ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΜΙΜΗΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΤΟ ΑΝΟΙΧΤΟ ΠΡΟΒΛΗΜΑ ΔΡΟΜΟΛΟΓΗΣΗΣ ΟΧΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΒΑΛΛΙΑΝΑΤΟΣ ΝΙΚΟΛΑΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ

ΜΑΡΙΝΑΚΗΣ ΙΩΑΝΝΗΣ

ΧΑΝΙΑ 2024

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή μου, κύριο Μαρινάκη Ιωάννη, που μου έδωσε την ευκαιρία να συντάξω την έρευνα μου και να εντρυφήσω στον κλάδο της διαχείρισης εφοδιαστικών αλυσίδων, που με βοήθησε στην διεύρυνση των προγραμματιστικών μου γνώσεων και μου έδωσε κίνητρο για περαιτέρω έρευνα σε προβλήματα βελτιστοποίησης. Ευχαριστώ το Πολυτεχνείο Κρήτης που επένδυσε σε εμένα, αφιέρωσε χρόνο για την εκπαίδευση μου και μου παρείχε δωρεάν εργαλεία και συγγράμματα για την ανάπτυξή μου σε νέο Μηχανικό Παραγωγής και Διοίκησης. Ευχαριστώ την οικογένεια μου, τους γονείς μου και τις τρεις αδελφές μου, για την ανιδιοτελή στήριξη τους σε κάθε στάδιο της ζωής μου και της φοιτητικής μου πορείας. Ευχαριστώ τους φίλους και την σχέση μου για την δημιουργία αναμνήσεων και εμπειριών που θα νοσταλγώ για όλη μου την ζωή.

Περιεχόμενα

1. Περίληψη

Στην παρούσα εργασία θα παρουσιαστεί η επίλυση του Ανοιχτού Προβλήματος δρομολόγησης οχημάτων ,ενός από τους τύπους Προβλημάτων Δρομολόγησης Οχημάτων(VRP), με την βοήθεια του μεθευρετικού Μιμητικού Αλγορίθμου Νήσων. Θα αναλυθούν τα βήμα που ακολουθήθηκαν ενώ θα παρουσιαστούν αποτελέσματα από την επίλυση παραδειγμάτων για την περεταίρω κατανόηση του αλγορίθμου. Επίσης θα γίνει χρήση Αλγορίθμων Τοπικής Αναζήτησης για την περεταίρω βελτιστοποίηση του προβλήματος.

2. Εισαγωγή

Σε μια εποχή όπου η παγκόσμια οικονομία στηρίζεται από την μεταφορά αγαθών, ενώ έχει παρατηρηθεί πως η διευκόλυνση ή η δυσκολία αυτής μπορεί να επηρεάσει την αξία ενός προϊόντος, σε μία εποχή οπού η ανθρώπινη δραστηριότητα βασίζεται στα μέσα μεταφοράς και στα δίκτυα που αυτά χρησιμοποιούν, χρειάζεται μια μεθοδευμένη προσέγγιση της ώστε αυτή να πραγματοποιείται όσο το δυνατών γρηγορότερα και με όσο το δυνατόν λιγότερο κόστος είτε οικονομικό, είτε οικολογικό γίνεται. Έτσι προκύπτουν τα προβλήματα δρομολόγησης οχημάτων(VRP). Σκοπός της εργασίας είναι η παραγωγή ενός προγράμματος εύχρηστου το οποίο θα έχει την δυνατότητα επίλυσης του κάθε ανοιχτού προβλήματος δρομολόγησης οχημάτων.

2.1 Δομή Εργασίας

Η Παρούσα διπλωματική αποτελείται από 6 Κεφάλαια. Στο πρώτο, της περίληψης και σε αυτό γίνεται μια εισαγωγή στις έννοιες που θα παρουσιαστούν και θα αναπτυχθούν παρακάτω. Στο τρίτο κεφάλαιο θα αναπτυχθεί ο ορισμός και η θεωρία του Ανοιχτού Προβλήματος Δρομολόγησης Οχημάτων καθώς και των δύο αλγορίθμων που αναφέρθηκαν του Μιμητικού Νήσων (IMA) και Τοπικής Αναζήτησης (LS). Στο τέταρτο κεφάλαιο θα γίνει παράθεση όλης της μεθοδολογίας και της μαθηματικής μοντελοποίησης του κάθε υποστοιχείου των αλγορίθμων με βάση το Ανοιχτό VRP. Στο πέμπτο κεφάλαιο παρατίθεται ο κώδικα που αναπτύχθηκε σε γλώσσα προγραμματισμού Python για την επίλυση του προβλήματος. Τέλος στο έκτο κεφάλαιο παρουσιάζονται τα αποτελέσματα του προγράμματος ενώ καταληκτικά αναφέρονται και τα συμπεράσματα που αντλήθηκαν.

3. Θεωρία

3.1 Ανοιχτό Πρόβλημα Δρομολόγησης Οχημάτων

Θέτοντας **ένα πρόβλημα δρομολόγησης οχημάτων** (Vehicle Routing Problem-VRP) αναζητούνται βέλτιστες λύσεις-μονοπάτια πάνω σε έναν χάρτη στον οποίο το προϊόν μιας εγκατάστασης (π.χ. γραμμή παραγωγής, αποθήκη) θα πρέπει να μεταφερθεί ώστε να καλύψει τη ζήτηση του σε διάφορα σημεία-κόμβους (π.χ. σημεία πώλησης).[5] Η διανομή του προϊόντος στους κόμβους περιορίζεται όμως από διάφορους παράγοντες όπως είναι η **χωρητικότητα** του κάθε οχήματος[5], ο **χρόνος εξυπηρέτησης του κάθε κόμβου**, ο **μέγιστος χρόνος λειτουργίας** και η **μέγιστη απόσταση** που μπορεί να διανύσει αυτό σε κάθε διαδρομή για την εξυπηρέτηση της ζήτησης. Χωρίς την βελτιστοποίηση της εφοδιαστικής αλυσίδας η εξυπηρέτηση της συνολικής ζήτησης μπορεί να αποδειχθεί κοστοβόρα και χρονοβόρα.

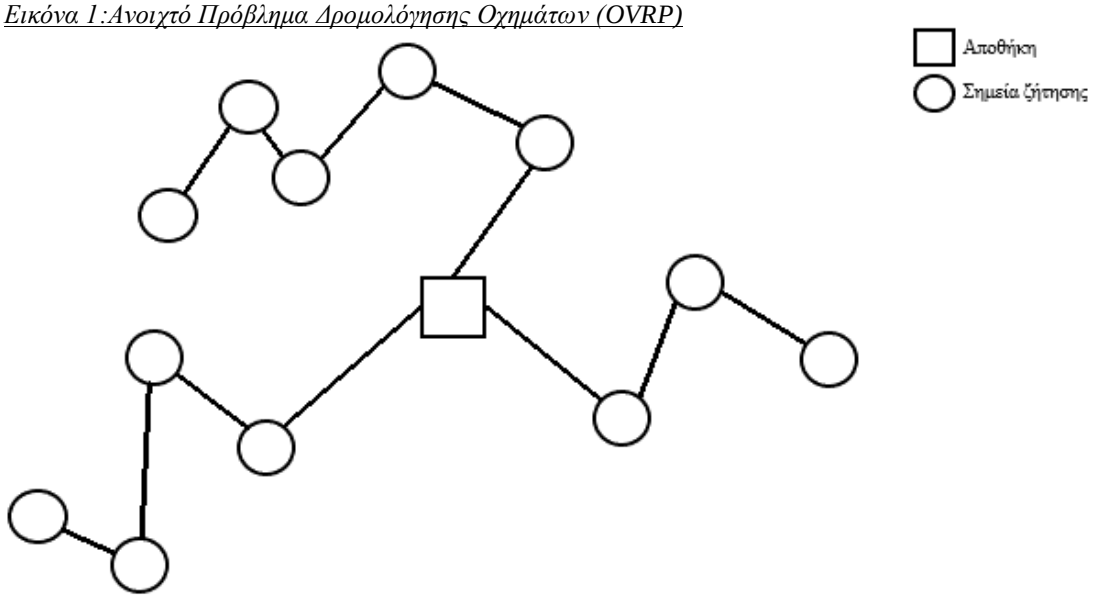
Έτσι λοιπόν, για την λύση ενός προβλήματος δρομολόγησης οχημάτων VRP πρέπει αναπτυχθεί ένα πλάνο εξυπηρέτησης το οποίο ακολουθώντας τους παραπάνω περιορισμούς θα διακρίνει τα εξής βασικά αποτελέσματα:

- **Τον συνολικό αριθμό διαδρομών** που εκτελούνται από την εγκατάσταση της επιχείρησης ώστε να καλυφθεί η ζήτηση όλων των κόμβων.
- **Την σειρά εξυπηρέτησης των κόμβων** που θα επισκεφθούν τα οχήματα εκτελώντας την κάθε διαδρομή ξεκινώντας από την εγκατάσταση.

- **Τον συνολικό χρόνο και την διανυόμενη απόσταση** για τον υπολογισμό του κόστους και των εξόδων μεταφοράς.

Το **ανοιχτό πρόβλημα δρομολόγησης οχημάτων** (Open Vehicle Routing Problem-OVRP) αποτελεί μια παραλλαγή του κλασσικού προβλήματος δρομολόγησης οχημάτων. Στο OVRP τα οχήματα δεν επιστρέφουν στην εγκατάσταση της εταιρίας για την ανάθεση καινούργιου δρομολογίου αλλά τερματίζουν την εξυπηρέτηση τους στον τελικό κόμβο της διαδρομής που μόλις εκτέλεσαν, από εκεί θέτονται ελεύθερα ώστε να αναλάβουν μια καινούργια εργασία με αρχή τον κόμβο που τερμάτισαν. Σκοπός του OVRP είναι η περεταίρω μείωση της διανυόμενης απόστασης αποφεύγοντας την διαδρομή της επιστροφής, αποσκοπεί επίσης στην χρήση οχημάτων για την κάλυψη αναγκών πολλαπλών εφοδιαστικών αλυσίδων.

Εικόνα 1:Ανοιχτό Πρόβλημα Δρομολόγησης Οχημάτων (OVRP)

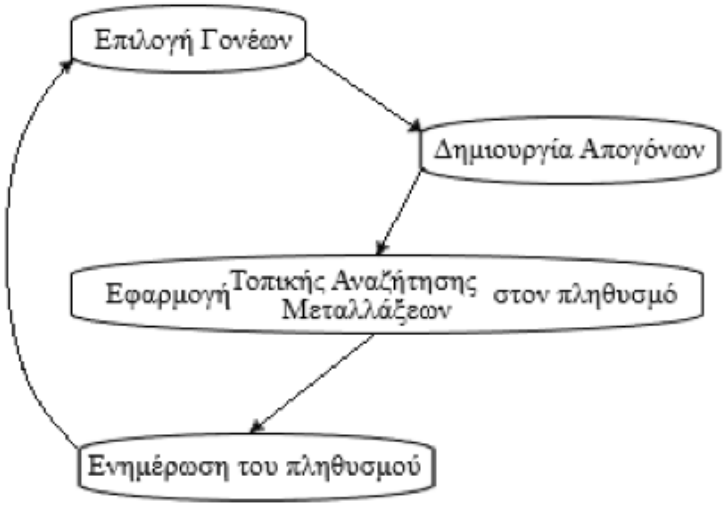


3.2 Μιμητικός Αλγόριθμος Νήσων

Στην επιστήμη των υπολογιστών και συγκεκριμένα στον τομέα της βελτιστοποίησης, μια μεθευρετική διαδικασία ονομάζεται μια διαδικασία που έχει σχεδιαστεί για να βρίσκει, να παράγει ή να συντονίζει μια ευρετική διαδικασία με σκοπό να παρέχει μια αρκετά καλή λύση σε ένα πρόβλημα βελτιστοποίησης, ειδικά με ελλιπείς ή ατελείς πληροφορίες ή περιορισμένη διαθέσιμη υπολογιστική ισχύ [6][7]. Οι μεθευρετικές διαδικασίες μπορεί να κάνουν σχετικά λίγες υποθέσεις σχετικά με το πρόβλημα βελτιστοποίησης που επιλύεται και έτσι μπορούν να χρησιμοποιηθούν για μια ποικιλία προβλημάτων ειδικά όταν ο χώρος του προβλήματος είναι πολύ μεγάλος.

Στην παρούσα εργασία θα αναλυθεί η λειτουργία του μιμητικού αλγόριθμου και τα βήματα του για την βελτιστοποίηση μιας διαδικασίας. Ο **Μιμητικός αλγόριθμος (Memetic Algorithm - MA**), εμπνευσμένος από την βιολογία και την γενετική, είναι μεθευρετικός αλγόριθμος, βασισμένος σε πληθυσμούς, που πλαισιώνεται σε ένα εξελικτικό περιβάλλον και αποτελείται από ένα σύνολο αλγορίθμων τοπικής αναζήτησης.[1] Έχει ως σκοπό να λύσει ένα πρόβλημα βελτιστοποίησης από την τυχαία διασταύρωση ενός πληθυσμού λύσεων την δημιουργία καινούργιων απογόνων-λύσεων από αυτή τη διασταύρωση και την επιβίωση των καλύτερων λύσεων σε επόμενες γενιές, όπου θα λάβουν μέρος καινούργιες διασταυρώσεις για την παραγωγή καινούργιων λύσεων. Όπως η αλληλουχία των γονιδίων του DNA προκαθορίζει την επιβίωση ενός οργανισμού και την δυνατότητα του να αναπαραχθεί στην φύση έτσι, όσο πιο κοντά βρίσκεται μια λύση στην βέλτιστη τόσο πιο πιθανό είναι αυτή να επιβιώνει ανά τις γενεές και να διασταυρώνεται με άλλες πιθανές λύσεις για την παραγωγή καινούργιων γενεών. [3].Αναλυτικότερα, ο μιμητικός αλγόριθμος ακολουθεί τα εξής βασικά βήματα:

Εικόνα 2:Βασικά βήματα Μιμητικού αλγορίθμου



- **Επιλογή γονέων**: Η επιλογή γονέων αποσκοπεί στον προσδιορισμό των υποψήφιων λύσεων που θα επιβιώσουν στις επόμενες γενιές και θα χρησιμοποιηθούν για τη δημιουργία νέων λύσεων. Η επιλογή λειτουργεί συχνά σε σχέση με την καταλληλόλητα (απόδοση) των υποψήφιων λύσεων προς αναπαραγωγή. Η απόδοση συνήθως ανέρχεται στο βαθμό στον οποίο η λύση μεγιστοποιεί/ελαχιστοποιεί την αντικειμενική συνάρτηση του προβλήματος που τίθεται προς λύση. [3]

- **Δημιουργία απογόνων**: Αποσκοπεί στη δημιουργία νέων υποσχόμενων υποψήφιων λύσεων μέσω της ανάμειξης υφιστάμενων λύσεων (γονέων), ενώ μια λύση είναι υποσχόμενη εάν μπορεί δυνητικά να οδηγήσει τη διαδικασία βελτιστοποίησης σε νέες περιοχές αναζήτησης όπου μπορούν να βρεθούν καλύτερες λύσεις. [3]

- **Εφαρμογή Τοπικής αναζήτησης/Μεταλλάξεων στον πληθυσμό**: Ο στόχος της όσο το δυνατόν περισσότερο μέσα σε μία γενιά. Μεταλλάξεις πραγματοποιούνται τυχαία στον πληθυσμό για να επιτευχθεί μεγαλύτερη ποικιλομορφία λύσεων και να αποτραπεί η διαδικασία αναζήτησης βέλτιστης λύσης από πρόωρη σύγκλιση (δηλαδή, πολύ γρήγορη σύγκλιση προς μια υποβέλτιστη περιοχή του χώρου αναζήτησης). [3]
- **Ενημέρωση Πληθυσμού**: Σε αυτό το βήμα αποφασίζεται αν μια νέα λύση πρέπει να γίνει μέλος του πληθυσμού και ποια υπάρχουσα λύση του πληθυσμού θα πρέπει να αντικατασταθεί. Συχνά, οι αποφάσεις αυτές λαμβάνονται σύμφωνα με κριτήρια που

σχετίζονται με την ποιότητα και την ποικιλομορφία. Οι πολιτικές που χρησιμοποιούνται για τη διαχείριση του πληθυσμού είναι απαραίτητες για τη διατήρηση της κατάλληλης ποικιλομορφίας του πληθυσμού, για να αποτραπεί η διαδικασία αναζήτησης βέλτιστης λύσης από πρόωρη σύγκλιση. [3]

Στην συγκεκριμένη διπλωματική εργασία θα αναπτυχθεί ο **Μιμητικός Αλγόριθμος Νήσων (Island Memetic Algorithm - IMA).** Πρόκειται για μια παραλλαγή του κλασσικού ΜΑ αλγορίθμου κατά τον οποίο τα βήματα του αλγορίθμου εφαρμόζονται όχι σε έναν αλλά σε πολλούς πληθυσμούς ξεχωριστά. Χωρίζεται δηλαδή ο συνολικός πληθυσμός των λύσεων σε νήσους στις οποίες πραγματοποιείται η διαδικασία του μιμητικού αλγορίθμου ξεχωριστά (διασταύρωση, δημιουργία απογόνων, μεταλλάξεις, τοπική αναζήτηση, ενημέρωση του πληθυσμού). Ανά γενιές πραγματοποιείται επίσης η διαδικασία της **μετανάστευσης**, ώστε να επιτευχθεί η ανταλλαγή χαρακτηριστικών- πληροφοριών μεταξύ των πληθυσμών των νήσων.

