ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Αλγόριθμοι Βελτιστοποίησης

Δρομολόγιο απορριμματοφόρου στον Άγιο Γεώργιο

Ομάδα 2

Γιαννέστρα Ειρήνη Μαρίνα 4432

Γιβανούδη Ελένη 4314

Κουφόπουλος Ανδρέας 4479

Επιβλέπων Καθηγητής:

Μιχαηλίδης Γεώργιος

ΚΑΒΑΛΑ
ΙΟΥΝΙΟΣ 2023

Περιεχόμενα

Περίληψη

Στις μέρες μας, λόγω της πολυπλοκότητας και του μεγάλου όγκου δεδομένων των προβλημάτων επίλυσης υπάρχει αναγκαιότητα για περεταίρω μεθόδους επίλυσης, καθώς και βελτιστοποίησης αυτών. Για αυτό το λόγο χρησιμοποιούμε ευριστικές μεθόδους, οι οποίες είναι σαν ένα παιχνίδι αναζήτησης και βελτίωσης. Αυτές κάνουν επαναληπτικά μία περιορισμένη αναζήτηση στον χώρο λύσεων, για να βελτιώνουν μία υπάρχουσα λύση.

Μία τέτοια ευριστική μέθοδος επίλυσης αποτελεί ο γενετικός αλγόριθμός, ο οποίος μιμείται τη διαδικασία της φυσικής επιλογής της εξέλιξης. Όπως οι οργανισμοί έτσι και οι λύσεις ενός προβλήματος, μέσα από την εφαρμογή γενετικών τελεστών, μεταβιβάζουν χαρακτηριστικά από γενιά σε γενιά, οδηγώντας σταδιακά στην εξαφάνιση κακών λύσεων και στη βελτίωση καλών λύσεων.

Η παρούσα εργασία αποτελείται από τρία (3) κεφάλαια. Στο πρώτο γίνεται μια εισαγωγή πάνω στις έννοιες του αλγορίθμου και της βελτιστοποίησης αλγορίθμων. Στο δεύτερο κεφάλαιο εξηγείται ο τρόπος λειτουργίας ενός γενετικού αλγορίθμου, καθώς επίσης τα πλεονεκτήματα και οι περιορισμοί του. Στο τρίτο κεφάλαιο παρουσιάζεται η διατύπωση του προβλήματος βελτιστοποίησης και ο τρόπος επίλυσης αυτού μέσω του γενετικού αλγόριθμο. Τέλος, στο παράρτημα, υπάρχει ο κώδικας του αλγορίθμου σε

Abstract

Nowadays, due to the complexity and large amount of data of the solving problems there is a necessity for further solving methods, as well as their optimization. For this reason we use heuristic methods, which are like a game of search and improvement. They iteratively perform a limited search in the solution space to improve an existing solution.

One such heuristic solution method is the genetic algorithm, which mimics the natural selection process of evolution. Like organisms, solutions to a problem, through the application of genetic operators, transmit characteristics from generation to generation, gradually leading to the disappearance of bad solutions and the improvement of good solutions.

This thesis consists of three (3) chapters. In the first, an introduction is made to the concepts of algorithm and algorithm optimization. The second chapter explains how a genetic algorithm works, as well as its advantages and limitations. The third chapter presents the formulation of the optimization problem and the way to solve it through the genetic algorithm. Finally, in the appendix, there is the code of the algorithm in Python.

Κεφάλαιο1: Εισαγωγή στους Αλγόριθμους

1. Εισαγωγή

Αρχικά έχει αποδειχθεί πέραν πάσης αμφιβολίας ότι οι μετα-ευρετικοί αλγόριθμοι βελτιστοποίησης έχουν καλές επιδόσεις, καθώς χειρίζονται βέλτιστα διάφορες ευέλικτες εργασίες βελτιστοποίησης του πραγματικού κόσμου, από τη ρομποτική [1], τα ασύρματα δίκτυα [2][3], τα συστήματα ενέργειας [4][5], τον προγραμματισμό εργασιών [6][7] έως την ταξινόμηση [8] και την εκπαίδευση τεχνητών νευρωνικών δικτύων [9]. Κατά την αναζήτηση της συνολικά καλύτερης (σχεδόν βέλτιστης) λύσης, οι μεταευρετικοί αλγόριθμοι απαιτούν πολυάριθμες αξιολογήσεις καταλληλότητας.

Συνήθως αποτελεί σοβαρό εμπόδιο για την εφαρμογή των μεταευρετικών αλγορίθμων σε προβλήματα βελτιστοποίησης υψηλής υπολογιστικής ικανότητας, τα οποία υπάρχουν εκτενώς στην υπολογιστική βελτιστοποίηση της δυναμικής ρευστών [10] και στη δομική βελτιστοποίηση [11], μεταξύ πολλών άλλων. Για την επεξεργασία αυτών των προβλημάτων, η απόδοση των υποψήφιων λύσεων αξιολογείται συνήθως με προσεγγίσεις αριθμητικής ανάλυσης υψηλής πιστότητας (π.χ. προσομοιώσεις υπολογιστικής ρευστοδυναμικής ή ανάλυση πεπερασμένων στοιχείων), οι οποίες μπορεί να αφαιρούν χρόνο CPU από λεπτά έως ώρες ή ακόμη και ημέρες [12][13].

Ως εκ τούτου, στο πλαίσιο του παραδείγματος των μετα-ευρετικών μεθόδων, προτάθηκαν πολλές τεχνικές και παραλλαγές της νοημοσύνης σμήνους (SI) για την αντιμετώπιση πολύπλοκων/μεγάλης κλίμακας προβλημάτων βελτιστοποίησης.

1.1 Ορισμός Αλγορίθμου

Ένας αλγόριθμος είναι μια ακολουθία οδηγιών ή βημάτων που προγραμματίζονται για να επιλύσουν ένα συγκεκριμένο πρόβλημα ή να εκτελέσουν μια συγκεκριμένη εργασία. Αναλυτικά, οι αλγόριθμοι περιλαμβάνουν την πλήρη περιγραφή των βημάτων που απαιτούνται για την εκτέλεση μιας εργασίας, από την αρχή μέχρι το τέλος.

1.2 Ορισμός βελτιστοποίηση

Βελτιστοποίηση είναι η επινόηση και η εφαρμογή Επιστημονικών Μεθόδων για την επίλυση Επιχειρησιακών Προβλημάτων με σκοπό τον καλύτερο έλεγχο και τον ΒΕΛΤΙΣΤΟ τρόπο λειτουργίας οργανωμένων συστημάτων ή διαδικασιών.

1.3 Ανάπτυξη αλγορίθμων

Οι αλγόριθμοι αναπτύσσονται για να επιλύουν προβλήματα και να εκτελούν εργασίες με κάποιον τρόπο που είναι καθορισμένος, προβλέψιμος και αποτελεσματικός. Μπορούν να χρησιμοποιηθούν σε πολλούς τομείς, όπως η Υπολογιστική Επιστήμη, η Μηχανική, η Τεχνητή Νοημοσύνη, η Κρυπτογραφία, η Βιοπληροφορική και πολλοί άλλοι. Ένας καλός αλγόριθμος πρέπει να είναι κατανοητός, αποτελεσματικός, ακριβής και να τρέχει με φρενήρη ταχύτητα. Συνήθως, υπάρχουν πολλοί τρόποι να επιλυθεί ένα πρόβλημα, οπότε οι αλγόριθμοι μπορούν να διαφέρουν ανάλογα με τον τρόπο που επιλύουν τα προβλήματα και τις μεθόδους που χρησιμοποιούν. Οι αλγόριθμοι μπορούν να είναι απλοί, μερικώς πολύπλοκοι ή πολύπλοκοι, ανάλογα με τον βαθμό δυσκολίας και την απαιτούμενη υπολογιστική ισχύ. Στη φάση της σχεδίασης ενός αλγορίθμου χρησιμοποιείται μια συνδυαστική προσέγγιση των ακόλουθων στοιχείων:

- **Είσοδος (Input)**

  Ο αλγόριθμος πρέπει να καθορίζει τις εισόδους που αναμένει για να εκτελέσει την εργασία του. Οι είσοδοι μπορεί να είναι δεδομένα, παράμετροι ή συνθήκες.

