f)
    print(f"{'='*70}", file=f)

    supplier_inventory = min(supplier_inventory +
production_rate[t], supplier_max_inventory)
    print(f"Depot Inventory:
{supplier_inventory:.2f}/{supplier_max_inventory}", file=f)

    is_critical = critical_customers(current_inventory, demands,
t)

    candidates = [i for i in range(1, N+1) if max_capacity[i] -
current_inventory[i] > 0.01]
    candidate_coords = [coords[i] for i in candidates]

    daily_sequence = []
    if supplier_inventory > 0.01 and candidates:
        for v in range(C):
            v_node = tuple(positions[t, v])
            ordered_coords = determine_nearest_nodes(v_node,
candidate_coords)

            found = False
            for coord in ordered_coords:
                for customer_id in coord_to_id[coord]:
                    if customer_id not in daily_sequence and
customer_id in candidates:
                        daily_sequence.append(customer_id)
                        found = True
                        break
            if found:
                break

    print(f"Suggested Priority Sequence: {daily_sequence}",
file=f) # Outputs will be stored on a text file

    routes = [[] for _ in range(K)]
    loads = [0.0] * K
    q_delivered = np.zeros(N+1)
```

```
current_vehicle = 0

for customer_id in daily_sequence:
    if supplier_inventory <= 0.01:
        break

    # Un-comment what will be used
    lookahead_horizon = min(t + 1, T)
    # lookahead_horizon = min(t + 2, T)
    # lookahead_horizon = min(t + 3, T)
    future_demand = np.sum(demands[customer_id,
t:lookahead_horizon])

    smart_space = max(0.0, future_demand -
current_inventory[customer_id])

    space = min(max_capacity[customer_id] -
current_inventory[customer_id], smart_space)
    desired_qty = min(space, supplier_inventory)

    # Hard Constraint: Avoid violation on customer's inventory
    (Using while loop for split deliveries)
    while desired_qty > 0.01:

        if current_vehicle >= K:
            break

        if loads[current_vehicle] >= Q:
            current_vehicle += 1
            if current_vehicle >= K:
                break

        qty = min(desired_qty, Q - loads[current_vehicle])

        if qty > 0:
            routes[current_vehicle].append(customer_id)
            loads[current_vehicle] += qty
            supplier_inventory -= qty
            q_delivered[customer_id] += qty # Plus-equals for
split deliveries

            current_inventory[customer_id] += qty
            desired_qty -= qty # Reduce the need

            if customer_id in candidates:
                candidates.remove(customer_id)

    for customer in candidates:
        if is_critical[customer]:
            print(f" <!-- WARNING: Infeasible delivery for Customer
{customer} on Day {t+1} / STOCKOUT!", file=f)

    routing_cost, route_costs = routing_costs(routes,
distance_matrix)
    total_routing_cost += routing_cost

    print("\n** ROUTES **", file=f)
    for k in range(K):
```

```

        if routes[k]:
            display_route = [0] + routes[k] + [0]
            arrow = " -> ".join(map(str, display_route))

            print(f"Vehicle {k+1}:", file=f)
            print(f"--> Path: {arrow}", file=f)
            print(f"--> Load: {loads[k]:.2f}/{Q} | Cost: {route_costs[k]:.2f}", file=f)
            # BUG: Showing total qty, not load!!! Delivering qtys into vehicles are correct

            deliveries = ", ".join([f"Customer {c} ({q_delivered[c]:.2f})" for c in routes[k]])
            print(f"--> Drops: {deliveries}", file=f)
        else:
            print(f"Vehicle {k+1}: Not used", file=f)

        old_inventory = current_inventory.copy()
        old_inventory -= q_delivered
        current_inventory -= demands[:, t]

        daily_cost = 0
        print("\n** INVENTORY STATUS (End of Day) **", file=f)

        print(f"*{'Customer':^10}|{'Initial':^9}|{'Demand':^8}|{'Delivered':^11}|{'New Stock':^11}|{'Inv.Cost':^15}*", file=f)
        print("-" * 70, file=f)

        for i in range(1, N+1):
            if current_inventory[i] >= 0:
                cost = current_inventory[i] * holding_costs[i]
                status_inv_cost = f"{cost:.2f}"
            else:
                # Dynamic LAMBDA_STOCKOUT
                cost = abs(current_inventory[i]) * LAMBDA_STOCKOUT
                status_inv_cost = f"STOCKOUT! ({cost:.0f})"

            daily_cost += cost
            print(f"* {i:8} | {old_inventory[i]:7.2f} | {demands[i, t]:6.2f} | {q_delivered[i]:9.2f} | {current_inventory[i]:9.2f} | {status_inv_cost:13} *", file=f)

            total_holding_cost += daily_cost

            if supplier_inventory >= 0:
                total_holding_cost += supplier_inventory * supplier_holding_cost
            else:
                total_holding_cost += abs(supplier_inventory) * LAMBDA_STOCKOUT

        print("\n" + "=" * 70, file=f)
        print("=== FINAL RESULTS ===")
        print(f"Total Routing Cost: {total_routing_cost:.2f}", file=f)
        print(f"Total Holding Cost & Penalties: {total_holding_cost:.2f}", file=f)

```