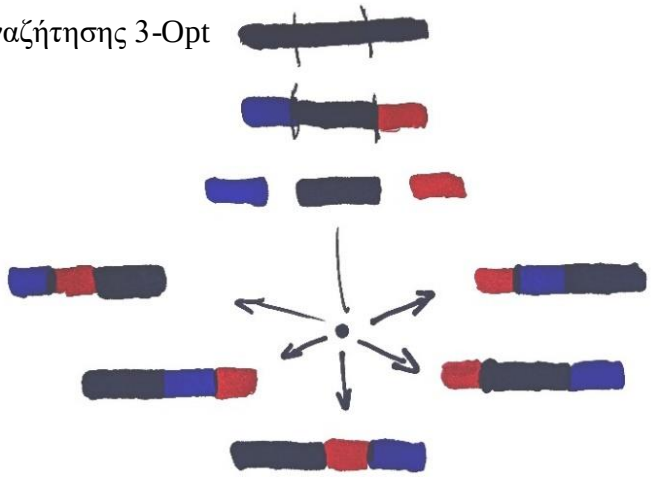
3.3 Αλγόριθμοι τοπικής αναζήτησης

Οι **Αλγόριθμοι τοπικής αναζήτησης (Local Search Algorithms – LS)** μια γενικώς εφαρμόσιμη προσέγγιση που μπορεί να χρησιμοποιηθεί για την εύρεση προσεγγιστικών λύσεων σε δύσκολα προβλήματα βελτιστοποίησης. Ξεκινάει από μία ήδη υπάρχουσα λύση του προβλήματος και προσπαθεί να βρει βέλτιστες λύσεις στη γειτονική περιοχή των εφικτών λύσεων. Ως γειτονιά λύσεων ορίζουμε τις εφικτές λύσεις με την μικρότερη δυνατή διαφορά από την υπάρχουσα. [4]

Η τοπική αναζήτηση για ένα VRP μπορεί να μεταφραστεί σε επιλογή k-Opt σημείων σε ένα διάνυσμα κόμβων, η διαχώριση τμημάτων του διανύσματος, η εναλλαγή της σειράς με την οποία αυτά παρουσιάζονται και η επανασύνδεση τους για την δημιουργία των γειτονικών λύσεων. Στην βιβλιογραφία αναφέρονται 3 τύποι k-Opt (2-Opt, 3-Opt, Lin-Kernighan). [4]

Στην συγκεκριμένη εργασία επιλέχθηκε και χρησιμοποιείται η 3-Opt τοπική αναζήτηση, κατά την οποία επιλέγονται 2 σημεία της πιθανής λύσης, τα οποία χωρίζουν ένα διάνυσμα σε 3 επιμέρους διανύσματα και αυτά τυχαία ξαναπαίρνουν θέση δημιουργώντας μία καινούργια λύση. Έπειτα αυτή συγκρίνεται με την αρχική λύση και επιλέγεται η ιδανικότερη για το πρόβλημα ανάλογα την επίδοση της. Αυτή η απλή διαδικασία λαμβάνει χώρα στη λύση επανειλημμένες φορές, χωρίζοντας ο αλγόριθμος κάθε φορά σε νέα τυχαία σημεία την λύση αναζητώντας κάθε φορά την βέλτιστη απόδοση της.[2]

Εικόνα 3:Σχεδιάγραμμα Αλγορίθμου Τοπικής Αναζήτησης 3-Opt



4. Μοντελοποίηση Προβλήματος

Σκοπός του κώδικα, που αναπτύχθηκε για τις ανάγκες της διπλωματικής εργασίας, είναι να προτείνει ένα βέλτιστο μονοπάτι - λύση για την κάλυψη της ζήτησης των κόμβων των οποίων του καταχωρούνται, λαμβάνοντας υπόψιν την χωρητικότητα των οχημάτων, ο μέγιστος χρόνος διαδρομής του κάθε οχήματος και τον χρόνο εξυπηρέτησης των πελατών του κάθε κόμβου.

4.1 Καταχώρηση Μεταβλητών

Είναι λοιπόν απαραίτητη η **καταχώρηση των μεταβλητών** αυτόν από τον χρήστη και η κατάλληλη επεξεργασία τους για την κλήση τους στον αλγόριθμο.

- **Αποστάσεις μεταξύ Κόμβων**: Εκχωρώντας τις καρτεσιανές συντεταγμένες όλων των κόμβων εξυπηρέτησης και της αποθήκης, στον πίνακα *A* υπολογίζονται και παρουσιάζονται οι ευκλείδειες αποστάσεις μεταξύ όλων των δεδομένων κόμβων.

$$A_{ij} = \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1N} \\ \vdots & \ddots & \vdots \\ \alpha_{N1} & \cdots & \alpha_{NN} \end{bmatrix} i, j = 1,2, \dots N \ (\text{όπου } N = \text{συνολικός αριθμός κόμβων})$$

Σε κάθε στοιχείο *α(i , j)* καταχωρείται η ευκλείδεια απόσταση μεταξύ των κόμβων *i , j.*

- **Ζήτηση**: Στο διάνυσμα d καταχωρείται η ζήτηση των πελατών του κάθε κόμβου.

  $$d_i = [\, d_1, \quad d_2, \dots \quad d_N] \ (\text{όπου } N = \text{συνολικός αριθμός κόμβων})$$

  Το d_i στοιχείο δείχνει την ζήτηση στον κόμβο *i.*

- Καταχωρούνται επίσης οι σταθερές, για το κάθε πρόβλημα, τιμές της χωρητικότητας των οχημάτων *(Cap)*, του μέγιστου χρόνου διαδρομής *(Tmax)* και του χρόνου εξυπηρέτησης των πελατών *(S)*.

4.2 Αντικειμενική Συνάρτηση & Περιορισμοί

Αντικειμενική Συνάρτηση

Έστω x μία πιθανή λύση, ένα διάνυσμα που n ο συνολικός αριθμός στοιχείων που την απαρτίζουν, δηλαδή $\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3} \dots, \boldsymbol{x_n})$, όπου τα x_i τιμές διακριτές ή συνεχείς ανάλογα με τις ανάγκες του προβλήματος προς επίλυση. Κατά την μοντελοποίηση ενός προβλήματος για την επίλυσή του είναι απαραίτητη η δημιουργία της **αντικειμενικής συνάρτησης** (μονοσήμαντο πρόβλημα: μία συνολικά αντικειμενική συνάρτηση \ πολυσήμαντο πρόβλημα: περισσότερες από μια αντικειμενικές συναρτήσεις) για την οποία ζητάμε την ελάχιστη ή μέγιστη τιμή της. Για την σωστή μοντελοποίηση του προβλήματος πρέπει να ληφθούν υπ' όψη και οι κατάλληλοι περιορισμοί οι οποίοι μπορεί να πλαισιώνουν ένα πρόβλημα. [3]

Παράδειγμα 1: Μοντελοποίηση Μονοσήμαντου προβλήματος

min/ max	f(x)	Αντικειμενική Συνάρτηση
Υπό.	g(x)≤G	Περιορισμός g συνάρτησης μικρότερης της σταθεράς G
	h(x)≥H	Περιορισμός h συνάρτησης μεγαλύτερης της σταθεράς H

$$x_i^L \leq x_i \leq x_i^U \qquad \text{Πεδίο Ορισμού των x τιμών } [x_i^L, x_i^U]$$

Η λύση $\boldsymbol{x}$ ενός OVRP θα αποτελεί ένα διάνυσμα διακριτών τιμών όπου το κάθε στοιχείο $\boldsymbol{x_i}$ του αντιπροσωπεύει τον κόμβο που επισκέπτεται ένα όχημα την χρονική στιγμή i.

Η Αντικειμενική Συνάρτηση στην περίπτωση της επίλυσης του OVRP είναι η συνολική απόσταση που διανύουν τα οχήματα που ξεκινούν από την αποθήκη.

$$\boldsymbol{f} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N-1}} (\, \boldsymbol{A_{x_i, x_{i+1}}} + \boldsymbol{S}\,) \;\; \forall\, \boldsymbol{x_{i+1}} \neq \boldsymbol{1} \quad (4.2.1)$$

Το άθροισμα δηλαδή των αποστάσεων των κόμβων του διανύσματος *x* όπως παρουσιάζονται στον πίνακα *A* συν τον χρόνο εξυπηρέτησης των πελατών *S* ο οποίος είναι σταθερός για κάθε κόμβο. Η μαθηματική έκφραση $x_{i+1} \neq 1$ είναι απαραίτητη καθώς θα πρέπει να παραληφθεί στο άθροισμα των κόμβων μιας πιθανής λύσης η επιστροφή ενός οχήματος στην αποθήκη.

Παράδειγμα 2: $\boldsymbol{x} = (\ldots, \mathbf{23}, \mathbf{31}, \mathbf{1}, \mathbf{17}, \ldots)$

Στην προκειμένη περίπτωση ενός OVRP ένα όχημα αφού επισκεφθεί τον κόμβο 23 έπειτα θα επισκεφθεί τον κόμβο 31 για να καλύψει την ζήτηση των πελατών και θα τερματίσει εκεί, ενώ ένα άλλο όχημα θα ξεκινήσει από το κόμβο 1 (αποθήκη) και θα μετακινηθεί προς τον κόμβο 17 και ύστερα. Κατά τον υπολογισμό της αντικειμενικής συνάρτησης πρέπει να αποφευχθεί, λοιπόν ο υπολογισμός της απόστασης $A_{31,1}$.

Σημείωση: Είναι σημαντικό να αναφερθεί ότι στην εργασία για την επίλυση του συγκεκριμένου προβλήματος **οι μονάδες χρόνου και απόστασης είναι ταυτόσημες**. Απαιτείται η ελαχιστοποίηση τόσο της μεταβλητής του χρόνου όσο και της απόστασης, οι οποίες στα προβλήματα VRP είναι αλληλένδετες και εξαρτώμενες η μία από την άλλη.

Περιορισμοί

- Η ζήτηση των κόμβων μίας διαδρομής δεν πρέπει να ξεπερνά την χωρητικότητα των οχημάτων.

$$\sum_{i=i_1}^{i_2-1} \boldsymbol{d_{x_i}} \leq \boldsymbol{Cap} (4.2.2)$$

Όπου i_1, i_2 παίρνουν τις τιμές δύο διαδοχικών εμφανίσεων του κόμβου 1 στο διάνυσμα της λύσης *x*. Καθώς αυτές σηματοδοτούν την έναρξη και το τέλος μιας διαδρομής σε μία λύση.

- Ο χρόνος μίας διαδρομής να μην υπερβαίνει το μέγιστο *(Tmax)*.

$$\sum_{i=i_1}^{i_2-2} (\boldsymbol{A_{x_i, x_{i+1}}} + \boldsymbol{S}) \leq \boldsymbol{Tmax} (4.2.3)$$

Όπου i_1, i_2 παίρνουν τις τιμές δύο διαδοχικών εμφανίσεων του κόμβου 1 στο διάνυσμα της λύσης *x*.

- Καμία από τις τιμές να μην παίρνει αρνητικές τιμές.

$\boldsymbol{A_{ij}}, \boldsymbol{d_i}$,Cap, Tmax, S, $\boldsymbol{x_i} \geq 0$ (4.2.4)

Με την μοντελοποίηση του προβλήματος πλέον γίνεται εφικτή η ανάπτυξη των βημάτων του Μιμητικού Αλγορίθμου Νήσων για την **ελαχιστοποίηση *της Αντικειμενικής Συνάρτησης f*** όπως μας προστάζει η έννοια του OVRP**.**

$$\boldsymbol{min\ f} = \sum_{i=1}^{N-1} (\boldsymbol{A_{x_i, x_{i+1}}} + \boldsymbol{S}) \quad \forall\ \boldsymbol{x_{i+1}} \neq \boldsymbol{1} (4.2.5)$$

4.3 IMA

Όπως προαναφέρθηκε, ο IMA βασίζεται στην εξέλιξη **ενός πληθυσμού λύσεων $\boldsymbol{x_i}$** μέσω της διασταύρωσης τους ανά γενιά και της επιβίωσης τους στην επόμενη ανάλογα με την επίδοση τους στην αντικειμενική συνάρτηση. Οι γενιές καθορίζουν την επαναληψημότητα του αλγορίθμου, πόσες φορές θα επαναληφθεί, δηλαδή, η διαδικασία της <u>εικόνας 2</u>. Ο συνολικός αριθμός των γενεών *(Gen)* εκχωρείται από τον χρήστη καθώς η σωστή επίλυση και επιλογή επαναλήψεων καθορίζεται από το μέγεθος του προβλήματος OVRP, δηλαδή τον αριθμό τον κόμβων που εκκρεμεί η ζήτηση παραγγελιών.

4.3.1 Δημιουργία Αρχικού πληθυσμού λύσεων

Το πρώτο βήμα για την εφαρμογή του ΙΜΑ στο πρόβλημα είναι η δημιουργία του πληθυσμού λύσεων. Μια λύση αντιπροσωπεύει ένα χρωμόσωμα και ο πληθυσμός των λύσεων-χρωμοσωμάτων εκφράζεται ως ένας πίνακας *Pop* με σειρές που αντιπροσωπεύουν το κάθε χρωμόσωμα και με στήλες που εκπροσωπούν την σειρά με την οποία τα οχήματα επισκέπτεται τους κόμβους.

$$Pop = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,N} \\ x_{2,1} & x_{2,2} & \dots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M,1} & x_{M,2} & \dots & x_{M,N} \end{bmatrix} \quad (4.3.1.1)$$

$$\text{ό}\boldsymbol{\pi ου}\ \ \boldsymbol{x_{1,*} = (x_{1,1}, x_{1,2}, x_{1,3} \dots, x_{1,N})} \rightarrow 1^{o}\ Χρωμόσωμα - Λύση$$

$$\boldsymbol{x_{2,*} = (x_{2,1}, x_{2,2}, x_{2,3} \dots, x_{2,N})} \rightarrow 2^{o}\ Χρωμόσωμα - Λύση$$

$$\vdots$$

$$\boldsymbol{x_{M,*} = (x_{M,1}, x_{M,2}, x_{M,3} \dots, x_{M,N})} \rightarrow M^{o}\ Χρωμόσωμα - Λύση$$

Σημείωση: Στον συγκεκριμένο πίνακα, παρατηρείτε ότι όλα τα χρωμοσώματα έχουν τον ίδιο μέγεθος N-στοιχείων (= αριθμό συνολικών κόμβων N). Αυτός προκύπτει καθώς η ζήτηση των πελατών των κόμβων εξυπηρετείται όλη κατά την επίσκεψη ενός οχήματος σε κάθε κόμβο, ενώ επίσης πρέπει να επισκεφθούν όλοι οι κόμβοι για τη εξυπηρέτηση τους. Έτσι, όλοι οι κόμβοι (που στο σύνολό τους είναι N) εμφανίζονται μία φορά στο διάνυσμα x. Επίσης, για κάθε φορά που οι περιορισμοί (4.2.2), (4.2.3) *της αντικειμενικής συνάρτησης δεν πληρούνται, η αποστολή νέου δρομολογίου από την αποθήκη (Κόμβος 1) εννοείται και η αναφορά της μπορεί να παραληφθεί. Συνεπώς, και ο κόμβος της αποθήκης εμφανίζεται μία μόνο φορά στην αρχή του κάθε διανύσματος x.*

Για τη δημιουργία του αρχικού πληθυσμού χρωμοσωμάτων λύσεων απαιτείται η καταχώρηση 2 σταθερών:

- **του αριθμού των κόμβων(N)**, η οποία σταθερά καταχωρείται, συνήθως, από το αρχείο των δεδομένων.
- **το μέγεθος του πληθυσμού(M)**, το οποίο ανάλογα με τις απαιτήσεις του προβλήματος καταχωρείται από τον χρήστη.