- **Έξοδος (Output)**

  Ο αλγόριθμος πρέπει να καθορίζει τις αναμενόμενες εξόδους μετά την εκτέλεση των βημάτων. Οι έξοδοι μπορεί να είναι αποτελέσματα, αναφορές ή αποφάσεις.

- **Επεξεργασία (Processing)**

  Ο αλγόριθμος πρέπει να περιλαμβάνει τα ακριβή βήματα που πρέπει να ακολουθηθούν για να εκτελεστεί η εργασία.

Αυτά τα βήματα μπορούν να περιλαμβάνουν μαθηματικές, λογικές, συγκριτικές, επαναληπτικές και άλλες πράξεις που απαιτούνται για την επίτευξη του επιθυμητού αποτελέσματος. Αυτές οι επεξεργασίες μπορεί να περιλαμβάνουν μεταβλητές, συναρτήσεις, δομές ελέγχου (όπως συνθήκες και βρόχους), αλγοριθμικές πράξεις (όπως αναζήτηση, ταξινόμηση, προσθήκη, αφαίρεση) και άλλα.

1.4 Τι είναι αλγόριθμοι βελτιστοποίησης

Οι αλγόριθμοι βελτιστοποίησης είναι υπολογιστικές μέθοδοι ή τεχνικές που χρησιμοποιούνται για την εύρεση της καλύτερης λύσης ή βέλτιστων τιμών για ένα δεδομένο πρόβλημα. Αυτοί οι αλγόριθμοι στοχεύουν στην ελαχιστοποίηση ή τη μεγιστοποίηση μιας αντικειμενικής συνάρτησης, η οποία αντιπροσωπεύει ένα μέτρο απόδοσης ή ποιότητας που πρέπει να βελτιστοποιηθεί.

Σε διάφορους τομείς όπως τα μαθηματικά, η επιστήμη των υπολογιστών, η μηχανική και η επιστήμη δεδομένων, οι αλγόριθμοι βελτιστοποίησης διαδραματίζουν κρίσιμο ρόλο στην επίλυση πολύπλοκων προβλημάτων όπου υπάρχουν πολλές πιθανές λύσεις. Αυτοί οι αλγόριθμοι εξερευνούν επαναληπτικά τον χώρο της λύσης και κάνουν προσαρμογές για τη βελτίωση της αντικειμενικής συνάρτησης μέχρι να βρεθεί μια ικανοποιητική λύση.

1.5 Κατηγορίες Αλγορίθμων

Οι αλγόριθμοι μπορούν να διακριθούν σε διάφορες κατηγορίες ανάλογα με τις ιδιότητες και τη λειτουργία τους. Ορισμένες συνήθεις κατηγορίες αλγορίθμων περιλαμβάνουν:

- **Αλγόριθμοι αναζήτησης**

  Χρησιμοποιούνται για την εύρεση μιας συγκεκριμένης τιμής ή αντικειμένου σε ένα σύνολο δεδομένων. Παραδείγματα αλγορίθμων αναζήτησης είναι ο αλγόριθμος δυαδικής αναζήτησης και ο αλγόριθμος γραμμικής αναζήτησης.

- **Αλγόριθμοι ταξινόμησης**

Χρησιμοποιούνται για την οργάνωση ενός συνόλου δεδομένων σε μια συγκεκριμένη σειρά. Ο αλγόριθμος ταξινόμησης των φυσαλίδων και ο αλγόριθμος γρήγορης ταξινόμησης (quicksort) είναι δύο παραδείγματα αλγορίθμων ταξινόμησης.

- **Αλγόριθμοι αναδρομής**

  Χρησιμοποιούνται όταν η λύση ενός προβλήματος μπορεί να επιτευχθεί με αναδρομική κλήση, δηλαδή όταν η λύση εξαρτάται από την επίλυση μικρότερων υποπροβλημάτων του ίδιου τύπου. Ένα παράδειγμα αλγορίθμου αναδρομής είναι ο αλγόριθμος του Πύργου του Χάνοι.

- **Αλγόριθμοι δυναμικού προγραμματισμού**

  Χρησιμοποιούνται για την επίλυση προβλημάτων που μπορούν να υπολογιστούν αποδοτικά, αποθηκεύοντας και επαναχρησιμοποιώντας ενδιάμεσα αποτελέσματα. Ο αλγόριθμος του σακιδίου (knapsack problem) και ο αλγόριθμος της ακολουθίας Fibonacci είναι παραδείγματα αλγορίθμων δυναμικού προγραμματισμού.

- **Αλγόριθμος καθολικού αποκλεισμού (Tabu)**

  Αυτό επιτρέπει στον αλγόριθμο να εξερευνήσει νέες περιοχές του χώρου αναζήτησης και να αποφύγει την εγκλωβισμένη σε κοιλάδες της λύσης. Ο Tabu Search είναι κατάλληλος για προβλήματα με περιορισμένο αριθμό διαθέσιμων λύσεων και όταν η γειτονική αναζήτηση μπορεί να οδηγήσει σε βελτίωση της τρέχουσας λύσης.

- **Αλγόριθμος βελτιστοποίησης μυρμηγκιών (ACO)**

  Ο ACO αλγόριθμος χρησιμοποιεί ένα σύνολο εικονικών μυρμηγκιών που ακολουθούν διαφορετικές διαδρομές σε ένα γράφο, αναζητώντας τη βέλτιστη διαδρομή. Καθώς τα μυρμήγκια προχωρούν, αφήνουν φερομόνες στους ακμές που διασχίζουν. Οι φερομόνες αυτές λειτουργούν ως αποθήκη πληροφοριών για το πού έχουν βρεθεί καλές λύσεις στο παρελθόν. Ο ACO είναι κατάλληλος για προβλήματα βελτιστοποίησης που αφορούν την επιλογή διαδρομής, όπως προβλήματα ταξιδιού και προβλήματα routing.

- **Γενετικός αλγόριθμος (GA)** Ένας γενετικός αλγόριθμος (GA) είναι ένας ευρετικός αλγόριθμος αναζήτησης που χρησιμοποιείται για την επίλυση προβλημάτων αναζήτησης και

βελτιστοποίησης. Αυτός ο αλγόριθμος είναι ένα υποσύνολο εξελικτικών αλγορίθμων, οι οποίοι χρησιμοποιούνται στον υπολογισμό.

1.6 Παράγοντες Αλγορίθμου

Η επιλογή του κατάλληλου αλγορίθμου εξαρτά από τη φύση του προβλήματος που πρέπει να επιλυθεί, τις διαθέσιμες πηγές και περιορισμούς, καθώς και τις απαιτήσεις απόδοσης. Κάποιοι παράγοντες που μπορούν να ληφθούν υπόψη κατά την επιλογή ενός αλγορίθμου περιλαμβάνουν:

- **Αποδοτικότητα**

  Η αποδοτικότητα ενός αλγορίθμου αξιολογείται βάσει του χρόνου εκτέλεσης και της απαιτούμενης μνήμης. Είναι σημαντικό να επιλεγεί ο αλγόριθμος που μπορεί να επιλύσει το πρόβλημα με τον πιο αποδοτικό τρόπο για τη συγκεκριμένη περίπτωση.

- **Ακρίβεια**

  Ορισμένα προβλήματα απαιτούν ακριβείς απαντήσεις, ενώ άλλα μπορούν να αποδώσουν προσεγγίσεις ή προσεγγίσεις με σφάλματα. Ανάλογα με την ακρίβεια που απαιτείται, μπορεί να επιλεγεί ο κατάλληλος αλγόριθμος.