```
print(f"TOTAL COST: {total_routing_cost +  
total_holding_cost:.2f}\n", file=f)
```

Το κύριο πρόγραμμα εκτέλεσης (multiprocessed_run_poa.py)

```
import os  
# Processing improvements  
os.environ["OMP_NUM_THREADS"] = "1"  
os.environ["OPENBLAS_NUM_THREADS"] = "1"  
os.environ["MKL_NUM_THREADS"] = "1"  
os.environ["VECLIB_MAXIMUM_THREADS"] = "1"  
os.environ["NUMEXPR_NUM_THREADS"] = "1"  
  
import glob  
import re  
import time  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
# import multiprocessing  
from concurrent.futures import ProcessPoolExecutor, as_completed  
  
from datetime import datetime  
from load_data import load_data  
from Pelican import Pelican  
  
from new_fitness_hard import v_nodes_fitness  
  
from new_improved_print_fitness import show_routes_v_nodes  
  
def get_sorted_instances(s):  
    return [int(text) if text.isdigit() else text.lower() for text in  
re.split(r'(\d+)', s)]  
  
# --- NEW: WORKER FUNCTION FOR MULTIPROCESSING ---  
def run_single_experiment(run_id, data, pop_size, max_iters,  
dimensions):  
    """  
    Executes a single run of the Pelican Optimization Algorithm.  
    Placed at the top level so it can be pickled by multiprocessing.  
    """  
    start_time = time.time() # Start timing  
  
    poa = Pelican(  
        data=data,  
        fitness=v_nodes_fitness,  
        population=pop_size,  
        max_iterations=max_iters,  
        dim=dimensions,  
        lb=0.0,  
        ub=500.0  
    )  
  
    best_position, best_cost, conv_curve = poa.pelican()  
  
    end_time = time.time() # Stop timing  
    run_time = end_time - start_time
```

```
        return run_id, best_cost, best_position, conv_curve, run_time
# -----

if __name__ == "__main__":
    # File/Repeat times
    FOLDER_PATH = "Process_Queue" # Folder for instance processing
    N_RUNS = 10 # 10 for better statistics

    # Store to file
    output_dir = "Process_Output" # Folder for the results
    os.makedirs(output_dir, exist_ok=True)
    timestamp = datetime.now().strftime("%Y%m%d_%H%M")

    # Sort files
    all_instances = glob.glob(os.path.join(FOLDER_PATH, "*.txt"))

    # Apply sort function
    all_instances.sort(key=get_sorted_instances)

    print(f"{len(all_instances)} file/-s to be processed for {N_RUNS}
time/-s each.")

    results_list = [] # List results

    for filepath in all_instances:
        filename = os.path.basename(filepath)
        print(f"\n{'-'*60}")
        print(f" Start processing: {filename} ")
        print(f"{'-'*60}")

        # Process reading error handling
        try:
            data = load_data(filepath)
        except Exception as e:
            import traceback
            traceback.print_exc()
            print(f"Error on processing {filename}. Skip to next.")
            continue

        # Set dimensions
        C = data['n_customers'] # - 1
        DIMENSIONS = data['n_periods'] * C * 2

        # --- ADAPTIVE HYPERPARAMETER TUNING ---
        if C <= 50:
            POP_SIZE = 30
            MAX_ITERATIONS = 500
        else:
            POP_SIZE = 50
            MAX_ITERATIONS = 800
        # -----

        print(f"|--> Adaptive Settings: Agents={POP_SIZE} |
Iterations={MAX_ITERATIONS}")

    # Keeping results
```

```
instance_costs = []
instance_times = []

# Store best solution
overall_best_cost = float('inf')
overall_best_position = None
costs = []
times = []

# Convergence curve data
instance_curves = []
best_curve_for_plot = None

# Determine optimal number of workers (leave 1 thread free for
OS stability)
# MAX_WORKERS = min(N_RUNS, max(1, multiprocessing.cpu_count()
- 1))

# Determine optimal number of workers / Cores (Use 4 for 6-core
CPU)
MAX_WORKERS = min(N_RUNS, 4)
print(f"|--> Parallel Execution started with {MAX_WORKERS}
workers...")