Οι αρχικές τιμές των χρωμοσωμάτων του πίνακα $\mathrm{Pop}_{\mathrm{M,N}}$(4.3.1.1) συμπληρώνονται με δύο τρόπους. **50% του πληθυσμού δημιουργείται τυχαία** και **50% του πληθυσμού δημιουργείται με την μέθοδο GRASP**.[1]

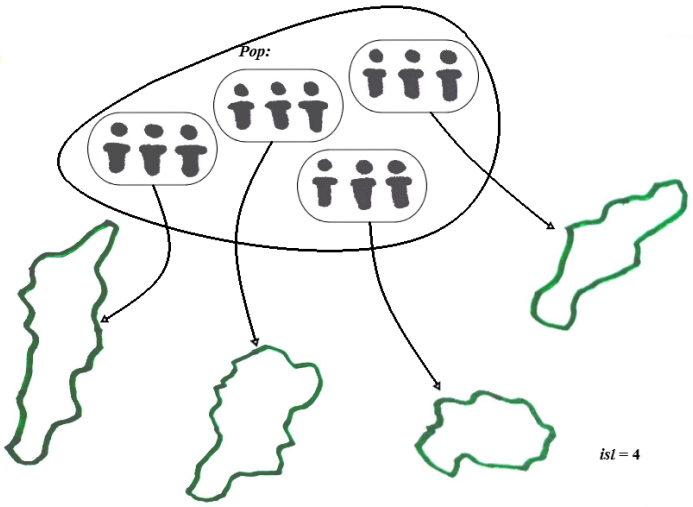
- Τυχαία καταχώρηση: Η σειρά επίσκεψης των κόμβων δεν ακολουθεί κάποιον κανόνα ή αλγόριθμο. Αρχίζοντας από τον πρώτο κόμβο η επιλογή και επίσκεψη του επόμενου γίνεται τυχαία και ανεξάρτητα από την θέση του προηγούμενου. Κάθε κόμβος, όπως προαναφέρθηκε, επισκέπτεται μόνο μία φορά.
- Μέθοδος **GRASP (Greedy Randomized Adaptive Search Procedure):** Οι μέθοδος GRASP αποτελείται από δύο μέρη. Στο πρώτο μέρος, δημιουργούνται οι λύσεις με τον εξής τρόπο. Αρχίζοντας από τον κόμβο της αποθήκης η μέθοδος GRASP, αρχικά καταχωρεί σε κάθε κόμβο ένα βάρος, ανάλογα με την απόσταση του από τον κόμβο που εξετάζεται σε σχέση με τους άλλους κόμβους. Η επιλογή του επόμενου κόμβου, γίνεται τυχαία όμως όσο μεγαλύτερο το βάρος ενός κόμβου τόσο μεγαλύτερη η πιθανότητα να επιλεχθεί αυτός αντί των άλλων. Ύστερα, στο δεύτερο μέρος, σε κάθε μία από τις λύσεις εφαρμόζεται ο αλγόριθμος τοπικής αναζήτησης 3-Opt (3.3 -Εικόνα 4).[1]

4.3.2 Δημιουργία Νήσων & Μετανάστευση

Για να αρχίσει η διαδικασία του ΙΜΑ ο αρχικός πληθυσμός του πίνακα Pop χωρίζεται σε υποπληθυσμούς. Ο κάθε ένας από αυτούς αντιπροσωπεύει ένα νησί και στον κάθε έναν λαμβάνει χώρα η διαδικασία του ΜΑ ξεχωριστά.

Από τον χρήστη καταχωρείται ο αριθμός των νήσων(isl), και αφού ήδη έχει δημιουργηθεί ο αρχικός πληθυσμός, μοιράζεται ισόποσα στους πίνακες των υποπληθυσμών του. **Ο αριθμός των χρωμοσωμάτων που απαρτίζει κάθε υποπληθυσμό υπολογίζεται,** δηλαδή, **από την διαίρεση M/isl.**

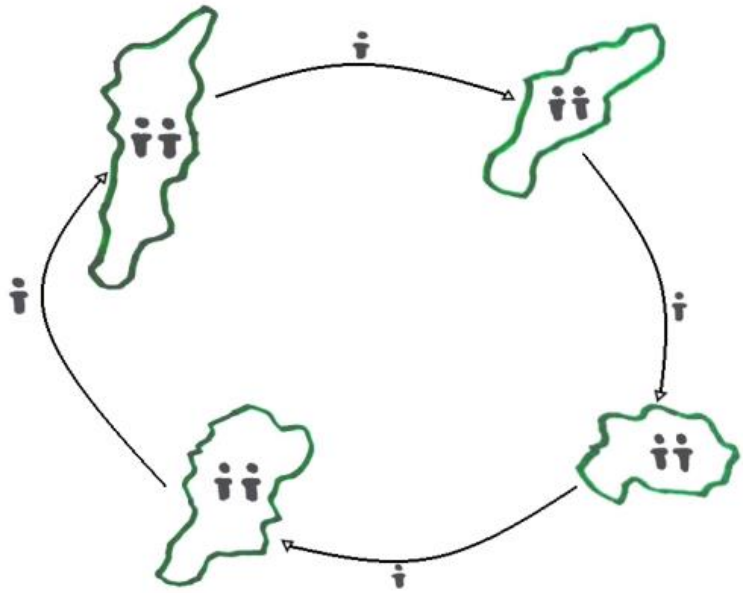
Εικόνα 4: Διαίρεση Αρχικού Πληθυσμού



Η **μετανάστευση** είναι η μεταφορά μερικών λύσεων του πληθυσμού μιας νήσου στον πληθυσμό μιας άλλης, και αποσκοπεί στην ανταλλαγή χρήσιμων πληροφοριών – βέλτιστων μονοπατιών μεταξύ των πληθυσμών της κάθε νήσου. Κατά την διαδικασία της μετανάστευσης επιλέγετε ποσοστό του συνολικού πληθυσμού μιας νήσου και μεταναστεύει σε μια άλλη νήσο με σκοπό να διασταυρωθεί με τον συγκεκριμένο πληθυσμό της.

Η μετανάστευση πραγματοποιείται κυκλικά (Νησί 1→Νησί 2, Νησί 2→Νησί 3 **…** Νησί *isl*→ Νησί 1) και ταυτόχρονα σε όλα τα νησιά ώστε να αποφευχθεί ο συνωστισμός πληθυσμού λύσεων σε μία νήσο. Ο συνολικός αριθμός μεταναστεύσεων *(imm)* καταχωρείται από τον χρήστη και η εκτέλεση τους κατανέμεται σε όλο το μήκος των γενεών ισόποσα. Η κάθε μετανάστευση, δηλαδή, θα συμβαίνει κάθε *Gen/imm* αριθμό γενεών.[1]

Εικόνα 5: Σχεδιάγραμμα Μετανάστευσης



4.3.3 ΜΑ

Μεταξύ δύο διαδοχικών μεταναστεύσεων σε κάθε νησί πραγματοποιούνται τα βήματα του μιμητικού αλγορίθμου όπως αναφέρονται στον πίνακα της *εικόνας 2*. Έχοντας εκχωρηθεί τα δεδομένα του προβλήματος (*Συντεταγμένες των κόμβων και της αποθήκης, d, Cap, Tmax, Gen, S, isl, imm)* και υπολογισθεί και αναπτυχθεί οι πίνακες *A, Pop*. Εφόσον ο πίνακας *Pop* έχει χωριστεί σε νήσους Pop_{isl} , στόχος του ΜΑ είναι η όσο το δυνατών μεγαλύτερη μείωση της αντικειμενικής συνάρτησης f στο τέλος της διαδικασίας του αλγορίθμου ακολουθώντας τα επόμενα βήματα

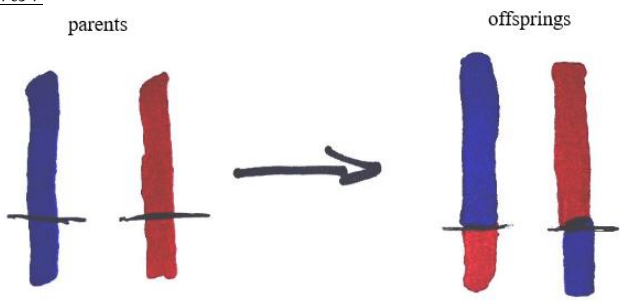
- **Επιλογή γονέων**

Ο συνολικός πληθυσμός μιας νήσου χωρίζεται σε ζευγάρια χρωμοσωμάτων με τυχαία επιλογή. Το κάθε ζευγάρι θα συντελέσει στην δημιουργία δύο απογόνων χρωμοσωμάτων και γι' αυτό τον λόγο ονομάζονται γονείς. Εάν ο αριθμός των χρωμοσωμάτων σε έναν πληθυσμό είναι μονός, ένα χρωμόσωμα δεν θα συμπεριληφθεί στην δημιουργία απογόνων

- **Δημιουργία απογόνων**

Σε κάθε ζευγάρι γονέων επιλέγεται ένα τυχαίο σημείο στο οποίο τα δύο χρωμόσωμα θα διχοτομηθούν και έπειτα θα ανταλλάξουν γενετικό υλικό με τον εξής τρόπο:

Εικόνα 6: Δημιουργία απογόνων



Το πρώτο τμήμα του ενός γονέα συνδέεται στην αρχή του δεύτερου τμήματος του άλλου γονέα, ενώ το δεύτερο τμήμα του ίδιου γονέα συνδέεται στο τέλος του πρώτου τμήματος του άλλου.

Οι απόγονοι που παράγονται, δηλαδή, είναι ίσοι σε αριθμό με τους γονείς που διασταυρώθηκαν. Όντας ήδη τα χρωμοσώματα των γονέων αποθηκευμένα στον πίνακα Pop_{isl} ,οι απόγονοι όλων των ζευγαριών αποθηκεύονται σε έναν πίνακα προσωρινής χρήσης *off,* ο οποίος χρησιμοποιείται μόνο για τους απογόνους που δημιουργούνται σε κάθε γενιά.

Παράδειγμα 3.1:

$$parent1 = (\dots 34, 12, 3, \downarrow^{\theta\acute{\epsilon}\sigma\eta\ i}\ 7, 21, 31 \dots)$$

$$parent2 = (\dots 5, 32, 17, \downarrow^{\theta\acute{\epsilon}\sigma\eta\ i}\ 25, 4, 33\ \dots)$$

$$\boldsymbol{offspring1} = (\dots \mathbf{34, 12, 3}, \downarrow^{\boldsymbol{\theta\acute{\epsilon}\sigma\eta\ i}}\ \mathbf{25, 4, 33} \dots)$$

$$\boldsymbol{offspring2} = (\dots \mathbf{5, 32, 17}, \downarrow^{\boldsymbol{\theta\acute{\epsilon}\sigma\eta\ i}}\ \mathbf{7, 21, 31}\ \dots)$$

Σημείωση: Με αυτή τη μέθοδο διασταύρωσης παρατηρείται το εξής πρόβλημα: Με την ανταλλαγή κόμβων μεταξύ των δύο χρωμοσωμάτων, το πιο πιθανό είναι να προκύψουν δύο λύσεις όπου στην κάθε μια δεν θα πληρούνται τα κριτήρια εγκυρότητας της, δηλαδή κάποιοι κόμβοι να μην παρουσιάζονται στο διάνυσμα λύσης του απογόνου και κάποιοι άλλοι να υπερκαλύπτονται καθώς υποδεικνύεται διπλή επίσκεψη οχήματος από το ίδιο διάνυσμα. Για την συνέχεια του αλγορίθμου είναι αναγκαίο, μετά από κάθε διασταύρωση οι απόγονοι να εξετάζονται για την παράληψη και την υπερκάλυψη κόμβων και να αντικαθιστούν τους διπλότυπους κόμβους με αυτούς που λείπουν.

Παράδειγμα 3.2:

$$parent1 = (1, 3, 2, 5, \downarrow 8, 4, 6, 7)$$

$$parent2 = (1, 4, 6, 3, \downarrow 7, 2, 8, 5)$$

$$\boldsymbol{offspring1} = (1, 3, 2, 5, 7, 2, 8, 5)$$

$$\boldsymbol{offspring2} = (1, 4, 6, 3, 8, 4, 6, 7)$$

Στο παραπάνω παράδειγμα με σύνολο οκτώ κόμβων παρατηρείται ότι στον $1^{ο}$ απόγονο λείπουν οι κόμβοι { 4 , 6 } ενώ στον $2^{ο}$ απόγονο λείπουν οι κόμβοι { 2 , 5 }. Με την αντικατάσταση των διπλότυπων κόμβων με τους απόντες οι απόγονοι διορθώνονται και γίνονται αποδεκτές λύσεις:

$$\boldsymbol{offspring1} = (1, 3, 2, 5, 7, 4, 8, 6)$$

$$\boldsymbol{offspring2} = (1, 4, 6, 3, 8, 2, 5, 7)$$

- **Εφαρμογή Τοπικής αναζήτησης/Μεταλλάξεων στον πληθυσμό**

Έπειτα σε κάθε χρωμόσωμα απόγονο του πληθυσμού πραγματοποιούνται :

1. **Μικρός αριθμός γενεών Local Search 3-Opt** σε μέρος του πληθυσμού της νήσου. Όπως αναπτύχθηκε η χρήση του στο κεφάλαιο 3.3, για την περαιτέρω βελτιστοποίηση μιας λύσης. Στην συγκεκριμένη εργασία επιλέγεται να επιβραβευτεί η μερίδα των λύσεων με τα καλύτερα αποτελέσματα της γενιάς και να εφαρμοστεί μόνο σε αυτές ο 3-Opt.
2. **Μεταλλάξεις**: Οι μεταλλάξεις , οι οποίες πραγματοποιούνται με πιθανότητα p στον πληθυσμό μιας νήσου, είναι τυχαίες αλλαγές στην συνοχή μιας λύσης. Αυτές αποσκοπούν, όπως και ο 3-Opt, στην βελτιστοποίηση του αποτελέσματος, ωστόσο η διαδικασία των μεταλλάξεων δημιουργεί ταυτόχρονα ποικιλότητα των λύσεων καθώς δημιουργεί ανωμαλίες και αποτρέπει την πρόωρη σύγκλιση του αλγορίθμου. Στην εκτέλεση του αλγορίθμου για την επίλυση του OVRP οι μεταλλάξεις συμβαίνουν με πιθανότητα 50% στο κάθε χρωμόσωμα με τον εξής τρόπο :

Κατά την διαδικασία της μετάλλαξης εάν αύτη συμβεί τυχαία επιλέγεται το $Pop_{isl} \times 10\%$ (το ακέραιο μέρος, για προβλήματα μεγαλύτερα των 20 κόμβων) των συνολικών κόμβων και ύστερα ανταλλάσσουν θέση κυκλικά. Σε κάθε ζευγάρι, οι κόμβοι που το απαρτίζουν, ανταλλάσσουν θέσεις, δηλαδή:

Παράδειγμα 4: N=30 Mutation (3 nodes)

$$off = (1,9,2,15,\mathbf{22},3,21,6,7,18,\mathbf{13},28,4,19,17,11,23,27,24,30,20,14,25,\mathbf{5},8,26,12,29,10,16)$$

$$\boldsymbol{mut\_off} = (1,9,2,15,\mathbf{5},3,21,6,7,18,\mathbf{22},28,4,19,17,11,23,27,24,30,20,14,25,\mathbf{13},8,26,12,29,10,16)$$

Τα μεταλλαγμένα χρωμοσώματα αποθηκεύονται σε έναν προσωρινό πίνακα, *mut,* ο οποίος χρησιμοποιείται μόνο για της μεταλλάξεις των απογόνων που δημιουργούνται σε κάθε γενιά

- **Ενημέρωση Πληθυσμού**

Στο τέλος μιας γενιάς απαιτείται η προετοιμασία του πληθυσμού της νήσου για την έναρξη της επόμενης. Έτσι οι πίνακες των γονέων Pop_{isl}, των απογόνων *off* και των μεταλλαγμένων απογόνων *mut* συγχωνεύονται σε έναν ενιαίο πίνακα.