- **Διαθέσιμος χρόνος**

  Ο διαθέσιμος χρόνος για την επίλυση του προβλήματος μπορεί να επηρεάσει την επιλογή του αλγορίθμου. Σε ορισμένες περιπτώσεις, ο διαθέσιμος χρόνος μπορεί να περιορίζει τη χρήση πιο περίπλοκων αλγορίθμων που απαιτούν περισσότερο χρόνο εκτέλεσης. Αν υπάρχει περιορισμένος χρόνος για την επίλυση ενός προβλήματος, μπορεί να επιλεγεί ένας αλγόριθμος που προσφέρει γρηγορότερη εκτέλεση, ακόμη και αν η λύση του δεν είναι τελείως ακριβής.

- **Απλότητα**

  Σε ορισμένες περιπτώσεις, η απλότητα του αλγορίθμου μπορεί να είναι προτιμότερη από πολύπλοκες λύσεις. Ένας απλός αλγόριθμος μπορεί να είναι πιο εύκολος στην

κατανόηση, την υλοποίηση και τη συντήρηση, και μπορεί να επαρκεί για την επίλυση του συγκεκριμένου προβλήματος.

- **Δυνατότητα επέκτασης**

  Σε ορισμένες περιπτώσεις, είναι σημαντικό να επιλεγεί ένας αλγόριθμος που μπορεί να επεκταθεί και να προσαρμοστεί εύκολα για να αντιμετωπίσει πιθανές μελλοντικές απαιτήσεις ή αλλαγές.

1.7 Επιλογή Αλγορίθμου

Η επιλογή ενός αλγορίθμου είναι ένα προσεκτικό και στρατηγικό βήμα κατά την επίλυση ενός προβλήματος. Οι παράγοντες που εξετάζονται στην επιλογή ενός αλγορίθμου περιλαμβάνουν τις απαιτήσεις του προβλήματος, τους περιορισμούς, την ακρίβεια και τον διαθέσιμο χρόνο. Επιπλέον, η επιλογή ενός αλγορίθμου εξαρτάται από τις γνώσεις και τις δεξιότητες του υλοποιητή.

Κατά την επιλογή ενός αλγορίθμου, μπορεί να είναι χρήσιμο να γίνει μια ανάλυση της πολυπλοκότητας του αλγορίθμου, όπως ο χρόνος εκτέλεσης και οι απαιτήσεις μνήμης. Μια αξιολόγηση της πολυπλοκότητας μπορεί να βοηθήσει να κατανοήσετε πόσο αποδοτικός είναι ο αλγόριθμος σε διάφορες περιπτώσεις χρήσης.

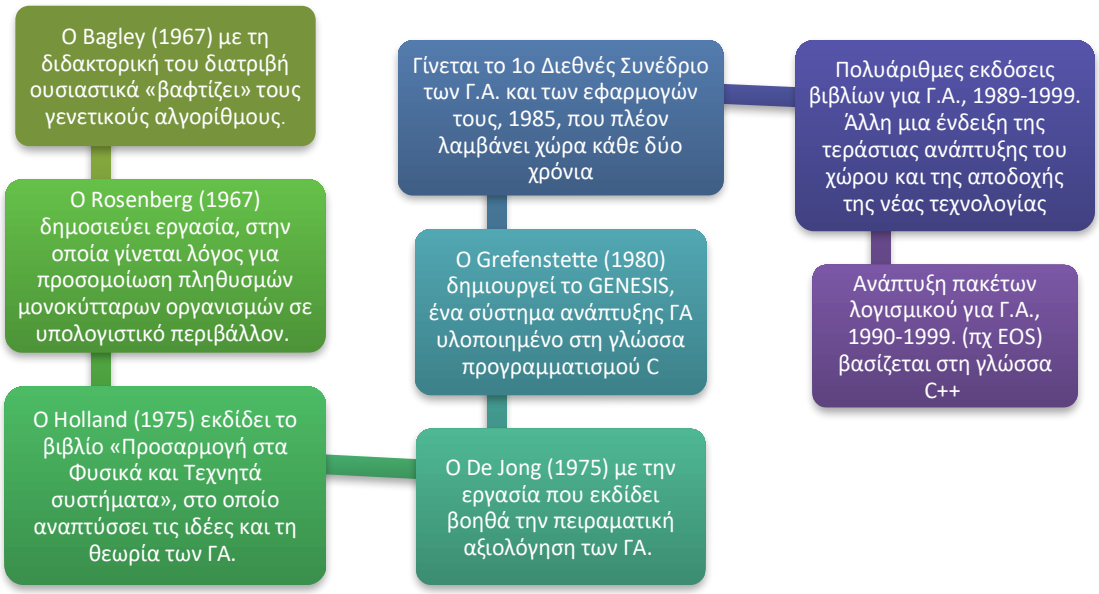
Κεφάλαιο 2: Γενετικός Αλγόριθμος

2.1 Γενετικός Αλγόριθμος

Ο γενετικός αλγόριθμος (GA), που αναπτύχθηκε από τον John Holland και τους συνεργάτες του τις δεκαετίες του 1960 και του 1970, είναι ένα μοντέλο ή αφαίρεση της βιολογικής εξέλιξης που βασίζεται στη θεωρία της φυσικής επιλογής του Κάρολου Δαρβίνου.

Ο Holland ήταν πιθανώς ο πρώτος που χρησιμοποίησε το crossover και τον ανασυνδυασμό, τη μετάλλαξη και την επιλογή στη μελέτη προσαρμοστικών και τεχνητών συστημάτων. Αυτοί οι γενετικοί τελεστές αποτελούν το ουσιαστικό μέρος του γενετικού αλγορίθμου ως στρατηγική επίλυσης προβλημάτων. Έκτοτε, πολλές παραλλαγές γενετικών αλγορίθμων έχουν αναπτυχθεί και εφαρμοστεί σε ένα ευρύ φάσμα προβλημάτων βελτιστοποίησης [14].

2.2 Ιστορική αναδρομή του GA [15]



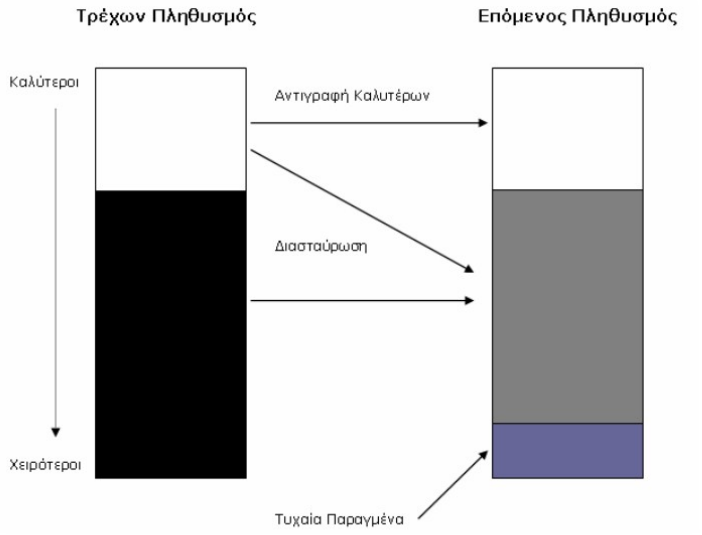
2.3 Τρόπος λειτουργίας του Γενετικού Αλγορίθμου

Αρχικά, πρέπει να καθοριστεί η κωδικοποίηση που θα χρησιμοποιηθεί. Στη συνέχεια, χρησιμοποιώντας μια τυχαία διαδικασία, δημιουργείται ένας αρχικός πληθυσμός συμβολοσειρών. Στη συνέχεια, ένα σύνολο τελεστών χρησιμοποιείται για να πάρει αυτόν τον αρχικό πληθυσμό για να δημιουργήσει διαδοχικούς πληθυσμούς, οι οποίοι ελπίζουμε ότι βελτιώνονται με το χρόνο. Οι κύριοι τελεστές των γενετικών αλγορίθμων είναι η **αναπαραγωγή**, η **διασταύρωση** και η **μετάλλαξη** [16].