# --- PARALLEL EXECUTION BLOCK ---
with ProcessPoolExecutor(max_workers=MAX_WORKERS) as executor:
    # Submit all runs to the executor
    futures = [
        executor.submit(run_single_experiment, run, data,
POP_SIZE, MAX_ITERATIONS, DIMENSIONS)
        for run in range(N_RUNS)
    ]

    # As runs complete, collect their results
    for future in as_completed(futures):
        run_id, best_cost, best_position, conv_curve, run_time
= future.result()

        instance_costs.append(best_cost)
        instance_times.append(run_time)
        instance_curves.append(conv_curve)

    # If we have the best solution keep it
    if best_cost < overall_best_cost:
        overall_best_cost = best_cost
        overall_best_position = best_position.copy()
        best_curve_for_plot = conv_curve

    print(f"|--> Run {run_id+1}/{N_RUNS} completed -> Best
cost: {best_cost:.2f} | Time: {run_time:.2f} seconds.")
    # -----

# Statistics
min_cost = np.min(instance_costs)
max_cost = np.max(instance_costs)
mean_cost = np.mean(instance_costs)
median_cost = np.median(instance_costs)
```

```
std_cost = np.std(instance_costs)
mean_time = np.mean(instance_times)

# Statistics: Gap & CV Percents
gap_percent = ((mean_cost - min_cost) / min_cost) * 100 if
min_cost != 0 else 0
cv_percent = (std_cost / mean_cost) * 100 if mean_cost != 0 else
0

print(f"Statistics for {filename}:")
print(f"|--> Best: {min_cost:.2f} | Worst: {max_cost:.2f} | Mean:
{mean_cost:.2f} | Avg Time: {mean_time:.2f} sec")

# Best solution as an output on terminal (or .txt file)
print(f"\n|--> Showing solution for {filename} (Cost:
{overall_best_cost:.2f})")

# Write terminal output in a txt file
base_name = filename.replace('.txt', '')
txt_filename = os.path.join(output_dir,
f"Analysis_{base_name}.txt")
print(f"--> Solution is stored to: {txt_filename}")

with open(txt_filename, "w", encoding="utf-8") as f:
    show_routes_v_nodes(overall_best_position, data, f=f)

# Store all results in a dictionary for df or excel
experiment_result = {
    "Instance Name": filename,
    "Customers (N)": C, # C = N
    "Periods (T)": data['n_periods'],
    "Vehicles (K)": data['n_vehicles'],
    "Runs": N_RUNS,
    "Pop Size": POP_SIZE,
    "Iterations": MAX_ITERATIONS,
    "Best Cost": round(min_cost, 2),
    "Mean Cost": round(mean_cost, 2),
    "Worst Cost": round(max_cost, 2),
    "Std Deviation": round(std_cost, 2),
    "Median Cost": round(median_cost, 2),
    "Mean Time (sec)": round(mean_time, 2),
    "Gap Percent": round(gap_percent, 2), # Gap percent
    "CV Percent": round(cv_percent, 2), # CV Percent
}

results_list.append(experiment_result)

# Intermediate save (Backup) after each file
excel_filename = f"Pelican_V_Nodes_Results_{timestamp}.xlsx"
excel_path = os.path.join(output_dir, excel_filename)
pd.DataFrame(results_list).to_excel(excel_path, index=False)

# Convergence Curve (Gray for all, Red for Best Cost)
plt.figure(figsize=(10, 6))
for curve in instance_curves:
    plt.plot(curve, color='gray', alpha=0.3, linewidth=1)
```

```
plt.plot(best_curve_for_plot, color='red', linewidth=2.5,
label=f'Best Cost ({overall_best_cost:,.2f})')

plt.title(f"Convergence Curve - {base_name}", fontsize=14,
fontweight='bold')
plt.xlabel("Iterations", fontsize=12)
plt.ylabel("Total Cost", fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)

# Save the Convergence Curve as a picture
plot_filename = f"Convergence_{base_name}_{timestamp}.png"
plot_path = os.path.join(output_dir, plot_filename)
plt.savefig(plot_path, dpi=300, bbox_inches='tight')
plt.close()

# FINAL EXPORT (Un-indented to run once at the very end)
if results_list:

    # Convert to df (data frame)
    df = pd.DataFrame(results_list)

    # Formating
    cost_cols = ['Best Cost', 'Mean Cost', 'Worst Cost', 'Std
Deviation', 'Median Cost', 'Mean Time (sec)', 'Gap Percent', 'CV
Percent']
    for col in cost_cols:
        df[col] = df[col].apply(lambda x: f"{x:,.2f}".replace(',', '
X')).replace('.', ',').replace('X', '.')

    # Store to excel
    df.to_excel(excel_path, index=False)
    print(f"\n{'*' * 60}")
    print(f"- Processing ended succesfully!")
    print(f"- All results are stored to: {excel_path}")
    print(f"\n{'*' * 60}")
else:
    print("\n<!-- No files where processed.")
```