Παράδειγμα 5.1: Συγχώνευση πινάκων

$$\begin{bmatrix} \boldsymbol{Pop_{isl\,1}} \\ \boldsymbol{Pop_{isl\,2}} \\ \vdots \\ \boldsymbol{off_1} \\ \boldsymbol{off_2} \\ \vdots \\ \boldsymbol{mut_1} \\ \boldsymbol{mut_2} \\ \vdots \end{bmatrix}$$

$Pop_{isl\,i}$: i-οστό χρωμόσωμα του πίνακα Pop_{isl}

off_i: i-οστό χρωμόσωμα του πίνακα off

mut_i: i-οστό χρωμόσωμα του πίνακα mut

Έπειτα πραγματοποιείται αξιολόγηση όλων των χρωμοσωμάτων και κατάταξη τους από το καλύτερο χρωμόσωμα στο χειρότερο. Αυτός ο ενιαίος πίνακας μετά την συγχώνευση θα έχει μέγεθος $[(\,Pop_{isl} + off + mut\,)\ \sigma\epsilon\iota\rho\acute{\epsilon}\varsigma\ ,(N)\sigma\tau\acute{\eta}\lambda\epsilon\varsigma]$.

Παράδειγμα 5.2: Πίνακας ύστερα από αξιολόγηση

$$\begin{bmatrix} \boldsymbol{Best}\colon \boldsymbol{off_{68}} \\ \boldsymbol{Pop_{isl\,25}} \\ \boldsymbol{mut_7} \\ \boldsymbol{off_{34}} \\ \boldsymbol{Pop_{isl\,89}} \\ \vdots \\ \vdots \\ \vdots \\ \boldsymbol{Worst}\colon \boldsymbol{off_{56}} \end{bmatrix}$$

Στον συνολικό πίνακα μετά την αξιολόγηση θα διατηρηθούν μόνο οι πρώτες Pop_{isl} ώστε ο πληθυσμός στο τέλος της γενιάς να παραμείνει ίδιος, σε σχέση με την αρχή της, δηλαδή το μέγεθος του θα καταλήξει πάλι σε διαστάσεις $[(\,Pop_{isl})\ \sigma\epsilon\iota\rho\acute{\epsilon}\varsigma\ ,(N)\sigma\tau\acute{\eta}\lambda\epsilon\varsigma]$.

Παράδειγμα 5.3: Πίνακας μετά την ενημέρωση πληθυσμού

$$\begin{bmatrix} \boldsymbol{Best}: \boldsymbol{off_{68}} \\ \boldsymbol{Pop_{isl\,25}} \\ \boldsymbol{mut_7} \\ \boldsymbol{off_{34}} \\ \boldsymbol{Pop_{isl\,89}} \\ \vdots \\ \boldsymbol{mut_{20}} \\ ------ \\ \vdots \\ \boldsymbol{Worst}: \boldsymbol{off_{56}} \end{bmatrix} \rightarrow \begin{bmatrix} \boldsymbol{Best}: \boldsymbol{off_{68}} \\ \boldsymbol{Pop_{isl\,25}} \\ \boldsymbol{mut_7} \\ \boldsymbol{off_{34}} \\ \boldsymbol{Pop_{isl\,89}} \\ \vdots \\ \boldsymbol{mut_{20}} \end{bmatrix}$$

$------$: Τομή του πίνακα μετά την Pop_{isl} -οστή σειρά

4.3.4 Σύγκλιση λύσης

Η δημιουργία μιας ή πολλαπλών λύσεων, οι οποίες έχουν καλύτερη επίδοση στην αντικειμενική συνάρτηση εν συγκρίσει των άλλων , μέσα σε έναν πληθυσμό, οδηγεί στην επιβίωση, την αναπαραγωγή και την διαιώνιση των χαρακτηριστικών τους από γενιά σε γενιά. Αυτή η συμπεριφορά της ανεξέλεγκτης αναπαραγωγής και επιβίωσης αποτελεί προβληματική, καθώς οδηγεί στην δημιουργία ενός πληθυσμού με ίδια χαρακτηριστικά και συνοχή ο οποίος στις εναπομείναντες γενιές παραμένει ίδιος και δεν εξελίσσεται. Έτσι θεωρείται ότι ο αλγόριθμος συγκλίνει πρόωρα σε μια λύση χωρίς να του δίνετε το περιθώριο βελτίωσης.

Παράδειγμα 6:

Έστω η δημιουργία μιας λύσης η οποία η οποία σε σχέση με τον υπόλοιπο πληθυσμό είναι αρκετά πιο ανταγωνιστική. Αυτή στην επόμενη γενιά πέρα της εξασφάλισης της επιβίωσης της, θα αναπαραχθεί και, θα δημιουργήσει απογόνους οι οποίοι θα φέρουν όμοια στοιχεία με τον γονέα τους, συνεπώς εξασφαλίζεται και η επιβίωση αυτών. Έχοντας κατοχυρώσει ήδη 3 θέσεις στον πληθυσμό της επομένης γενιάς, η ξανά επιβίωση τους και αναπαραγωγή τους έχουν υψηλή πιθανότητα να ξανασυμβούν στην επόμενη γενιά κατοχυρώνοντας περισσότερες θέσεις στον πληθυσμό.

Αυτό που πρέπει να αποφευχθεί είναι η διαγραφή χρήσιμων πληροφοριών-μονοπατιών που υπάρχουν σε λύσεις με όχι και τόσο καλή επίδοση ή ,και η μη δημιουργία νέων πληθυσμών που θα φέρουν καινούργιες πληροφορίες και μπορούν να οδηγήσουν στη περεταίρω βελτιστοποίηση του αποτελέσματος

Αυτό το πρόβλημα στην προκειμένη εργασία με τους εξής τρεις τρόπους:

- **Μεταλλάξεις**: Όπως αναπτύχθηκε η χρήση τους στο κεφάλαιο 4.3.3
- **Εντοπισμός σύγκλισης**: Ανά δέκα γενιές, πραγματοποιείται έλεγχος σύγκλισης στη κάθε νήσο. Με τον έλεγχο σύγκλισης εννοείται ο εντοπισμός ομοιότητας μεταξύ των χρωμοσωμάτων ενός πληθυσμού. Αυτή γίνεται με την εξής διαδικασία:

  Σε κάθε πίνακα Pop_{isl} ελέγχεται η κάθε στήλη ξεχωριστά, εξετάζοντας κάθε φορά την ομοιότητα των κόμβων στην συγκεκριμένη θέση του διανύσματος όλων των λύσεων του πληθυσμού. Ξεκινώντας από την **δεύτερη θέση** του πρώτου χρωμοσώματος (καθώς η πρώτη θέση όλων των χρωμοσωμάτων φέρουν τον κόμβο '1'- αποθήκη) ελέγχεται πόσα χρωμοσώματα φέρουν τον ίδιο κόμβο στην ίδια θέση. Συνεχίζοντας το δεύτερο χρωμόσωμα στην ίδια θέση, καταμετρούνται τα όμοια χρωμοσώματα αυτού και ούτω κάθε εξής και στα υπόλοιπα χρωμοσώματα του πληθυσμού. Αφού ελεγχθεί η ομοιότητα των χρωμοσωμάτων στη δεύτερη θέση αλγοριθμικά εξετάζεται ομοίως η τρίτη και έπειτα οι υπόλοιπες μέχρι την N-οστή θέση των χρωμοσωμάτων. Στο τέλος του ελέγχου γίνεται καταμέτρηση του αριθμού των ζευγαριών των κόμβων που υπήρχε ομοιότητα ο οποίος διαιρείται με το συνολικό αριθμό των ζευγαριών που εξεταστήκαν ώστε να προκύψει ένα ποσοστό ομοιότητας. Εάν αυτό το ποσοστό δεν υπερβαίνει το όριο του 50% τότε θεωρείται ότι η ομοιότητα δεν είναι αρκετή και η διαδικασία του IMA μπορεί να συνεχίσει εάν ωστόσο το ποσοστό υπερβαίνει το όριο του 50%, πραγματοποιείται 3-Opt τοπική αναζήτηση σε όλο τον πληθυσμό του πίνακα με σκοπό την ανακατανομή των κόμβων και ταυτόχρονα την αναζήτηση καλύτερων λύσεων.
- **Ποινή χειρότερης επίδοσης:** Για την ποινή χειρότερης επίδοση αναπτύχθηκε μια συνάρτηση κώδικα η οποία χρησιμοποιείται κάθε 50 γενιές μιμητικού αλγόριθμου και έχει ως σκοπό την τιμωρία του λιγότερο εξελιγμένου, δηλαδή των χρωμοσωμάτων που θα έχουν συγκλίνει λιγότερο προς την καλύτερη μέχρι στιγμής λύση. Κάθε 50 γενιές, λοιπόν, το νησί με την χειρότερη επίδοση στον πληθυσμό του, επιλέγεται να σταματήσει να λαμβάνεται υπόψιν για τις επόμενες γενιές. Σταματάει να συμμετέχει στην μετανάστευση και όλες τις άλλες διαδικασίες του IMA. Αντίθετα δημιουργείται ένα πλήρως καινούργιο νησί, με την μέθοδο GRASP και παίρνει αυτό την θέση του προηγούμενου. Αυτό θα συνεχίσει να εξελίσσεται στις επόμενες γενιές και να συμμετέχει στις μεταναστεύσεις με τα υπόλοιπα νησιά. Αυτός ο τρόπος προσφέρει την ποικιλότητα των λύσεων καθώς το συγκεκριμένο νησί αρχίζει μέσα στις γενιές να προσεγγίζει και να συγκλίνει σε μια πλήρως διαφορετική λύση η οποία μεταδίδεται και στις υπόλοιπες νήσους μέσω της μετανάστευσης. Όπου μετά από κάθε γενιά επιβιώνουν τα πιο «ισχυρά» μονοπάτια της κάθε λύσης αντίστοιχα. Ενώ μόλις αρχίσουν οι λύσεις να συγκλίνουν πάλι σε μία η διαδικασία της ποινής και δημιουργίας καινούργιας νήσου θα επαναληφθεί.

5. Παρουσίαση Κώδικα

Ο κώδικα που θα παρουσιαστεί αναπτύχθηκε σε γλώσσα προγραμματισμού Python, στο περιβάλλον της Pycharm, και διεκπεραίωσε την διαδικασία του IMA και του 3opt αλγορίθμου τοπικής αναζήτησης ώστε να παράγει το βελτιστοποιημένο αποτέλεσμα ενός μονοπατιού το οποίο θα λύνει το πρόβλημα OVRP του οποίου τα δεδομένα είναι οι συντεταγμένες και η ζήτηση των κόμβων, καθώς και τα χαρακτηριστικά των οχημάτων (χωρητικότητα, χρόνος εξυπηρέτησης, όριο μέγιστης διαδρομής).

MAIN.PY	
1	*// Prompt the user for a file input*
2	***Display*** *"Enter the file:"*
3	*file = Get user-selected file*
4	
5	*// Call a function to process the file and extract necessary data*
6	*nodes_num, vcap, maxroute, servtime, coordinates, demand = createv(file)*
7	
8	*// Create the distance matrix from coordinates*
9	*distance = make_distance_matrix(coordinates)*
10	
11	*// Prompt the user for the number of islands*
12	*Display "Enter Number of Islands:"*
13	*isl_num = Get user input (as integer)*
14	
15	*// Prompt the user for the number of chromosomes in the population*
16	*Display "Enter Number of Chromosomes in the population:"*
17	*chrom_num = Get user input (as integer)*
18	
19	*// Create the first population and divide into islands*
20	*first_parents = parent_making(nodes_num, chrom_num, demand, distance, vcap, maxroute, servtime)*
21	*islands = make_islands(first_parents, chrom_num, isl_num)*
22	
23	*// Optimization via fitness function*
24	*Survival_of_the_fittest = fitness(islands, nodes_num, demand, vcap,*

	maxroute, servtime, distance, isl_num)
25	*Display survival_of_the_fittest*
26	
27	*// Initialize variables to track the best cost and path*
28	*Best_Cost = infinity*
29	*Best_Path = None*
30	
31	*// Find the best path and cost from the population*
32	*For each island in Survival_of_the_fittest:*
33	*Cost_of_the_Best =cost(Survival_of_the_fittest[island], demand, distance, nodes_num, vcap, maxroute, servtime)*
34	*cost_indices = Sort indices of Cost_of_the_Best in ascending order*
35	*island_Best_Cost = Get minimum cost from sorted indices*
36	*island_Best_Path = Get corresponding path for minimum cost*
37	
38	*// Update the best path and cost if the current island is better*
39	*If island_Best_Cost < Best_Cost:*
40	*Best_Cost = island_Best_Cost*
41	*Best_Path = island_Best_Path*
42	
43	*// Output the best path and cost*
44	*Display Best Path*
45	*Display Best Cost*
46	
47	*// Execute restock on the best path and visualize the result*
48	*restock1 = restock(Best_Path, nodes_num, demand, distance)*
49	*visualize_path(restock1, coordinates)*
50	
51	*// Prompt the user for the number of local search iterations*
52	*Display "Enter Number of Local Search Iterations:"*
53	*ls_iteration = Get user input (as integer)*
54	*// Execute the 3-opt function*
55	*opt_path = 3-opt(Best_Path, nodes_num, demand, distance, ls_iteration)*
56	*cost_opt_path = cost.cost_tsp(opt_path, demand, distance, nodes_num)*
56	
57	*// Execute the restock function on the optimized path and display results restock = restock(opt_path, nodes_num, demand, distance)*
58	*Display restock*

59	*Display cost_opt_path*
60	
61	*// Visualize the restock path*
62	*visualize_path(restock, coordinates)*

Από τις παραπάνω γραμμές κώδικα της main συνάρτησης παρατηρείται η κατοχύρωση των μεταβλητών μέσω ενός αρχείου που εισάγεται από τον χρήστη και έπειτα μέσω της επεξεργασίας αυτού από την συνάρτηση *createv().* Στην *main* καλούνται οι πιο σημαντικές συναρτήσεις του αλγορίθμου, δηλαδή η δημιουργία του πίνακα τον αποστάσεων,$\boldsymbol{A_{ij}}$, *make_distance_matrix()* η δημιουργία του πρώτου πληθυσμού (*parent_making()*) και έπειτα ο διαχωρισμό του σε νήσους *island_making()*, η *fitness()* η οποία είναι υπεύθυνη για την εξέλιξη του πληθυσμού και η *cost()* που υπολογίζει την απόδοση μιας πιθανής λύσης- χρωμοσώματος και η *3-Opt(),*η οποία εμπεριέχει την διαδικασία του αλγορίθμου τοπικής αναζήτησης. Καλούνται επίσης η *restock ()* η οποία αναλαμβάνει την επεξεργασία μιας λύσεις ώστε να γίνεται αντιληπτό στον χρήστη πότε ένα όχημα επιστρέφει στην αποθήκη, καθώς όπως αναφέρεται στην σημείωση του κεφαλαίου 4.3.1 η επίσκεψη αυτή παραλείπετε από μία λύση, και η *visualize_path()* η οποία οπτικοποιεί το καλύτερο μονοπάτι-λύση στο τέλος της διαδικασίας. Παρακάτω παρουσιάζονται όλες οι συναρτήσεις που αναπτύχθηκαν για την επίλυση του προβλήματος.

5.1 Εισαγωγή και επεξεργασία δεδομένων

Η παρακάτω συνάρτηση είναι υπεύθυνη για την σωστή καταχώρηση των πληροφοριών του Database των κόμβων. Για να λειτουργήσει σωστά το πρόγραμμα είναι απαραίτητη η συγκεκριμένη δομή του αρχείου που εισάγεται :

Στη πρώτη γραμμή να εμφανίζεται ο συνολικός αριθμός των κόμβων(καταχωρείται ως nodesnum),

Στην τρίτη γραμμή, η χωρητικότητα του αμαξιού (vcap),

Στην τέταρτη γραμμή, το όριο μέγιστης διαδρομής (maxroute),

Στην πέμπτη γραμμή, ο χρόνος εξυπηρέτησης του κάθε κόμβου (servtime),

Έπειτα, σε κάθε γραμμή παρουσιάζονται οι συντεταγμένες του κάθε κόμβου

Και ύστερα, στις τελευταίες γραμμές, η ζήτηση του κάθε κόμβου.