1. **Επιλογή (Selection)** : Η διαδικασία επιλογής καθορίζει ποια άτομα από τον τρέχοντα πληθυσμό θα επιλεγούν ως γονείς για την επόμενη γενιά. Τα άτομα με υψηλότερες βαθμολογίες φυσικής κατάστασης είναι πιο πιθανό να επιλεγούν, καθώς έχουν μεγαλύτερη πιθανότητα να μεταδώσουν τα ευνοϊκά χαρακτηριστικά τους στην επόμενη γενιά. Μπορούν να χρησιμοποιηθούν διάφορες στρατηγικές επιλογής, όπως η επιλογή τροχού ρουλέτας ή η επιλογή τουρνουά.

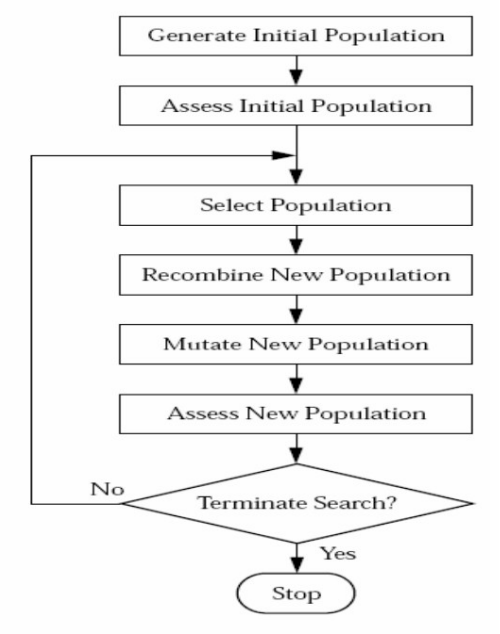
2. **Αναπαραγωγή (Reproduction)** : Είναι μια διαδικασία που βασίζεται στην αντικειμενική συνάρτηση (συνάρτηση καταλληλότητας) κάθε χορδής. Αυτή η αντικειμενική συνάρτηση προσδιορίζει πόσο «καλή» είναι μια συμβολοσειρά. Έτσι, οι χορδές με υψηλότερη αξία φυσικής κατάστασης έχουν μεγαλύτερη πιθανότητα να συνεισφέρουν απογόνους στην επόμενη γενιά [16].

3. **Διασταύρωση (Crossover)** : Περιλαμβάνει το συνδυασμό γενετικών πληροφοριών από δύο γονικά άτομα για τη δημιουργία νέων απογόνων. Αυτή η ανταλλαγή επιτρέπει την εξερεύνηση διαφορετικών συνδυασμών γενετικού υλικού και μπορεί να οδηγήσει στη δημιουργία δυνητικά καλύτερων λύσεων.



Εικόνα 1: Μεταβατική διαδικασία μεταξύ διαδοχικών γονέων

4. **Μετάλλαξη (Mutation)** : Είναι η περιστασιακή (με πιθανότητα μικρότερη του 1%) τυχαία αλλαγή της τιμής μιας θέσης συμβολοσειράς. Βοηθά στη διατήρηση της ποικιλομορφίας στον πληθυσμό και εισάγει νέες γενετικές πληροφορίες που μπορεί να οδηγήσουν σε νέες και βελτιωμένες λύσεις. Σκοπός του είναι να διασφαλίσει ότι σημαντικές πληροφορίες που περιέχονται στις συμβολοσειρές ενδέχεται να μην χαθούν πρόωρα.



Εικόνα 2 : Τα βασικά βήματα ενός Γενετικού αλγορίθμου

2.4 Πλεονεκτήματα – Περιορισμοί Γενετικού Αλγορίθμου

Πλεονεκτήματα γενετικού αλγορίθμου

- ☑ Έχει εξαιρετικές παράλληλες δυνατότητες.
- ☑ Μπορεί να βελτιστοποιήσει διάφορα προβλήματα όπως διακριτές συναρτήσεις, προβλήματα πολλαπλών στόχων και συνεχείς συναρτήσεις.
- ☑ Παρέχει απαντήσεις που βελτιώνονται με την πάροδο του χρόνου.
- ☑ Ένας γενετικός αλγόριθμος δεν χρειάζεται παράγωγες πληροφορίες

Περιορισμοί γενετικών αλγορίθμων

- ☒ Δεν είναι αποτελεσματικά στην επίλυση απλών προβλημάτων.
- ☒ Η έλλειψη σωστής εφαρμογής μπορεί να κάνει τον αλγόριθμο να συγκλίνει σε μια λύση που δεν είναι βέλτιστη.
- ☒ Η ποιότητα της τελικής λύσης δεν είναι εγγυημένη.
- ☒ Ο επαναλαμβανόμενος υπολογισμός των τιμών καταλληλότητας μπορεί να δημιουργήσει ορισμένα προβλήματα στην αντιμετώπιση υπολογιστικών προκλήσεων.

Κεφάλαιο3: Ανάλυση Προβλήματος

3.1 Στάδια της μεθόδου

Η αντιμετώπιση ενός προβλήματος όπου πρέπει να ληφθεί η βέλτιστη απόφαση περιλαμβάνει συνήθως τα εξής στάδια:

- **Αναγνώριση και περιγραφή του προβλήματος**
  Δεν αρκεί μόνο η παρατήρηση των συμπτωμάτων, αλλά πρέπει να εντοπιστούν οι αιτίες του προβλήματος. Επίσης πρέπει να εξεταστεί αν συνδέεται με άλλα προβλήματα και πρέπει να καθοριστούν οι στόχοι με αντικειμενικό τρόπο. Είναι το πιο σημαντικό βήμα γιατί οποιοδήποτε λάθος θα οδηγήσει σε αποτυχία τα επόμενα στάδια.

- **Καθορισμός των παραμέτρων του προβλήματος**
  Ποιοι παράγοντες επηρεάζουν τη λύση και πως μπορούμε να τους μεταβάλουμε ώστε να έχουμε εναλλακτικές λύσεις.

- **Εντοπισμός των περιορισμών του προβλήματος**
  Ποιοι είναι οι περιορισμοί ή τα όρια μέσα στα οποία μπορούμε να κινηθούμε.

- **Αναζήτηση λύσεων και επιλογή της βέλτιστης λύσης**
  Αφού βρεθούν οι εφικτές λύσεις, επιλέγεται η Βέλτιστη λύση, με βάση των αντικειμενικό στόχο που θέσαμε στο στάδιο 1.

- **Δοκιμή και υλοποίηση-εφαρμογή της βέλτιστης λύσης**
  Στο τελευταίο βήμα η προτεινόμενη λύση δοκιμάζεται και αν επιτύχει, εφαρμόζεται στο πραγματικό πρόβλημα. Είναι το πιο δύσκολο βήμα γιατί ακόμη και η τέλεια λύση αν εφαρμοστεί λάθος θα αποτύχει.

3.2 Διατύπωση προβλήματος βελτιστοποίησης

Η εργασία αναφέρεται στο πρόβλημα εύρεσης του βέλτιστου δρομολογίου ενός απορριμματοφόρου. Στο πρόβλημα αυτό το απορριμματοφόρο πρέπει να περάσει μόνο μία φορά από τους 124 κάδους, της περιοχής του Άγιου Γεωργίου στην Καβάλα, στην διάρκεια μιας μέρας (24 ώρες).

- Το απορριμματοφόρο πρέπει να ακολουθήσει μια ενιαία συνεχή διαδρομή.
- Τα σημεία που βρίσκονται οι κάδοι είναι γεωμετρικά
- Δεν υπάρχει ο περιορισμός της φοράς του δρόμου
- Το απορριμματοφόρο μπορεί να κινηθεί προς όλες τις κατευθύνσεις

Το βασικό κριτήριο βελτιστοποίησης είναι η συνολική χιλιομετρική απόσταση που θα διανύσει το απορριμματοφόρο.

Ερώτημα: Ποια είναι η βέλτιστη διαδρομή του απορριμματοφόρου, ώστε να μεγιστοποιείται το κέρδος σε χρόνο και χιλιομετρική απόσταση;

3.3 Επεξήγηση επιλογής αλγορίθμου βελτιστοποίησης

Για την επίλυση του προβλήματος της βέλτιστης διαδρομής ενός απορριμματοφόρου στην περιοχή του Αγίου Γεωργίου στην Καβάλα, χρησιμοποιήθηκε ο **γενετικός αλγόριθμος (GA).** Επιλέχτηκε το συγκεκριμένο είδος αλγορίθμου διότι ο GA έχει τη δυνατότητα να εξερευνά αποτελεσματικά έναν μεγάλο χώρο λύσεων και μπορεί να χειρίζεται πολύπλοκα, πολυδιάστατα προβλήματα. Επιπλέον μπορεί να βρει μια παγκόσμια βέλτιστη ή σχεδόν βέλτιστη λύση. Λόγω ότι χρησιμοποιεί μηχανισμούς όπως η διασταύρωση και η μετάλλαξη μπορεί να προσεγγίσει τον χώρο αναζήτησης με τρόπο που διατηρεί την ποικιλία και εξερευνά τις διάφορες πιθανές λύσεις. Αυτό το χαρακτηριστικό τον καθιστά εύρωστο και ανθεκτικό στην πιθανή παγίδα της τοπικής ελάχιστης λύσης.

Επίσης λόγω ότι το πρόβλημα ταιριάζει με το πρόβλημα του πλανόδιου πωλητή, στο οποίο το GA θα μπορούσε να χρησιμοποιηθεί για τη βελτιστοποίηση της τοποθεσίας ή της επιλογής των σημείων πώλησης. Οι πιθανές λύσεις θα μπορούσαν να αντιπροσωπεύουν διαφορετικούς συνδυασμούς

σημείων και η αντικειμενική συνάρτηση θα μπορούσε να βασίζεται σε παράγοντες όπως η κίνηση με τα πόδια, ο ανταγωνισμός και η εγγύτητα σε πελάτες-στόχους.

Ακόμη αν στο μέλλον προκύψουν νέοι περιορισμοί ή απαιτήσεις στη διαδρομή των απορριμματοφόρων, ο γενετικός αλγόριθμος μπορεί να προσαρμοστεί εύκολα για να αντιμετωπίσει τις νέες συνθήκες. Ωστόσο, το GA μπορεί να είναι υπολογιστικά ακριβό, ειδικά για προβλήματα μεγάλης κλίμακας, και απαιτεί τον καθορισμό κατάλληλων γενετικών τελεστών και τον συντονισμό διαφόρων παραμέτρων.

3.4 Αλγόριθμοι που απορρίφθηκαν

Οι αλγόριθμοι που απορρίφθηκαν για την υλοποίηση της εφαρμογής η οποία υπολογίζει την βέλτιστη διαδρομή είναι οι εξής:

- **Αναζήτηση Tabu:** Ο συγκεκριμένος αλγόριθμος απορρίφθηκε διότι ενδέχεται να απαιτεί μεγάλο αριθμό επαναλήψεων για να συγκλίνει και μπορεί να είναι υπολογιστικά ακριβή για μεγάλες περιπτώσεις προβλημάτων. Επίσης βασίζεται σε μεγάλο βαθμό στον συντονισμό παραμέτρων, όπως η θητεία ταμπού και το μέγεθος της γειτονιάς.

- **Particle Swarm Optimization (PSO):** Ο συγκεκριμένος αλγόριθμος απορρίφθηκε διότι είναι σχετικά απλό στην εφαρμογή και απαιτεί λιγότερες παραμέτρους για συντονισμό σε σύγκριση με το GA. Μπορεί να συγκλίνει γρήγορα σε μια καλή λύση και να χειριστεί προβλήματα συνεχούς ή διακριτής βελτιστοποίησης. Ωστόσο, το PSO μπορεί να δυσκολεύεται με το τοπικό βέλτιστο και μπορεί να είναι ευαίσθητο στο μέγεθος του σμήνους και τις αρχικές ρυθμίσεις παραμέτρων. Μπορεί να μην εξερευνήσει το χώρο λύσης τόσο εκτενώς όσο ο GA.

- **Ant Colony Optimization (ACO):** Ο συγκεκριμένος αλγόριθμος απορρίφθηκε διότι είναι κατάλληλο για προβλήματα συνδυαστικής βελτιστοποίησης και υπερέχει στην εύρεση σχεδόν βέλτιστων λύσεων. Μπορεί να προσαρμοστεί σε μεταβαλλόμενα περιβάλλοντα και είναι λιγότερο ευαίσθητο στις αρχικές παραμέτρους. Ωστόσο, το ACO μπορεί να απαιτεί μεγάλο αριθμό επαναλήψεων για να συγκλίνει και μπορεί να είναι υπολογιστικά ακριβό για μεγάλες

περιπτώσεις προβλημάτων. Το ACO ήταν η επόμενη πιθανή επιλογή στο πρόβλημα στην περίπτωση που δεν θα δούλευε ο GA.

- **Multi-Objective NSGA :** Η πρώτη απόπειρα για την λύση του προβλήματος έγινε με το Multi-Objective NSGA**,** ωστόσο λόγω δυσκολίας στην σύνταξη του κώδικα απορρίφτηκε.

3.5 Περιγραφή συναρτήσεων που χρησιμοποιήθηκαν

3.5.1 Συνάρτηση readPoints(filename)

Όνομα	**`readPoints`**		
Λειτουργία	Διαβάζει ένα kml αρχείο και εξάγει το xml περιερχόμενο των σημείων σε έναν δισδιάστατο πίνακα points		
Παράμετροι	**`FileName`**		
Μορφή	str		
Περιγραφή	Το όνομα του kml αρχείου		
Τιμές Επιστροφής	**`points`**	**`names`**	**`size`**
Μορφή	numpy.ndarray	list	int
Περιγραφή	Ένας 2D πίνακας με τις συνταγμένες των σημείων του kml αρχείου	Λίστα με τα ονόματα των σημείων	Ο αριθμός των σημείων

3.5.2 Συνάρτηση calcDistances(points, size)

Όνομα	**`calcDistances`**	
Λειτουργία	Υπολογίζει τις αποστάσεις μεταξύ των σημείων	
Παράμετροι	**`points`**	**`size`**
Μορφή	numpy.ndarray	int
Περιγραφή	Ένας 2D πίνακας με τις συνταγμένες των σημείων	Ο αριθμός των σημείων

Τιμές Επιστροφής	`G`
Μορφή	numpy.ndarray
Περιγραφή	Ένας 2D πίνακας με τις αποστάσεις (βάρη/κόστος) μεταξύ των σημείων

3.5.3 Συνάρτηση calcPathDistance(G, path)

Όνομα	`calcPathDistance`	
Λειτουργία	Υπολογίζει τη συνολική απόσταση ενός μονοπατιού	
Παράμετροι	`G`	`path`
Μορφή	numpy.ndarray	numpy.ndarray
Περιγραφή	Ένας 2D πίνακας με τις αποστάσεις μεταξύ των σημείων	Ένας πίνακας με τους δείκτες (indices) των σημείων του μονοπατιού
Τιμές Επιστροφής	`dist`	
Μορφή	float	
Περιγραφή	Η συνολική απόσταση του μονοπατιού	

3.5.4 Συνάρτηση crossover(p1, p2)

Όνομα	`crossover`	
Λειτουργία	Κάνει crossover μεταξύ δύο μονοπάτια (γονείς), δημιουργώντας ένα καινούργιο μονοπάτι από αυτά (παιδί)	
Παράμετροι	`p1`	`p2`
Μορφή	numpy.ndarray	numpy.ndarray
Περιγραφή	Ένας πίνακας με δείκτες των σημείων του πρώτου μονοπατιού	Ένας πίνακας με δείκτες των σημείων του δεύτερου μονοπατιού
Τιμές Επιστροφής	`child`	
Μορφή	numpy.ndarray	
Περιγραφή	Ένας πίνακας με τους δείκτες των σημείων του καινούργιου μονοπατιού (child path)	

3.5.5 Συνάρτηση mutation(path)

Όνομα	`mutation`
Λειτουργία	Κάνει mutation σε ένα μονοπάτι
Παράμετροι	`path`
Μορφή	numpy.ndarray
Περιγραφή	Ένας πίνακας με τους δείκτες των σημείων του μονοπατιού
Τιμές Επιστροφής	`path`
Μορφή	numpy.ndarray
Περιγραφή	Ένας πίνακας με τους δείκτες των σημείων του μονοπατιού μετά από bit-swap