CREATEV(FILE)	
1	*Try:*
2	*Open file for reading*
3	*Read all lines into a list `lines`*
4	*If file is not found:*
5	*Print "File not found."*
6	*Return None, None, None, None, None, None*
7	
8	*Initialize empty list `nodesnum`*
9	*Initialize variable `B` as None*
10	*Initialize empty list `vcap`*
11	*Initialize empty list `maxroute`*
12	*Initialize empty list `servtime`*
13	*Initialize empty list `coordinates`*
14	*Initialize empty list `demand`*
15	
16	*For each i from 0 to 4:*
17	*Convert `lines[i]` to integer and store in `number`*
18	*If i is 0:*
19	*Append `number` to `nodesnum`*
20	*Else if i is 1:*
21	*Set `B = 0` (not used)*
22	*Else if i is 2:*
23	*Append `number` to `vcap`*
24	*Else if i is 3:*
25	*Append `number` to `maxroute`*
26	*Else if i is 4:*
27	*Append `number` to `servtime` Set `nodesnum` to first value of `nodesnum`*
28	
29	*For each line from `lines[5]` to `lines[nodesnum + 5]`:*
30	*Split the line into `elements`*
31	*Append [elements[1] as integer, elements[2] as integer] to `coordinates`*
32	
33	*For each line from `lines[nodesnum + 5]` to `lines[2 * nodesnum + 5]`:*
34	*Split the line into `elements`*

35	*Append elements[1] as integer to `demand`*
36	*Return `nodesnum`, `vcap`, `maxroute`, `servtime`, `coordinates`, `demand`*

Η συνάρτηση Make_Distance_Matrix() καλείται από την Main ώστε να υπολογίσει τις αποστάσεις των κόμβων μεταξύ τους, όπως αναφέρεται στο κεφάλαιο 4.1 και να τις αποθηκεύσει στον πίνακα 'distance_matrix' ώστε να χρησιμοποιηθούν στον υπολογισμό της επίδοσης των χρωμοσωμάτων.

MAKE_DISTANCE_MATRIX(COORDINATES)	
1	*Create a 2D array `distance_matrix` with dimensions [len(coordinates)] x [len(coordinates)], filled with zeros*
2	
3	*For each index `i` from 0 to len(coordinates) - 1:*
4	*For each index `j` from 0 to len(coordinates) - 1:*
6	*Calculate the distance between `coordinates[i]` and `coordinates[j]`*
7	*Store the result in `distance_matrix[i][j]`*
8	
9	*Return `distance_matrix`*

5.2 IMA-Κώδικας

Παρακάτω παρουσιάζεται η διαδικασία που ακολουθεί ο αλγόριθμος IMA για την επίλυση του OVRP όπως αυτή αναπτύχθηκε στα κεφάλαια 4.2 και 4.3 , σε μορφή ψευδοκώδικα.

5.2.1 Δημιουργία Αρχικού Πληθυσμού και Νήσων

Η Parent_Making() συνάρτηση καλείται από την Main () και είναι υπεύθυνη για την δημιουργία του αρχικού πληθυσμού όπου τα μισά χρωμοσώματα δημιουργούνται τυχαία ενώ τα υπόλοιπα χρησιμοποιώντας την μέθοδο GRASP.

PARENT_MAKING(NODES_NUM, CHROM_NUM, DEMAND, DISTANCE, VCAP, MAXROUTE, SERVTIME))	
1	// Create a list of node indices excluding the first node (assuming nodes are indexed from 1)

2	*node_list = [2, 3, …, nodes_num]*
3	
4	*// Calculate half of the number of chromosomes*
5	*half_chrom_num = chrom_num // 2*
6	
7	*// Initialize two populations*
8	*random_population = ARRAY of size (half_chrom_num, nodes_num) with all values set to 0*
9	*grasp_population = ARRAY of size (half_chrom_num, nodes_num) with all values set to 0*
10	
11	*// Set the first node (index 0) to be fixed in all chromosomes*
12	*FOR each chromosome in random_population*
13	*SET first element (index 0) of chromosome to 1*
14	
15	*// Generate half of the population randomly*
16	*FOR i FROM 0 TO half_chrom_num - 1*
17	*// Randomly assign node indices (excluding the first node) to the chromosome*
18	*random_population[i][1:nodes_num] = RANDOM sample of node_list with size (nodes_num - 1)*
19	
20	*//Generate the other half of the population using GRASP*
21	*FOR i FROM 0 TO half_chrom_num - 1*
22	*//Assign a GRASP-generated chromosome to the population*
23	*grasp_population[i] = CALL Grasp function with parameters (node_list, nodes_num, Demand, Distance, vcap, maxroute, servtime)*
24	
25	*//Concatenate both populations*
26	*new_population = CONCATENATE random_population and grasp_population*
27	*// Shuffle the concatenated population*
28	*SHUFFLE new_population randomly*
29	*RETURN new_population*

Η Island_Making() επίσης καλείται από την Main και λαμβάνοντας σαν όρισμα τον πίνακα του πρώτου πληθυσμού, σκοπός της είναι ο διαχωρισμός του πληθυσμού σε νήσους. Οι πίνακες των νήσων αποθηκεύονται όλοι σε ένα 'dictionary' της Python με κωδικό- κλειδί το καθένα I_i (όπου i = 0,1 … isl)

ISLAND_MAKING(PARENTS, CHROM_NUM, ISL_NUM)	
1	*// Calculate the number of chromosomes per island*
2	*chrom_per_island = chrom_num // isl_num*
3	
4	*// Initialize an empty dictionary to store islands*
5	*islands = EMPTY DICTIONARY*
6	
7	*// Split the parents into islands*
8	*FOR i FROM 0 TO isl_num - 1*
9	*// Calculate the starting and ending indices for this island*
10	*start_idx = i * chrom_per_island*
11	*end_idx = (i + 1) * chrom_per_island*
12	*// Slice the parents array for this island and store it in the dictionary*
13	*islands['I' + (i + 1)] = SLICE parents from start_idx to end_idx*
14	*// Return the dictionary of islands*
15	*RETURN islands*

Η Grasp() είναι η συνάρτηση στην καλούνται όλες οι εντολές για την πραγματοποίηση της μεθόδου GRASP όπως αναπτύχθηκε στο κεφάλαιο 4.3.1. Αυτή καλείται από την συνάρτηση Parent_Making για την δημιουργία του αρχικού πληθυσμού καθώς και από τη Replace_worst_island() για την δημιουργία καινούργιου νησιού έπειτα από την ποινή και αντικατάσταση της χειρότερης νήσου κάθε 50στή γενιά. Κατά την διαδικασία της Grasp καλείται επίσης ο αλγόριθμος 3-Opt μέσω της συνάρτησης Threeopt_Grasp() για την βελτιστοποίηση του αρχικού πληθυσμού που μόλις δημιουργήθηκε.

GRASP(NODE_LIST, NODES_NUM, DEMAND, DISTANCE, VCAP, MAXROUTE, SERVTIME)	
1	*new_list = [1]*
2	*D = 0*
3	*M = 0*
4	
5	*// CREATE PATH*

```
6   WHILE LENGTH of new_list < nodes_num:
7       current_city = LAST element in new_list
8       unvisited_cities = [node FOR node IN node_list IF node NOT in
        new_list]
9       costs = []
10
11      // COST APPEND
12      FOR city IN unvisited_cities:
13          IF D + Demand[city-1] <= vcap AND M + Distance[current_city-
            1][city-1] + servtime <= maxroute:
14              cost = Distance[current_city-1][city-1]
15          ELSE:
16              cost = Distance[0][city-1]
17          APPEND cost TO costs
18
19      // PROBABILITY OF SELECTION
20      probabilities = [1 / cost FOR cost IN costs]
21      probabilities = probabilities / SUM(probabilities)
22
23      // SELECT CITY BASED ON PROBABILITY
24      selected_city = RANDOM CHOICE from unvisited_cities based on
        probabilities
25
26      // APPEND SELECTED CITY TO THE PATH AND UPDATE DEMAND
        AND DISTANCE
27      APPEND selected_city TO new_list
28      IF D + Demand[selected_city-1] <= vcap AND M + Dis-
        tance[current_city-1][selected_city-1] + servtime <= maxroute:
29          D += Demand[selected_city-1]
30          M += Distance[current_city-1][selected_city-1] + servtime
31      ELSE:
32          D = Demand[selected_city-1]
33          M = Distance[0][selected_city-1] + servtime
34
35  RETURN new_list
```

THREEOPT_GRASP(PATH,NODES_NUM,DEMAND,DISTANCE)

```
1   num_generation = 500
```

```
2   FOR i FROM 1 TO num_generation
3       a = RANDOM INTEGER BETWEEN 1 AND (nodes_num - 1)
4       b = RANDOM INTEGER BETWEEN (a + 1) AND nodes_num
5       c = RANDOM INTEGER BETWEEN 0 AND 4
6       vec1 = SUBARRAY OF Path FROM 0 TO a
7       vec2 = SUBARRAY OF Path FROM a TO b
8       vec3 = SUBARRAY OF Path FROM b TO END
9
10      candidates = [
11          CONCATENATE([1], vec1, vec3, vec2)
12          CONCATENATE([1], vec2, vec3, vec1)
13          CONCATENATE([1], vec2, vec1, vec3)
14          CONCATENATE([1], vec3, vec1, vec2),
15          CONCATENATE([1], vec3, vec2, vec1),
16      ]
17
18      costs = [CALL cost_tsp(candidate, Demand, Distance, nodes_num) FOR
        EACH candidate IN candidates]
19
20
21      min_index = FIND INDEX OF MINIMUM VALUE IN costs
22
23      Path = candidates[min_index]
24
25  RETURN Path
```

5.2.2 Επίδοση (Fitness) και Εξέλιξη

Στην συνάρτηση fitness πραγματοποιούνται τα κύρια βήματα του μιμητικού αλγορίθμου όπως αναφέρονται στο κεφάλαιο 3.2. Αρχικά καταχωρούνται από τον χρήστη ο αριθμός τον γενεών που θα πραγματοποιηθούν, ο αριθμός των επαναλήψεων που θα συμβεί ο 3-Opt ενδιάμεσα των γενεών καθώς και ο αριθμός των μεταναστεύσεων που θα συμβούν.

FITNESS(FIRST_PARENTS, NODES_NUM, DEMAND, VCAP, MAXROUTE, SERVTIME, DISTANCE, ISL_NUM)

```
1   // Input the number of generations
2   PRINT "Enter Number of Generations:"
3   num_generations = INPUT as integer
```

4	
5	// Input the number of local search iterations\
6	PRINT "Enter Number of Local Search Iterations:"
7	ls_iterations = INPUT as integer
8	
9	// Input the number of immigrations
10	PRINT "Enter Number of Immigrations to happen:"
11	imm = INPUT as integer
12	// Calculate the step size for immigration
13	step = num_generations // (imm + 1)
14	
15	// Determine when to call the immigration process
16	call_generations = ARRAY of values from step to (imm * step) with step size
17	
18	
19	
20	
21	// Initialize the new population with the first parents
22	new_population = first_parents
23	// Iterate over generations
24	FOR generation FROM 0 TO num_generations:
25	// Check if immigration needs to happen
26	IF generation is in call_generations:
27	PRINT "Generation:", generation
28	PRINT "New population:", new_population
29	CALL immigration(new_population, isl_num)
30	
31	// Replace the worst island every 50 generations
32	IF generation % 50 == 0:
33	new_population = CALL replace_worst_island(new_population, nodes_num, Demand, Distance, vcap, maxroute, servtime)
34	
35	// Perform local search and other operations on the islands

```
36      IF generation <= num_generations:
37          // Perform local search on the 2 best chromosomes
38          new_population = CALL lsinbetween(new_population, Demand,
    Distance, nodes_num, vcap, maxroute, servtime, isl_num, ls_iterations)
39
40          // Iterate over each island
41          FOR i FROM 0 TO isl_num - 1:
42              island = 'I' + (i + 1)
43              isl_chrom_num = LENGTH of new_population[island]
44              // Perform similarity check every 5 generations
45              IF generation % 10 == 0:
46                  new_population[island] = CALL similarity_check
    (new_population[island], nodes_num, Demand, Distance)
47
48              // CROSSOVER process
49              offspring_crossover = CALL
    ma.create_offspring_matrix(new_population[island], isl_chrom_num)
50
51              // Fix duplicates in the offspring
52              offspring_crossover = CALL fix_duplicates_in_matrix (off-
    spring_crossover)
53
54              // Perform mutation
55              offspring_mutation = CALL ma.mutation(offspring_crossover,
    nodes_num, Demand, Distance, ls_iterations)
56
57              new_population[island] = CONCATENATE offspring_crossover
    and offspring_mutation
58
59              // Calculate the cost of the new population
60              C = CALL cost(new_population[island], Demand, Distance,
    nodes_num, vcap, maxroute, servtime)
61
62              // Sort the population based on fitness (cost)
```

63	*new_population[island] = CALL select_mating(new_population[island], C)*
64	
65	*// Keep only the best chromosomes*
66	*new_population[island] = new_population[island][0 : isl_chrom_num]*
67	*// Return the final population*
68	*RETURN new_population*

Για την βελτιστοποίηση του αποτελέσματος του πληθυσμού καλούνται από τη fitness() οι εξής συναρτήσεις:

- Η Create_Offspring_Matrix() είναι η συνάρτηση που θα δημιουργήσει τον πίνακα των απογόνων. Μέσα σε αυτή καλείται η crossover η οποία αναλαμβάνει την δημιουργία των απογόνων (4.3.3).

CREATE_OFFSPRING_MATRIX(PARENTS_MATRIX, CHROM_NUM)	
1	*offspring_matrix = EMPTY matrix with same shape as parents_matrix*
2	*chromosome_list = LIST(range(LENGTH of parents_matrix))*
3	*FOR i FROM 0 TO chrom_num - 2, STEP 2:*
4	*IF LENGTH of chromosome_list < 2:*
5	*CONTINUE*
6	*parent1_idx, parent2_idx = RANDOMLY SELECT 2 indices FROM chromosome_list*
7	*parent1 = parents_matrix[parent1_idx]*
8	*parent2 = parents_matrix[parent2_idx]*
9	
10	*offspring1, offspring2 = PERFORM crossover ON parent1 AND parent2*
11	*offspring_matrix[i] = offspring1*
12	*offspring_matrix[i + 1] = offspring2*
13	*REMOVE parent1_idx AND parent2_idx FROM chromosome_list*
14	*IF chrom_num IS ODD:*
15	*last_parent_idx = chromosome_list[0]*
16	*offspring_matrix[-1] = COPY parents_matrix[last_parent_idx]*
17	*combined_matrix = CONCATENATE parents_matrix AND offspring_matrix ALONG axis 0*
18	

19	*RETURN combined_matrix*
CROSSOVER(PARENT1, PARENT2)	
1	*PERFORM single-point crossover on the two parents:*
2	*crossover_point = RANDOM integer BETWEEN 1 AND LENGTH of parent1 - 1*
3	*offspring1 = CONCATENATE parent1 UP TO crossover_point WITH parent2 FROM crossover_point*
4	*offspring2 = CONCATENATE parent2 UP TO crossover_point WITH parent1 FROM crossover_point*
5	
6	*RETURN offspring1, offspring2*

- Στην συνάρτηση Fix_Duplicates_in_Matrix( ) διορθώνονται οι διπλοτυπίες και οι απώλειες κόμβων από όλα τα χρωμοσώματα του πίνακα των απογόνων

FIX_DUPLICATES_IN_MATRIX(MATRIX)	
1	*num_rows, num_cols = matrix.shape*
2	*fixed_matrix = COPY of matrix*
3	*FOR row_idx FROM 0 TO num_rows - 1:*
4	*row = fixed_matrix[row_idx, :]*
5	*unique_elements, counts = FIND unique elements IN row AND their counts*
6	*duplicates = FIND elements WITH counts > 1*
7	*missing_elements = FIND elements IN range(1, num_cols + 1) NOT IN row*
8	*FOR each duplicate IN duplicates:*
9	*duplicate_indices = FIND indices WHERE row EQUALS duplicate*
10	*FOR idx IN duplicate_indices[1:]: // Keep the first occurrence*
11	*IF LENGTH of missing_elements EQUALS 0:*
12	*BREAK*
13	*row[idx] = missing_elements[0]*
14	*missing_elements = missing_elements[1:]*
15	
16	*RETURN fixed_matrix*

- Η συνάρτηση Mutation() δημιουργεί τον πίνακα mut και δημιουργεί τις μεταλλάξεις του 10% των χρωμοσωμάτων που απαρτίζουν τον πίνακα απογόνων. Στο τέλος της συνάρτησης καλείται ο αλγόριθμος τοπικής αναζήτησης 3-Opt με σκοπό την αύξηση της πιθανότητας η μετάλλαξη να περάσει και στην επόμενη γενιά.