3.5.6 Συνάρτηση genetic_algorithm(G, size, pop_size, gen_limit)

Όνομα	`genetic_algorithm`			
Λειτουργία	Υλοποιεί το γενετικό αλγόριθμο για να βρει το μικρότερο μονοπάτι			
Παράμετροι	**G**	**size**	**pop_size**	**gen_limit**
Μορφή	numpy.ndarray	int	int	int
Περιγραφή	Ένας 2D πίνακας με τις αποστάσεις (βάρη/κόστος) μεταξύ των σημείων	Ο αριθμός των σημείων	Το μέγεθος του πληθυσμού	Ο αριθμός των γενεών που θα πραγματοποιηθούν

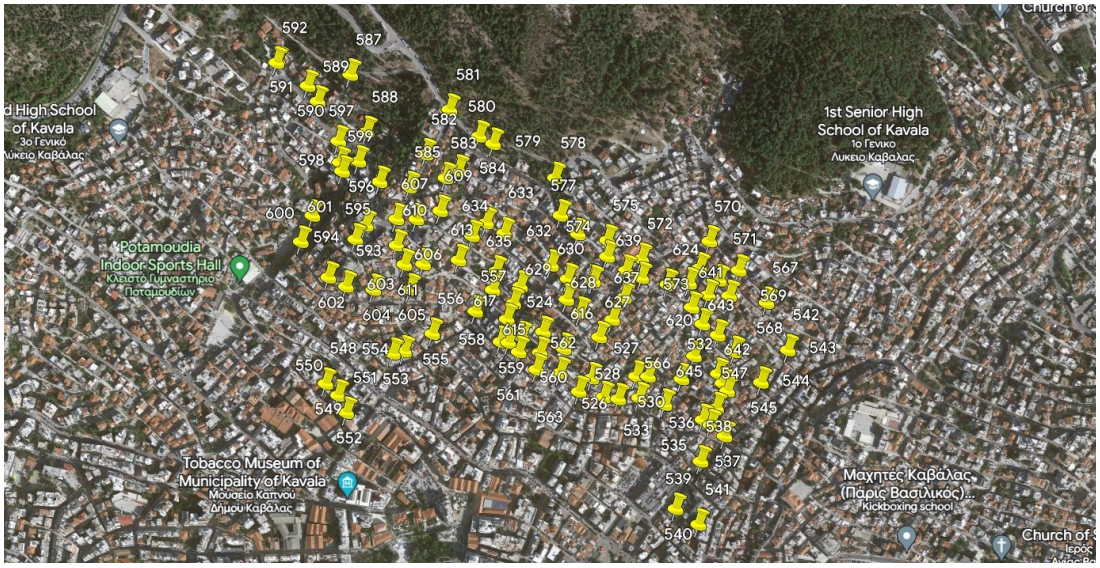
Τιμές Επιστροφής	`bestPath`	`bestDistance`	`gens`
Μορφή	numpy.ndarray	float	int
Περιγραφή	Ένας πίνακας με τους δείκτες των σημείων του μονοπατιού	Η συνολική απόσταση από το μικρότερο μονοπάτι	Ο αριθμός των γενεών που πραγματοποιήθηκαν

3.6 Επεξήγηση επιρροής παραμέτρων

Στην συνάρτηση readPoints η παράμετρος FileName αποτελεί ένα kml αρχείο. Η επιλογή αυτής της παραμέτρου έγινε για να εξάγουμε το xml περιερχόμενο των σημείων στην περιοχή του Αγίου Γεωργίου προκειμένου να πάρουμε τις απαραίτητες πληροφορίες που θα χρησιμοποιήσουμε στην συνέχεια, όπως τις συντεταγμένες, το συνολικό πλήθος των σημείων, το όνομα του κάθε σημείου, κ.ο-.κ. Έπειτα, οι παράμετροι points και size που αντιστοιχούν στην συνάρτηση calcDistances χρησιμοποιούνται για να υπολογιστεί το κόστος μεταξύ των σημείων. Ακόμα, στην συνάρτηση calcPathDistance οι παράμετροι G και path έχουν άμεση επιρροή και χρησιμότητα στην επίλυση του προβλήματος καθώς παρέχουν σημαντικές πληροφορίες για τον υπολογισμό και δημιουργία των πιθανών τελικών μονοπατιών και χρησιμοποιούνται εκτενώς στον κώδικα. Η παράμετρος path έχει χρήση και στη συνάρτηση mutation, όπου πραγματοποιείται τυχαία μετάλλαξη αντιμετάθεσης των bits του μονοπατιού για υψηλότερο βαθμό “επιβίωσης”. Επιπλέον, στην συνάρτηση crossover, οι παράμετροι p1 και p2, που αποτελούν τους δύο μονοπάτια όπου μέσα από την διαδικασία της διασταύρωσης θα κατασκευαστεί ένα νέο μονοπάτι, δεν έχουν περαιτέρω χρήση πέρα από αυτό. Παρόλα αυτά, η διαδικασία της διασταύρωσης έχει την μεγαλύτερη επιρροή στον αλγόριθμό. Τέλος, οι παράμετροι pop_size και gen_limit στην συνάρτηση genetic_algorithm επηρεάζουν τον επιθυμητό αριθμό πληθυσμού για τον οποίο θα εφαρμοστεί ο γενετικός αλγόριθμος και το πλήθος των επαναλήψεων που θα πραγματοποιήσει ο κώδικας αντίστοιχα.

3.7 Επίλυση προβλήματος

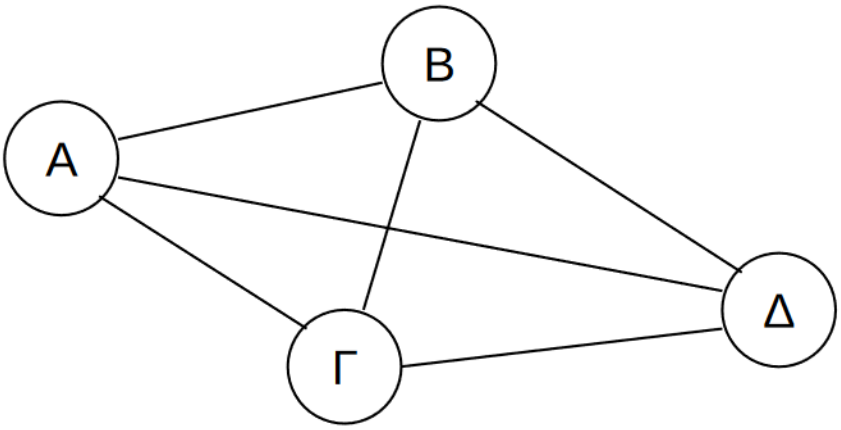
Στην παρούσα εργασία υλοποιήσαμε ένα γενετικό αλγόριθμο που ακολουθεί την μεθοδολογία για την επίλυση του προβλήματος του Περιοδεύοντος Πωλητή προκειμένου να βρει τη βέλτιστη λύση. Ειδικότερα, ο αλγόριθμος διανύει τη μικρότερη συνολική απόσταση αναζητώντας το βέλτιστο μονοπάτι (τη μικρότερη διαδρομή) που θα περάσει από όλα τα σημεία και πιο συγκεκριμένα τους κάδους απορριμάτων στην περιοχή του Αγίου Γεωργίου στο χάρτη της Καβάλας. Κατασκευάσαμε έξι (6) συναρτήσεις για την επίλυση του προβλήματος τις οποίες την λειτουργία τις αναλύουμε παρακάτω.



Εικόνα 3 : Περιοχή του Αγίου Γεωργίου από το Google Earth

Αρχικά, εισάγουμε τις απαιτούμενες βιβλιοθήκες που θα χρησιμοποιηθούν στον κώδικα μας (*γρ.* 1-6). Έπειτα, με την συνάρτηση readPoints(fileName), η οποία παίρνει ως παράμετρο το όνομα του kml αρχείου με τα σημεία, και ανοίγουμε το αρχείο αυτό για να πάρουμε το XML περιερχόμενο του με χρήση του parser (*γρ.* 10-11). Αρχικοποιούμε μια κενή λίστα names και το πρώτο στοιχείο xml του φακέλου στην μεταβλητή folder για να υπολογίσουμε τον συνολικό αριθμό των σημείων (size) (*γρ.* 12-14). Δημιουργούμε έναν δισδιάστατο πίνακα points που τον γεμίζουμε, με μία δομή επανάληψης for για κάθε παιδί του φακέλου, με τα ονόματα των σημείων και τις συντεταγμένες x,y αυτών (*γρ.* 16-27). Σημειώνεται ότι, μπορούμε να αρχειοθετήσουμε τα σημεία με τυχαία σειρά εξαρχής (γρ. 15).

Στη συνέχεια, θέλουμε να βρούμε το κόστος που έχει ένα σημείο αν θέλαμε να πάμε από το σημείο Α στο σημείο Β, κ.ο.κ., δηλαδή αναζητούμε την μεταξύ τους χιλιομετρική απόσταση (*γρ.* 30-48). Για αυτό το λόγο, δημιουργούμε έναν γράφο G, ο οποίος στον κώδικα είναι ένας δισδιάστατος πίνακας με διπλάσιο μέγεθος του size, δηλαδή είναι επί 2 του size (*γρ.* 32-33), προκειμένου να καλύψουμε την περίπτωση να κινηθούμε από το σημείο Β στο Α και με μηδενική διαγώνιος του πίνακα. Μπορούμε να το φανταστούμε όπως φαίνεται στα αφηρημένα σχήματα παρακάτω (*βλ.* Σχήμα 1. και Πίνακας 1.).



Σχήμα 1: Ένας γράφος όπου τα σημεία Α,Β,Γ και Δ ενώνονται μεταξύ τους

	Α	**Β**	**Γ**	**Δ**
Α	**0**	5	3	7
Β	5	**0**	2	5
Γ	3	2	**0**	5
Δ	7	5	5	**0**

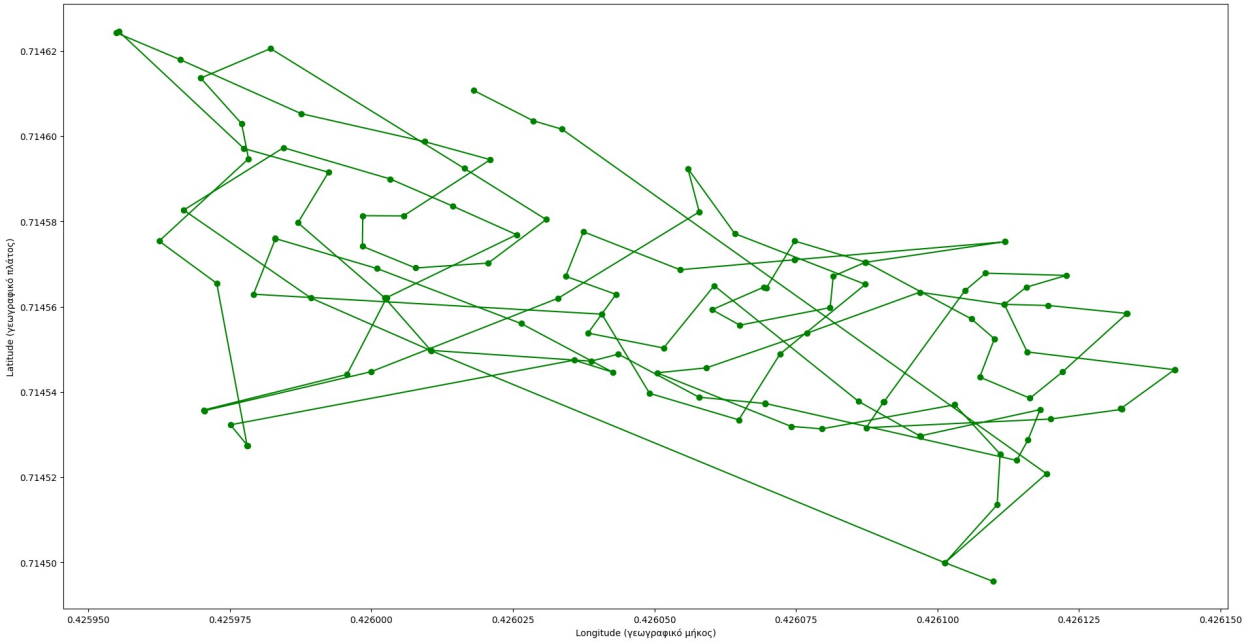
Πίνακας 1. Ένας πίνακας που οι αριθμοί αναπαριστούν το κόστος που χρειάζεται για να πάμε από το ένα σημείο στο άλλο. Για παράδειγμα, για να πάμε από το Α στο Β ή αντίστροφα από το Β στο Α έχουμε το ίδιο κόστος. Σημειώνεται ότι, το κόστος για την μετακίνηση από ένα σημείο στον εαυτό του είναι μηδέν

Έχοντας όλες τις μεταξύ χιλιομετρικές αποστάσεις των σημείων, τρέχουμε το γενετικό αλγόριθμο για τον επιθυμητό αριθμό γενεών και πληθυσμό (*γρ.* 119-120). Ο τρόπος λειτουργίας του αλγορίθμου αναλύεται παρακάτω. Αρχικά, μέσω μίας επανάληψης γίνεται premutation του

αρχικού πληθυσμού για να “ξεφύγουμε” από τα τοπικά μέγιστα (*γρ.* 81-85). Στη συνέχεια, για την τρέχουσα γενιά υπολογίζει την συνολική χιλιομετρική απόσταση (κόστος) κάθε μονοπατιού (*γρ.* 92-94), τα ταξινομεί κατά αύξουσα σειρά (*γρ.* 96) και ελέγχει ποιο μονοπάτι από την τρέχουσα γενιά είναι το μικρότερο προκειμένου να το ορίσει το βέλτιστο μονοπάτι πριν ξεκινήσει την ίδια διαδικασία για τις υπόλοιπες γενιές (*γρ.* 99-102).

Έπειτα, παίρνουμε τυχαία το μισό πληθυσμό και κάνουμε crossover (*γρ.* 105-108), δηλαδή επιλέγουμε δύο τυχαία μονοπάτια (γονείς) και δημιουργούν ένα νέο μονοπάτι (παιδί) μέσω διασταύρωσης των bits των γονέων (*γρ.* 56-68). Ειδικότερα, στον πρώτο γονέα δημιουργείται μια τομή σε ένα τυχαίο σημείο, η σειρά των bits του δεύτερου γονέα μέχρι και την τομή που δίνεται στο παιδί. Έπειτα, ελέγχεται με την σειρά τα bits μέχρι που σειρά των bits είναι διαφορετική από αυτή του πρώτου γονέα και δίνονται στο παιδί. Τα νέα μονοπάτια αντικαθιστούν τον μισό πληθυσμό.

Τέλος, κάθε μονοπάτι έχει 50% πιθανότητα να του γίνει mutation (*γρ.* 111-113), δηλαδή δύο τυχαίες θέσεις στα μονοπάτια των σημείων να εναλλάξουν μεταξύ τους. Η μετάλλαξη που θα γίνει μπορεί να είναι θετική ή αρνητική για την “επιβίωση” του. Αυτή η διαδικασία επαναλαμβάνεται για όλες τις γενιές. Τέλος, ο κώδικας μας εμφανίζει την βέλτιστη λύση (βέλτιστο μονοπάτι) στην οθόνης μας.