MUTATION(OFFSPRING_CROSSOVER, NODES_NUM, DEMAND, DISTANCE, LS_ITERATIONS)	
1	*pop_size = LENGTH of offspring_crossover*
2	*mutation_size = MAX(1, pop_size // 10) // Calculate 10% of the population size, minimum of 1*
3	*FOR j FROM 0 TO pop_size - 1:*
4	*probability = RANDOM number BETWEEN 0 AND 100*
5	*IF probability < 50:*
6	
7	*// Generate mutation_size unique random indices*
8	*indices = RANDOM SAMPLE of mutation_size elements FROM range(1, nodes_num)*
9	
10	*// Perform swapping among the selected nodes in a cyclic manner*
11	*temp = offspring_crossover[j][indices[0]]*
12	
13	*FOR i FROM 0 TO mutation_size - 2:*
14	*offspring_crossover[j][indices[i]] = off-spring_crossover[j][indices[i + 1]]*
15	*offspring_crossover[j][indices[-1]] = temp*
16	*ELSE:*
17	*CONTINUE*
18	*// Perform local search (e.g., 3-opt)*
19	*offspring_crossover[j] = PERFORM tsp ON offspring_crossover[j] WITH nodes_num, Demand, Distance, ls_iterations*
20	
21	*RETURN offspring_crossover*

- Υπεύθυνη για τον υπολογισμό της απόδοσης του κάθε χρωμοσώματος της γενιάς, η Cost() είναι αυτή που θα καθορίσει την επιβίωση τους. Υπολογίζει σε κάθε χρωμόσωμα την συνολική απόσταση που θα διανύσουν τα οχήματα σύμφωνα με τον τύπο της αντικειμενικής συνάρτησης (4.2.1).

COST(POPULATION, DEMAND, DISTANCE, NODES_NUM, VCAP, MAXROUTE, SERVTIME)	
1	*start = 1*
2	*total_chrom = population.shape[0]*
3	*C = ARRAY of zeros of size total_chrom*
4	*FOR chromosome FROM 0 TO total_chrom - 1:*
5	*d = 0*
6	*m = 0*
7	*// Iterate through nodes in the chromosome*
8	*FOR n FROM 0 TO nodes_num - 2:*
9	*x = population[chromosome, n] - 1*
10	*y = population[chromosome, n + 1] - 1*
11	
12	*// Check if capacity and route constraints are satisfied*
13	*IF d + Demand[y] <= vcap AND m + Distance[x, y] + servtime <= maxroute:*
14	*C[chromosome] += Distance[x, y]*
15	*m += Distance[x, y] + servtime*
16	*d += Demand[y]*
17	*ELSE:*
18	*C[chromosome] += Distance[start, y]*
19	*m = Distance[start, y] + servtime*
20	*d = Demand[y]*
21	*// Return cost for each chromosome*
22	*RETURN C*

- Μετά την αξιολόγηση του συνολικού πληθυσμού η Select_Mating θα κατατάξει όλα τα χρωμοσώματα από το καλύτερο στο χειρότερο ανάλογα με την επίδοση τους (min → max)

SELECT_MATING(NEW_POP, C)	
1	*// Selecting the best individuals in the current generation as parents for producing the offspring of the next generation.*
2	*row_indices = SORT indices of C in ascending order*
3	*parents = new_pop[row_indices]*

4	*RETURN parents*

- Η διαδικασία της κυκλικής μετανάστευσης εκτελείται από την συνάρτηση Immigration().

IMMIGRATION(POPULATION, ISL_NUM)	
1	*FOR i FROM 0 TO isl_num - 1:*
2	*island_name = 'I' + (i + 1)*
3	*island_population = population[island_name]*
4	*num_chromosomes = SHAPE of island_population[0]*
5	*num_chromosomes_to_move = 10% of num_chromosomes*
6	
7	*// Select chromosomes to move*
8	*chromosomes_to_move_indices = RANDOM SAMPLE OF SIZE num_chromosomes_to_move FROM non_zero_chromosomes_indices*
9	*chromosomes_to_move = SELECT chromosomes_to_move_indices FROM island_population*
10	
11	*// Move chromosomes to the next island*
12	*next_island_name = 'I' + (i + 2) IF i + 2 <= isl_num ELSE 'I1'*
13	*next_island_population = population[next_island_name]*
14	*population[next_island_name] = STACK next_island_population AND chromosomes_to_move VERTICALLY*
15	*// Remove moved chromosomes from the current island*
16	*population[island_name] = DELETE rows AT chromosomes_to_move_indices FROM island_population*
17	*RETURN population*

- Ο αλγόριθμος τοπικής αναζήτησης 3-Opt, ο οποίος εφαρμόζεται στα 10 καλύτερα χρωμοσώματα της κάθε νήσου σε κάθε γενιά αναπτύσσεται στην παρακάτω συνάρτηση LSinbetween().

LSINBETWEEN(POPULATION, DEMAND, DISTANCE, NODES_NUM, VCAP, MAXROUTE, SERVTIME, ISL_NUM, LS_ITERATIONS)	
1	*// Initialize an empty list to collect the first 10 chromosomes from each island*
2	*first_ten_chromosomes = []*
3	*// Loop through each island in the population*
4	*FOR each island IN population:*
5	* // Check if the island has at least 10 chromosomes*
6	* IF LENGTH OF population[island] >= 10:*
7	* // Add the first 10 chromosomes of the island to the list*
8	* first_ten_chromosomes.EXTEND(population[island][:10])*
9	*// Convert the list of chromosomes to a NumPy array*
10	*first_ten_chromosomes = TO_NUMPY_ARRAY(first_ten_chromosomes)*
11	*// Initialize an empty list to store optimized chromosomes*
12	*INITIALIZE optimized_chromosomes AS EMPTY LIST*
13	*// Apply local search (TSP) to each chromosome in the list*
14	*FOR each chromosome IN first_ten_chromosomes:*
15	* // Optimize the chromosome using TSP*
16	* optimized_chromosome = CALL threeopt(chromosome, nodes_num, Demand, Distance, ls_iterations)*
17	* // Add the optimized chromosome to the list*
18	* optimized_chromosomes.APPEND(optimized_chromosome)*
19	*// Index to keep track of position in optimized_chromosomes list*
20	*INITIALIZE index TO 0*
21	*// Replace the first 10 chromosomes of each island with the optimized chromosomes*
22	*FOR each island IN population:*
23	* IF LENGTH OF population[island] >= 10:*
24	* // Replace the first 10 chromosomes of the island with optimized ones*
25	* population[island][:10] = optimized_chromosomes[index:index + 10]*
26	* // Move the index forward by 10*
27	* index += 10*
28	
29	*RETURN population*

- Στην συνάρτηση Replace_Worst_Island() εκτελούνται οι εντολές για την δημιουργία καινούργιας νήσου, με την χρήση της μεθόδου GRASP, στην θέση της χειρότερης νήσου.

REPLACE_WORST_ISLAND(POPULATION, NODES_NUM, DEMAND, DISTANCE, VCAP, MAXROUTE, SERVTIME)	
1	*// Create a list of nodes starting from 2 to nodes_num*
2	*node_list = LIST FROM 2 TO nodes_num*
3	*// Evaluate the costs of the first chromosome of each island*
4	*first_chromosomes = [first chromosome FROM each chromosome_list IN population.values()]*
5	
6	*costs = ARRAY OF costs FOR each chromosome IN first_chromosomes USING the cost function WITH parameters: Demand, Distance, nodes_num, vcap, maxroute, servtime*
7	*// Identify the worst chromosome and corresponding island*
8	*worst_index = FIND INDEX OF MAX value IN costs*
9	*worst_island_key = GET ISLAND key FROM population.keys() USING worst_index*
10	
11	*PRINT "Replacing population of", worst_island_key, "due to high cost."*
12	*island_size = LENGTH OF population[worst_island_key]*
13	
14	*// Generate new chromosomes using GRASP*
15	*new_chromosomes = LIST OF new chromosomes GENERATED USING GRASP FOR each element IN range(island_size)*
16	
17	*// Replace the old population with the new one*
18	*population[worst_island_key] = new_chromosomes*
19	*RETURN population*
20	

- Ο Έλεγχος και η επεξεργασία των όμοιων χρωμοσωμάτων γίνεται από την συνάρτηση Similarity_check(). Εάν πληθυσμός των ομοιοτήτων μεταξύ χρωμοσωμάτων υπερβαίνει το όριο του 50% τότε εκτελείται σε όλο τον πληθυσμό ο αλγόριθμος 3-Opt, όπως φαίνεται στην συνάρτηση Threeopt_similarity().

SIMILARITY_CHECK(POPULATION, NODES_NUM, DEMAND, DISTANCE)	
1	*// Define the similarity threshold*
2	*threshold = 0.5*
3	*num_chromosomes = LENGTH OF population*
4	*num_nodes = LENGTH OF population[0]*
5	*// Iterate over columns starting from 1 (0th column is always 1)*
6	*FOR column FROM 1 TO num_nodes EXCLUSIVE:*
7	*total_pairs = 0*
8	*similar_pairs = 0*
9	*// Compare each pair of chromosomes*
10	*FOR i FROM 0 TO num_chromosomes EXCLUSIVE:*
11	*FOR j FROM i + 1 TO num_chromosomes EXCLUSIVE:*
12	*total_pairs += 1*
13	*IF population[i][column] == population[j][column]:*
14	*similar_pairs += 1*
15	*// Calculate the similarity percentage*
16	*similarity_percentage = similar_pairs / total_pairs*
17	*// Check if similarity exceeds the threshold*
18	*IF similarity_percentage >= threshold:*
19	*// Apply a local search algorithm (e.g., 3-opt) to improve diversity*
20	*population = threeopt_similarity(population, nodes_num, Demand, Distance)*
21	*BREAK*
22	
23	*RETURN population*
THREEOPT _SIMILARITY(POPULATION, NODES_NUM, DEMAND, DISTANCE)	
1	*num_generation = 500*
2	*FOR path_index FROM 0 TO population.shape[0] - 1*
3	*Path = population[path_index]*
4	*FOR i FROM 1 TO num_generation*
5	*// Generate 2 unique random cut points to split into 3 segments*
6	*cuts = SORTED RANDOM SAMPLE OF 2 UNIQUE INTEGERS BETWEEN 1 AND (nodes_num - 1)*
7	*// Slice the path into 3 segments*
8	*vec1 = SUBARRAY OF Path FROM 1 TO cuts[0]*
9	*vec2 = SUBARRAY OF Path FROM cuts[0] TO cuts[1]*
10	*vec3 = SUBARRAY OF Path FROM cuts[1] TO END*

11	*segments = [vec1, vec2, vec3]*
12	*new_paths = []*
13	*// Generate all possible arrangements of the 3 segments*
14	*FOR EACH permutation IN itertools.permutations(segments)*
15	*new_Path = CONCATENATE([1], permutation)*
16	*ADD new_Path TO new_paths*
17	
18	*// Calculate the cost for each new path and choose the best one*
19	*first_cost = CALL cost_tsp(Path, Demand, Distance, nodes_num)*
20	*best_cost = first_cost*
21	*best_path = Path*
22	*FOR EACH new_Path IN new_paths*
23	*new_cost = CALL cost_tsp(new_Path, Demand, Distance, nodes_num)*
24	*IF new_cost < best_cost THEN*
25	*best_cost = new_cost*
26	*best_path = new_Path*
27	
28	*// Update the path to the best one found*
29	*Path = best_path*
30	*population[path_index] = Path*
31	*RETURN population*

5.2.3 Αλγόριθμος τοπικής αναζήτησης 3-Opt

- Από την διαδικασία του IMA προκύπτει ο καλύτερος απόγονος της τελευταίας γενιάς. Έπειτα σε αυτόν, μέσω της Threeopt() συνάρτησης, πραγματοποιούνται όσες γενιές τοπικής αναζήτησης καταχωρήσει ο χρήστης για περεταίρω βελτιστοποίηση που μπορεί να προκύψει από το συγκεκριμένο διάνυσμα-λύση. Στην Threeopt() καλείται και η συνάρτηση Cost_Threeopt() η οποία υπολογίζει την απόσταση που διανύουν τα οχήματα μίας λύσης μονοπατιού, και σκοπός του είναι η σύγκριση της επίδοσης των διανυσμάτων που προκύπτουν από τον αλγόριθμο τοπικής αναζήτησης.
  Η Cost_Threeopt() χρησιμοποιείται επίσης σε κάθε συνάρτηση που καλεί τον αλγόριθμο 3-Opt για τον παραπάνω σκοπό(Threeopt_similarity(), LSinbetween(), Threeopt_Grasp(), Mutation()) .

THREEOPT(PATH, NODES_NUM, DEMAND, DISTANCE, LS_ITERATIONS)	
1	*FOR i FROM 1 TO ls_iterations*

2	a = GENERATE RANDOM INTEGER BETWEEN 1 AND (nodes_num - 1)
3	b = GENERATE RANDOM INTEGER BETWEEN (a + 1) AND nodes_num
4	vec1 = SUBARRAY OF Path FROM 1 TO a
5	vec2 = SUBARRAY OF Path FROM a TO b
6	vec3 = SUBARRAY OF Path FROM b TO (nodes_num + 1)
7	// Generate all six possible new paths
8	candidates = [
9	CONCATENATE([1], vec1, vec3, vec2),
10	CONCATENATE([1], vec2, vec3, vec1),
11	CONCATENATE([1], vec2, vec1, vec3),
12	CONCATENATE([1], vec3, vec1, vec2),
13	CONCATENATE([1], vec3, vec2, vec1),
14	Path // Include the original path for comparison
15	]
16	// Calculate the cost for each candidate path
17	costs = [CALL cost_tsp(candidate, Demand, Distance, nodes_num) FOR EACH candidate IN candidates]
18	// Find the index of the candidate with the minimum cost
19	min_index = FIND INDEX OF MINIMUM VALUE IN costs
20	// Update the path to the best candidate
21	Path = candidates[min_index]
22	RETURN Path
COST_THREEOPT(NEW_POPULATION,DEMAND,DISTANCE,NODES_NUM)	
1	start = 1
2	D = 0
3	C = 0
4	Maxroute = 0
5	FOR j FROM 0 TO nodes_num - 3:
6	x = new_population[j]
7	y = new_population[j + 1]
8	x = x - 1 // Distance index shows the x-1 element
9	y = y - 1 // Distance index shows the y-1 element
10	D += Demand[y] // Accumulate demand
11	Maxroute += Distance[x, y] // Accumulate distance
12	// Check if demand and distance constraints are satisfied

13	*IF D < 160 AND Maxroute < 180:*
14	*C += Distance[x, y]*
15	*ELSE:*
16	*C += Distance[start, y] // New vehicle starts from start*
17	*D = 0 // Reset demand*
18	*Maxroute = 0 // Reset distance*
19	
20	*RETURN C*

5.2.4 Παρουσίαση αποτελεσμάτων

Οι παρακάτω συναρτήσεις εκτελούνται με σκοπό την επεξεργασία του τελικού αποτελέσματος και την παρουσίαση της λύσης στον χρήστη. Αναπτύχθηκαν, ουσιαστικά δυο συναρτήσεις. Η Restock() και η Visualise_Path().

- Η Restock() συνάρτηση λαμβάνει ως όρισμα το τελικό αποτέλεσμα και στόχος της είναι να παρουσιάσει στο τελικό αποτέλεσμα πότε ένα καινούργιο όχημα ξεκινάει από την αποθήκη. Έτσι επεξεργάζεται το τελικό διάνυσμα και όποτε οι περιορισμοί της χωρητικότητας του αμαξιού ή και του ορίου της μέγιστης απόστασης δεν τηρούνται η συνάρτηση παρουσιάζει τον κόμβο ‘1’ στο διάνυσμα και μάλιστα για περισσότερη ευκρίνεια εκτυπώνει, επίσης μήνυμα για την αποστολή καινούργιου οχήματος από την αποθήκη.

RESTOCK(PATH, NODES_NUM, DEMAND, DISTANCE)	
1	*D = 0*
2	*Maxroute = 0*
3	
4	*restock_path = CONVERT Path to list // Convert Path to list for easy insertion*
5	*FOR j FROM 1 TO nodes_num - 1: // Start from index 1 to check the route between nodes*
6	*x = restock_path[j - 1] // Previous city*
7	*y = restock_path[j] // Current city*
8	*// Adjust for 0-based indexing in the Demand and Distance matrices*
9	*y_index = y - 1*
10	*x_index = x - 1*
11	*D += Demand[y_index] // Accumulate demand*
12	*Maxroute += Distance[x_index, y_index] // Accumulate distance*

13	
14	*// Check if either demand or distance exceeds the limits*
15	*IF D > 160 OR Maxroute > 180:*
16	*// Print the city before which the restock happens*
17	*PRINT "Restock after city",x*
18	*INSERT '1' (restock point) BEFORE the current city in restock_path*
19	*D = 0 // Reset demand*
20	*Maxroute = 0 // Reset distance*
21	*RETURN restock_path*

- Η παρακάτω συνάρτηση Visualize_Path() είναι υπεύθυνη για την δημιουργία χάρτη που περιέχει όλους τους κόμβους στις συντεταγμένες τους και δείχνει τον τρόπο με τον οποίο η αποθήκη καλύπτει την ζήτηση όλων των κόμβων. Το δρομολόγιο που εκτελεί κάθε όχημα αποτυπώνεται με διαφορετικό χρώμα από τα υπόλοιπα ώστε να είναι όλα πιο ευδιάκριτα.

VISUALIZE_PATH(RESTOCK_PATH, COORDINATES)	
1	*INITIALIZE an empty graph G*
2	*// ADD NODES TO THE GRAPH*
3	*FOR each coordinate WITH index i IN coordinates*
4	*ADD node i+1 TO G with position as coordinate*
5	
6	*// ADD EDGES TO THE GRAPH*
7	*FOR each node i IN restock_path UNTIL second last node*
8	*IF next node in restock_path is not node 1 THEN*
9	*ADD an edge FROM current node TO next node*
10	
11	*// GET node positions from the graph*
12	*// DRAW all nodes in blue except node 1*
13	*DRAW nodes from 2 to the number of coordinates IN blue*
14	*DRAW node 1 IN red*
15	*INITIALIZE colors list*
16	*INITIALIZE color_index TO 0* *// DRAW EDGES FOR EACH VEHICLE IN DIFFERENT COLORS*
17	*FOR each node i IN restock_path UNTIL second last node*
18	*IF current node is node 1 THEN*
19	*CHANGE color_index TO the next color*

20	*IF next node is not node 1 THEN*
21	*DRAW edge FROM current node TO next node USING the color at color_index*
22	
23	*ADD LABELS TO THE NODES*
24	*// DISPLAY the graph*
25	*SHOW plot*

6. Αποτελέσματα

Για την λήψη αποτελεσμάτων και τη πλήρη κατανόηση αυτών και τον αποτελεσμάτων είναι σημαντική η δοκιμή του κώδικα σε πολλά περιβάλλοντα με διαφορετικές συνθήκες. Έτσι, έγινε εφαρμογή πολλών παραδειγμάτων, με διαφορετικό αριθμό κόμβων και με διαφορετική

χρήση του 3-opt αλγορίθμου τοπικής αναζήτησης. Σε αυτά τα παραδείγματα δοκιμάστηκαν επανηλημένες φορές διαφορετικοί αριθμοί γενεών, νησιών και μεταναστεύσεων. Παρακάτω παρατίθενται τα πιο ενδιαφέροντα από τα αποτελέσματα που προέκυψαν.

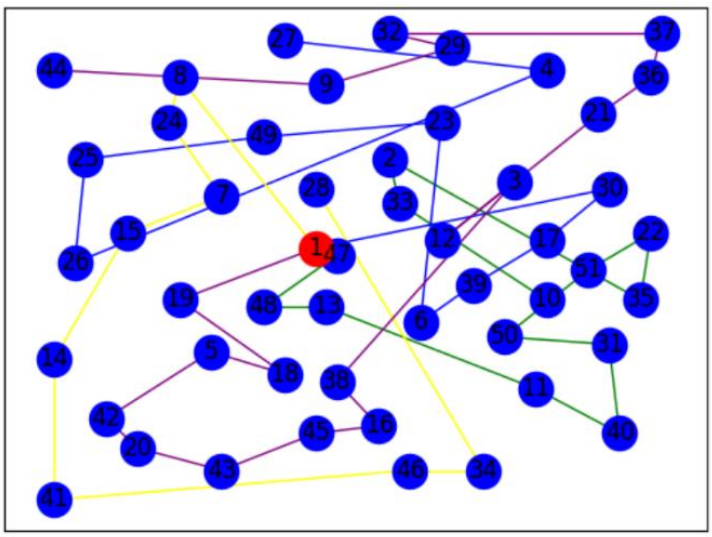
6.1 Πρώιμη μορφή IMA

Στην προκειμένη περίπτωση θα παρουσιαστούν τα αποτελέσματα που προέκυψαν από την διαδικασία του IMA δίχως όμως την εκτέλεση του 3-Opt αλγορίθμου ενδιάμεσα των γενεών και χωρίς ποινή της χειρότερης, αποδοτικά, νήσου. Αυτή εφαρμόστηκε με διάφορους αριθμούς νήσων σε ένα παράδειγμα 51 κόμβων και τα αποτελέσματα του παρουσιάζονται παρακάτω:

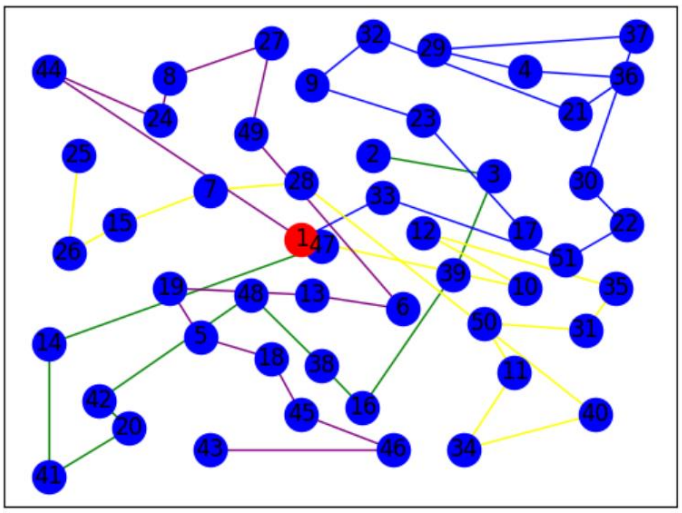
	Isl=2	*Isl=5*	*Isl=10*	*Isl=15*	*Isl=20*
Pop_Size=200	636.27	604.01	558.42	525.9	543.27
Gen=400	After LS:	After LS:	After LS:	After LS:	After LS:
Immigrations=20	610.59	596.19	551.46	516.2	517.7
LS_iterations=500					

Στην παραπάνω εφαρμογή παρατηρείται το εξής φαινόμενο : Ενώ ο πληθυσμός παραμένει ίδιος, όσο τα νησιά, στα οποία αυτός μοιράζεται στην αρχή του IMA, αυξάνονται τόσο καλύτερο είναι και το αποτέλεσμα που παράγεται. Βέβαια όταν τα νησιά δεν φέρουν μεγάλο πληθυσμό η πρόωρή σύγκλιση είναι αναπόφευκτη, καθώς ένα χρωμόσωμα είναι πιο πιθανό να διασταυρωθεί με ένα άλλο που ήδη έχουν ξανασυναντηθεί σε προηγούμενη γενιά. Πέραν των παραπάνω παρατηρείται επίσης από τα διαγράμματα ότι τα οχήματα δεν ακολουθούν το βέλτιστο μονοπάτι. Ακόμη και στην καλύτερη λύση που δημιουργήθηκε (*Για isl=15*), διακρίνεται εύκολα ότι τα οχήματα υπερπηδούν κόμβους για να επισκεφθούν άλλους που βρίσκονται σε πολύ μεγαλύτερη απόσταση(Κόμβος ‘40’→Κόμβος ‘2’ , Κόμβος ‘44’→Κόμβος ‘28’). Το γεγονός αυτό μαζί με την αποφυγή της πρόωρης σύγκλισης οδηγούν στο συμπέρασμα ότι ο αλγόριθμος δεν παράγει το καλύτερο δυνατό αποτέλεσμα και έτσι αυτός χρειάζεται αναβάθμιση.

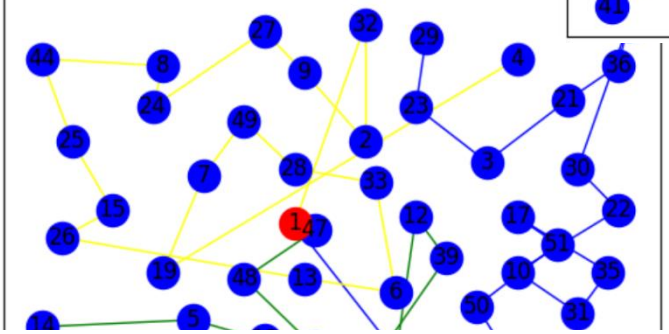
isl=2



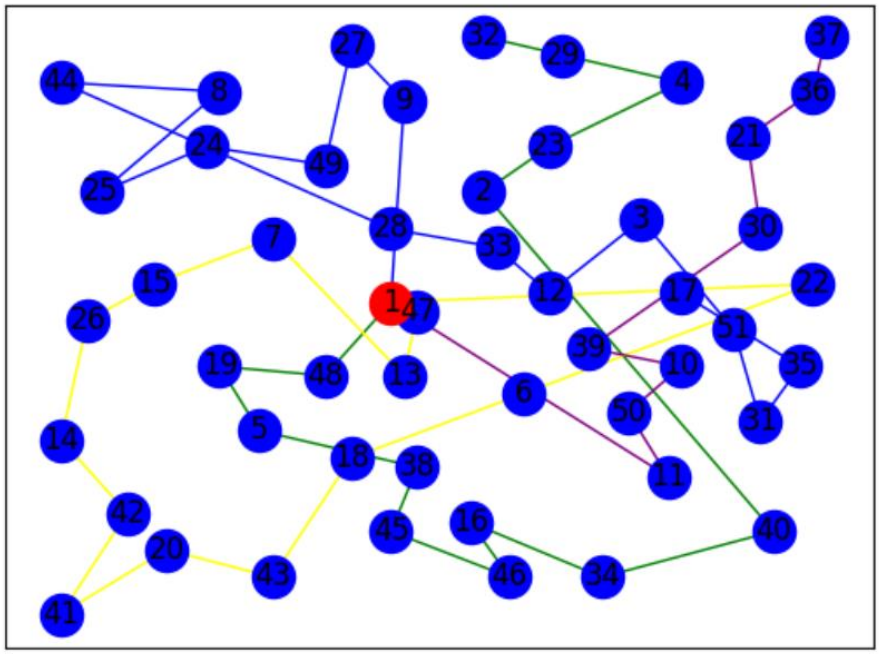
isl=5

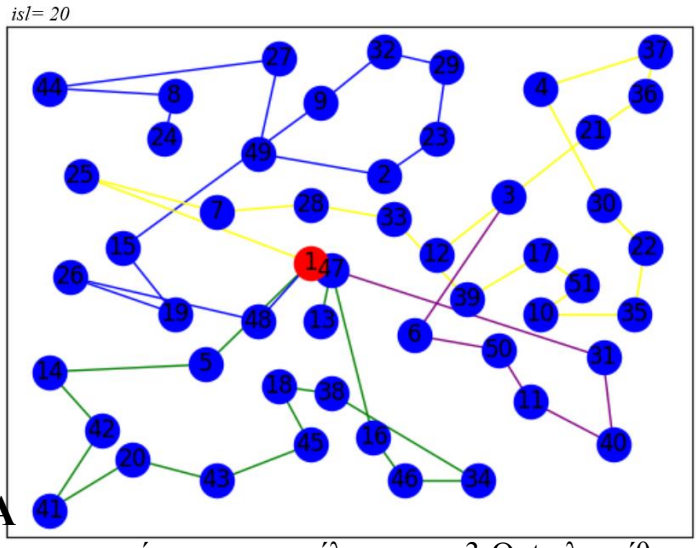


isl= 10



isl=15





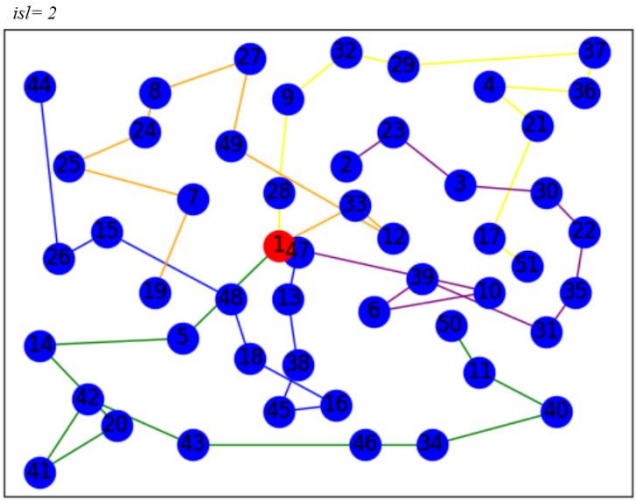
6.2 Τελική μορφή ΙΜΑ

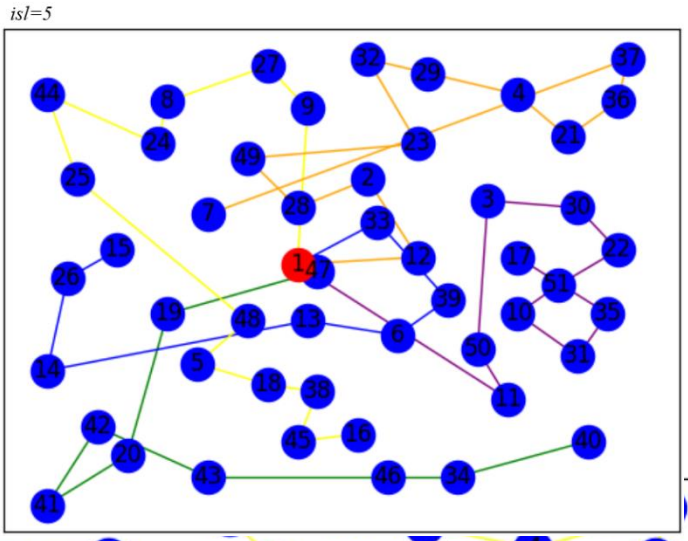
Με την ανάπτυξη των δύο συναρτήσεων της ποινής και της εκτέλεσης του 3-Opt αλγορίθμου, η πρόωρη σύγκλιση των λύσεων έχει αποφευχθεί. Η δυνατότητα του αλγορίθμου να εκτελέσει περισσότερες γενιές, χωρίς αυτές να είναι πρακτικά αναποτελεσματικές, όπως στην πρώιμη μορφή όπου τα αποτελέσματα σύγκλιναν πολύ γρήγορα σε μία λύση, είναι πλέον εφικτή.

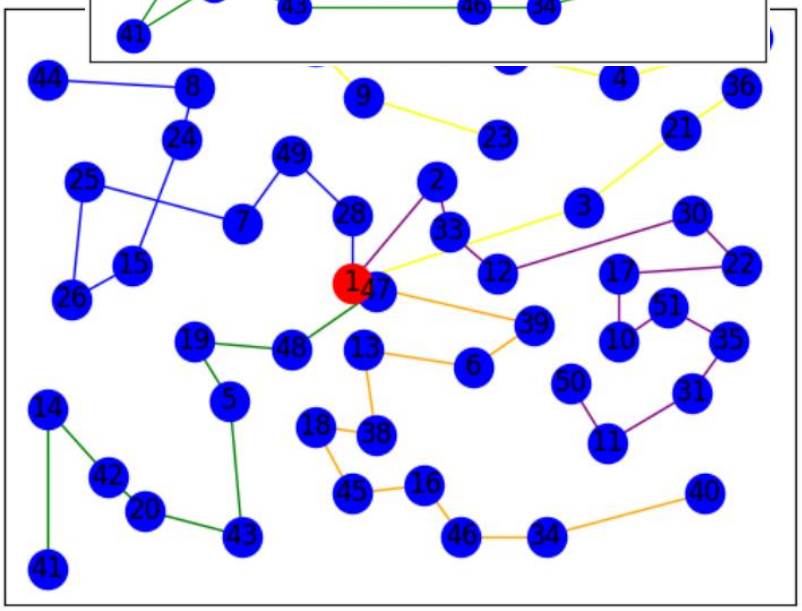
Στην πρώιμη κατάσταση η επιλογή μεγάλου αριθμού γενεών ήταν περιττή καθώς στις τελευταίες γενιές ο έλεγχος των χρωμοσωμάτων και η εκτέλεση του 3-Opt δεν ήταν αρκετά για την δημιουργία ποικιλομορφίας των λύσεων. Πλέον, η ποικιλομορφία προκύπτει κυρίως από την δημιουργία καινούργιας νήσου ενώ η εφαρμογή του 3-Opt βελτιώνει το αποτέλεσμα από γενιά σε γενιά.

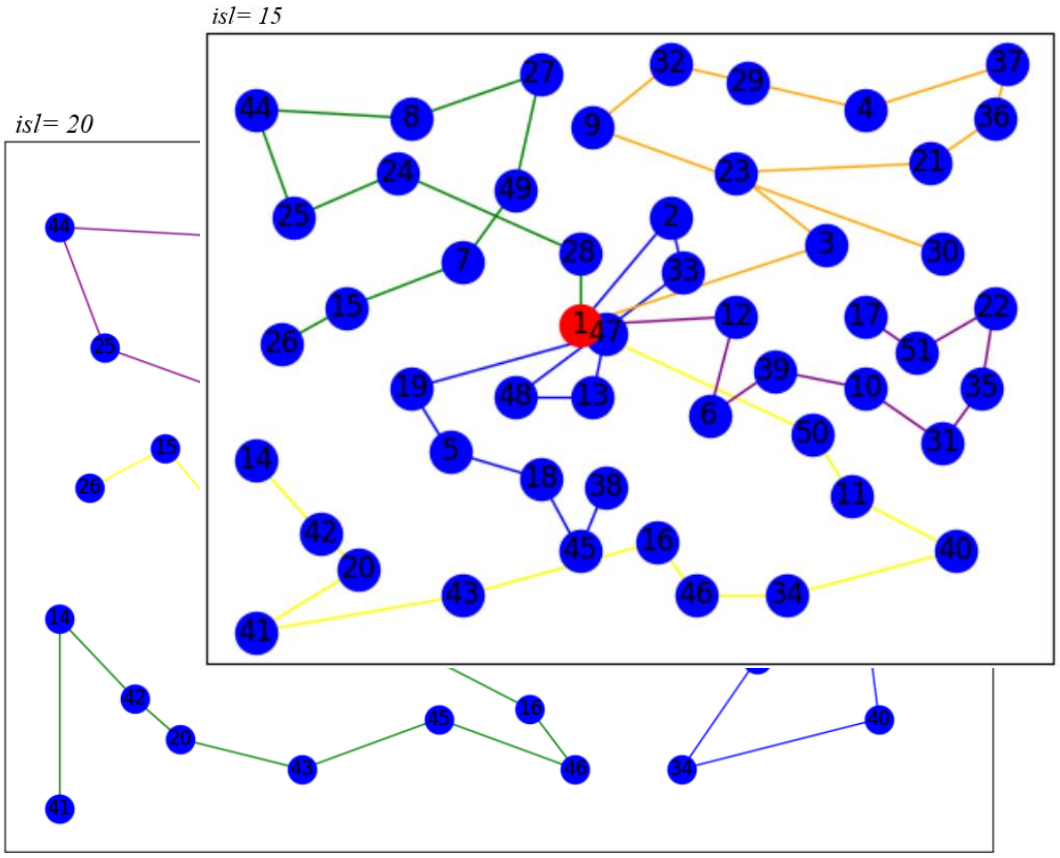
	Isl=2	*Isl=5*	*Isl=10*	*Isl=15*	*Isl=20*
Pop_Size=400 *Gen=600~700* *Immigrations=30* *LS_iterations=500* 50	569.42 After LS: 536.49	561.15 After LS: 505.4	457.55 After LS: 416.46	486.62 After LS: 477.57	494.14 After LS: 481.82

Όπως παρατηρείται και στον παραπάνω πίνακα καθώς και στις εικόνες των μονοπατιών, τα αποτελέσματα του αλγορίθμου είναι σαφώς καλύτερα , καθώς φαίνεται η μείωση του τελικού αποτελέσματος κατά 80-100 μονάδες. Ταυτόχρονα παρατηρείται ότι και πάλι ο αριθμός των νήσων παίζει καθοριστικό ρόλο στην απόδοση του IMA, με την καλύτερη επίδοση να την δημιουργεί το σύμπλεγμα των 10 νήσων.









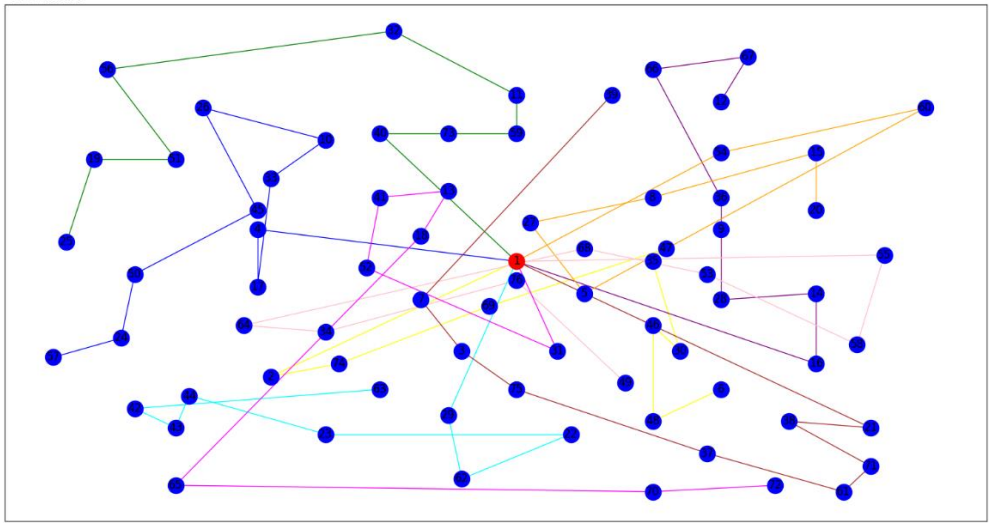
Είναι αξιοσημείωτη η ικανότητα του αλγορίθμου να επιλύσει και προβλήματα διαφορετικού μεγέθους. Παρακάτω αναλύονται τα αποτελέσματα από 5 διαφορετικά παραδείγματα. Καταχωρούνται οι ίδιες τιμές του μιμητικού αλγορίθμου ενώ επιλέγετε η τιμή των 10 νήσων αφού είχε την καλύτερη επίδοση στο προηγούμενο παράδειγμα

	75 nodes	*200 nodes*	*101 nodes*	*151 nodes*
Pop_Size=400 Gen=600~700	1007.59	2583.46	1131.99	1827.16

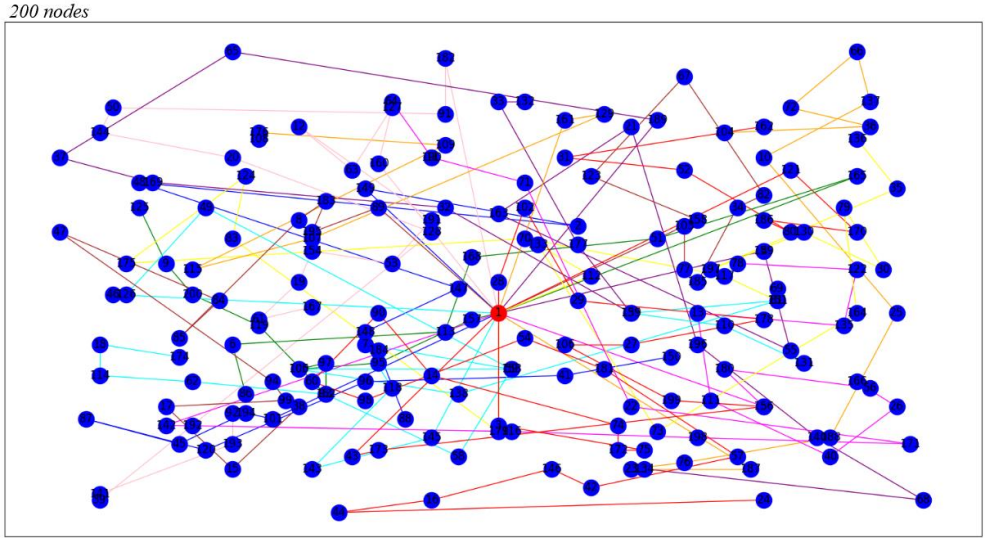
Isl=10 *Immigrations=30* *LS_iterations=500*	After LS: 877.65	After LS: 2410.44	After LS: 1050.88	After LS: 1612.04

Από τα διαγράμματα παρατηρείται ότι όσο μεγαλύτερο το πρόβλημα τόσο πιο χρονοβόρα είναι και η επίλυση του. Για την εύρεση ενός ικανοποιητικού αποτελέσματος σε ένα πρόβλημα 200 κόμβων απαιτείται η καταχώρηση μεγαλύτερου δείγματος και περισσότερων γενεών – επαναλήψεων του αλγορίθμου. Ενώ η διερεύνηση λύσης για ένα πρόβλημα 76 κόμβων τείνει σε πιο ικανοποιητικά αποτελέσματα από την λειτουργία του μιμητικού αλγορίθμου στις παραπάνω συνθήκες. Παρακάτω παρουσιάζονται τα διαγράμματα για τα 4 διαφορετικά μεγέθη.

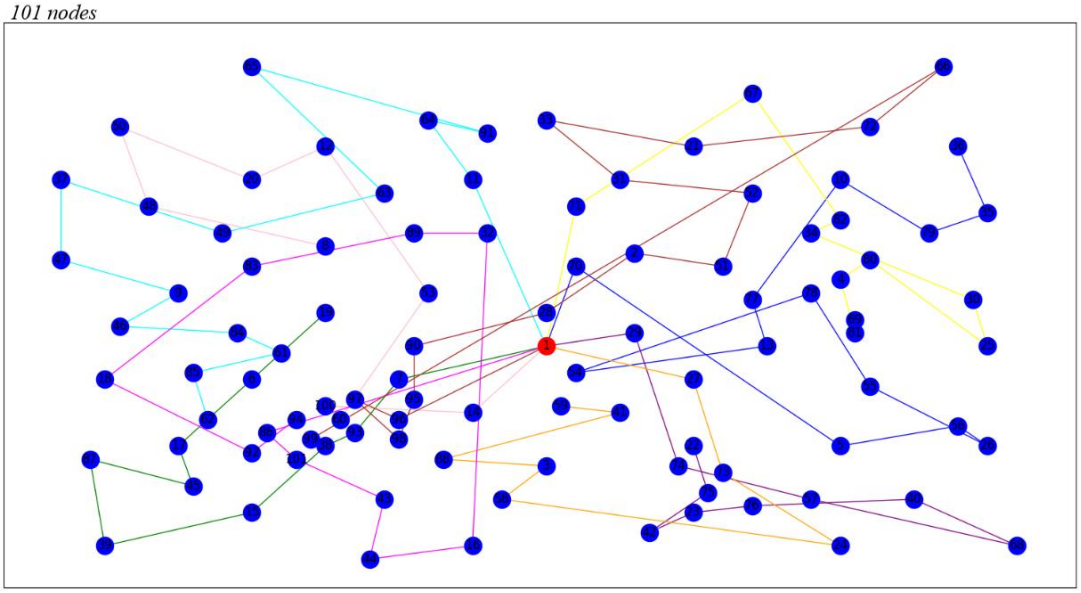
76 nodes



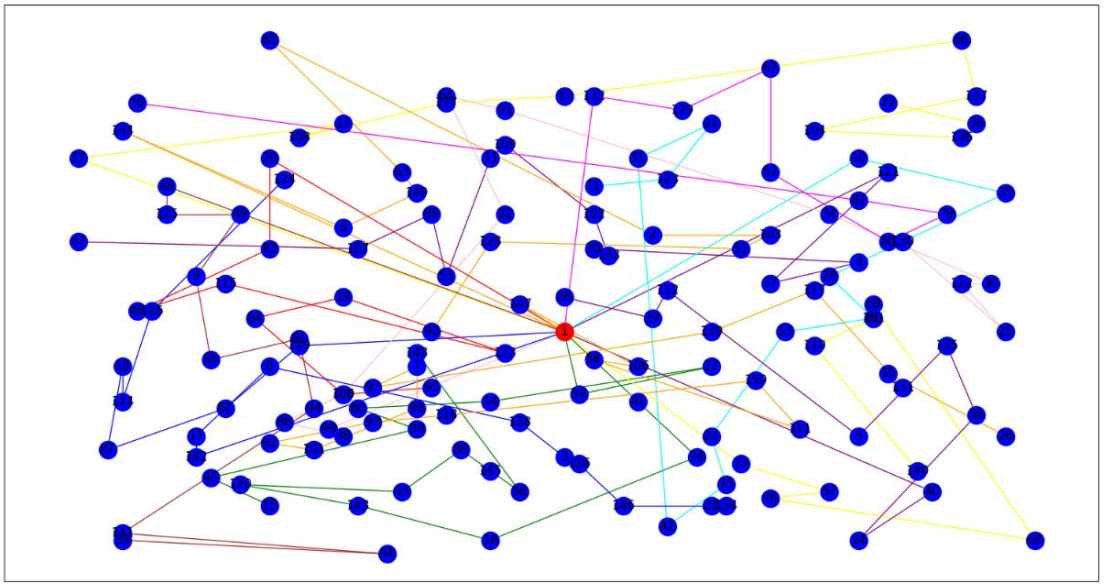
200 nodes



101 nodes



151 nodes



7. Συμπεράσματα

Ο Μιμητικός αλγόριθμος είναι αρκετά ευέλικτος ως προς την εφαρμογή του. Το ίδιο μοντέλο έχει την δυνατότητα να βελτιστοποιήσει το οποιοδήποτε πρόβλημα δρομολόγησης οχημάτων, εάν και εφόσον γίνουν οι κατάλληλες προσαρμογές στην αντικειμενική συνάρτηση που αυτός θα χρησιμοποιεί για αξιολόγηση των χρωμοσωμάτων. Ωστόσο χρειάζεται να υποστηρίζεται και από τα κατάλληλα εργαλεία. Η μέθοδος GRASP , για παράδειγμα, είναι αρκετά σημαντική καθώς ακόμη και εάν και αυτή , όπως ο μιμητικός αλγόριθμος βασίζεται εν μέρη στην τύχη για την παραγωγή του αρχικού πληθυσμού, αυτή είναι πιο πιθανό να επιλέξει για την επόμενη επίσκεψη τον πιο κοντινό κόμβο. Μια τέτοια διαδικασία προσφέρει μια κατευθυντική πορεία στον αλγόριθμο. Και ενώ η μέθοδος GRASP μπορεί να οδηγεί σε μία σύγκλιση οι υπόλοιπες διαδικασίες όπως η δημιουργία καινούργιας νήσου στη θέση μιας παλιάς, οι μεταλλάξεις και ο έλεγχος ομοιότητας απαιτείται να δημιουργούν ποικιλομορφία στις λύσεις με σκοπό την δημιουργία καινούργιων μονοπατιών.

8. Πηγές

[1]	Rogdakis, I., Marinaki, M., Marinakis, Y., & Migdalas, A. (2017). An island memetic algorithm for real world vehicle routing problems. In *Operational Research in Business and Economics: 4th International Symposium and 26th National Conference on Operational Research, Chania, Greece, June 2015* (pp. 205-223). Springer International Publishing.
[2]	Vaessens, R. J., Aarts, E. H., & Lenstra, J. K. (1998). A local search template. *Computers & Operations Research*, *25*(11), 969-979.
[3]	Neri, F., & Cotta, C. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, *2*, 1-14.
[4]	Vaessens, R. J., Aarts, E. H., & Lenstra, J. K. (1998). A local search template. *Computers & Operations Research*, *25*(11), 969-979.
[5]	Li, F., Golden, B., & Wasil, E. (2007). The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & operations research*, *34*(10), 2918-2930.
[6]	El-Ghazali Talbi, Metaheuristics: from design to implementation. New. Wiley Series on Parallel and Distributed Computing. Wiley, 2009.
[7]	Gendreau, M., Potvin, J.-Y., Metaheuristics in Combinatorial Optimization. Annals of Operations Research. 140, 190-197 ???, 2005.