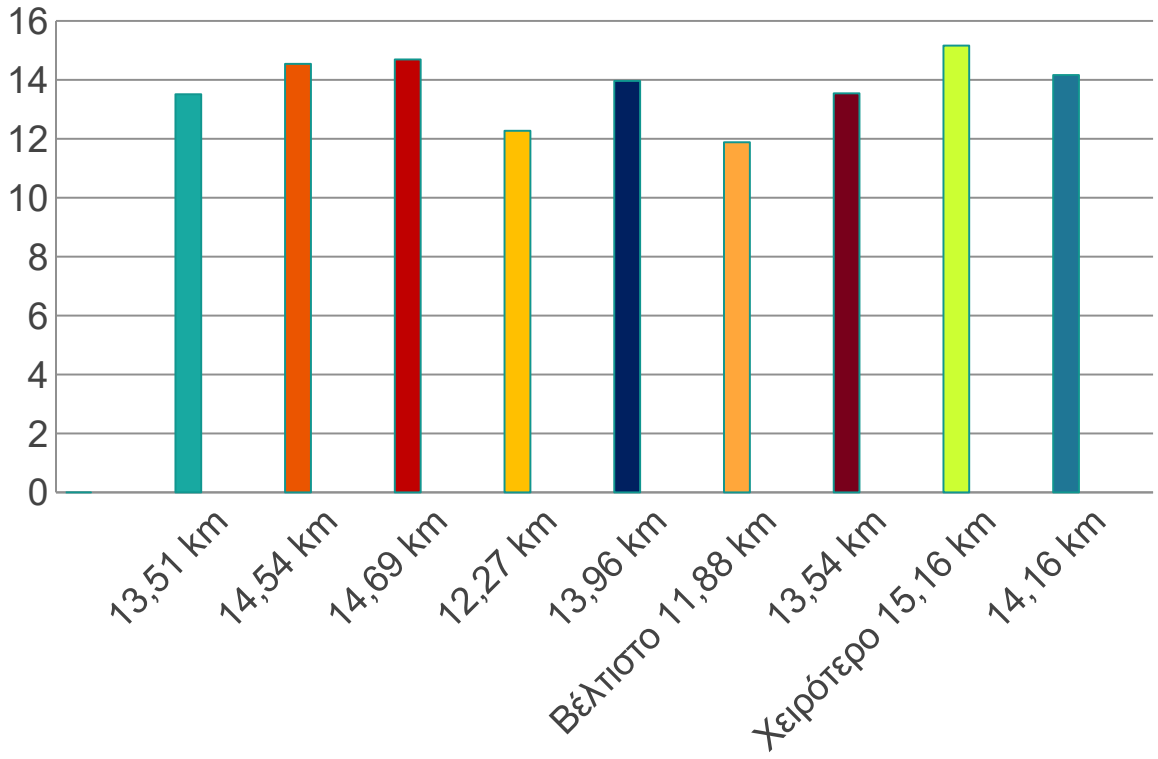
Best Distance: 11.88614932291123 km

With path:

541 540 538 535 532 565 564 525 616 626 624 621 643 543 542 545 544 546 566 529 530 623 622 569 618 620 619 568 567 644 645 531 642 641 572 575 638 637 629 628 625 639 573 571 570 574 631 632 635 636 523 615 630 528 533 534 646 547 536 526 527 562 524 559 556 605 593 586 598 591 592 590 588 582 584 608 607 610 612 613 633 583 587 589 597 599 601 602 551 552 550 558 560 557 611 595 594 603 617 561 563 627 640 576 578 577 614 554 549 548 553 606 634 609 585 596 600 604 555 539 537 579 580 581

Εικόνα 4. Το βέλτιστο μονοπάτι και η καλύτερη απόσταση εμφανίζεται στην οθόνη. Η μικρότερη απόσταση που βγάζει ο αλγόριθμος είναι 11,88 km, η οποία είναι η μικρότερη απόσταση που έχει καταγραφεί από τις δοκιμές που πραγματοποιήθηκαν

Εικόνα



5. Στατιστικά από πολλαπλές δοκιμές του κώδικα

Συμπεράσματα

Το προγραμματιστικό πρόβλημα της εύρεσης της βέλτιστης (μικρότερης) διαδρομής ενός απορριμματοφόρου, στην περιοχή του Αγίου Γεωργίου στην Καβάλα, επιλύθηκε με τη χρήση του γενετικού αλγορίθμου. Ο κώδικας υλοποιήθηκε σε Python ακολουθώντας τη μεθοδολογία επίλυσης του προβλήματος του Περιοδεύοντος Πωλητή. Ο γενετικός αλγόριθμος έχει τη δυνατότητα να εξερευνά αποτελεσματικά έναν μεγάλο χώρο λύσεων και μπορεί να χειρίζεται πολύπλοκα, πολυδιάστατα προβλήματα και αυτό, κατά συνέπεια, τον καθιστά κατάλληλο για το πρόβλημα μας. Αφού βρέθηκαν οι κατάλληλες παράμετροι όσο αναφορά το γενετικό αλγόριθμο, όπως το πλήθος των σημείων, το πλήθος του πληθυσμού κ.ο.κ., γράφτηκαν έξι συναρτήσεις σε Python: readPoints(filename), calcDistances(points, size), CalcPathDistance(G, path), crossover(p1, p2), mutation(path) και genetic_algorithm (G, size, pop_size, gen_limit) προκειμένου να βρεθεί το βέλτιστο μονοπάτι. Η πλειοψηφία των αποτελεσμάτων που βγάζει ο κώδικας είναι ακριβής και ένα μικρό ποσοστό αποτυχίας (βρίσκει μια πάρα πολύ καλή τιμή αλλά όχι τη βέλτιστη δυνατή) που μπορεί να παρατηρηθεί, δικαιολογείται από τη τυχαιότητα του γενετικού αλγορίθμου. Μετά από πολλαπλές δοκιμές έγινε σύγκριση μεταξύ των αποτελεσμάτων και καταγράφηκε το βέλτιστο μονοπάτι, που θα πρέπει να ακολουθήσει το απορριμματοφόρο, για να μεγιστοποιήσει το κόστος και την αποδοτικότητα του. Η σύγκριση αυτή ήταν απαραίτητη λόγω ότι ο αλγόριθμος βρίσκει βέλτιστο μονοπάτι για διαφορετικό αρχικό σημείο σε κάθε εκτέλεση του.

Σημειώνεται ότι, έγινε αρχική απόπειρα επίλυσης του προβλήματα με τη χρήση του αλγορίθμου Multi-Objective NSGA, παρόλα αυτά παρουσιάστηκαν προβλήματα κατά τη συγγραφή και σύνταξη του κώδικα και για αυτό απορρίφτηκε. Επίσης, ένας άλλος ενδεχόμενος τρόπος επίλυσης του προβλήματος θα γίνονταν με τη μέθοδο του Ant Colony Optimization (ACO), στην περίπτωση που ο γενετικός αλγόριθμός εμφάνιζε παρόμοια προβλήματα με το NSGA κατά τη συγγραφή ή την εύρεση μη επιθυμητών αποτελεσμάτων.

Βιβλιογραφία

[1] Chen, C.-H., Liu, T.-K., & Chou, J.-H. (2014). A Novel Crowding Genetic Algorithm and Its Applications to Manufacturing Robots. *IEEE Transactions on Industrial Informatics*, *10*(3), 1705–1716. https://doi.org/10.1109/TII.2014.2316638

[2] Wu, T.-Y., & Lin, C.-H. (2015). Low-SAR Path Discovery by Particle Swarm Optimization Algorithm in Wireless Body Area Networks. *IEEE Sensors Journal*, *15*(2), 928–936. https://doi.org/10.1109/JSEN.2014.2354983

[3] Yoon, Y., & Kim, Y.-H. (2013). An Efficient Genetic Algorithm for Maximum Coverage Deployment in Wireless Sensor Networks. *IEEE Transactions on Cybernetics*, *43*(5), 1473–1483. https://doi.org/10.1109/TCYB.2013.2250955

[4] del Valle, Y., Venayagamoorthy, G. K., Mohagheghi, S., Hernandez, J.-C., & Harley, R. G. (2008). Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. *IEEE Transactions on Evolutionary Computation*, *12*(2), 171–195. https://doi.org/10.1109/TEVC.2007.896686

[5] Ji, B., Song, X., Sciberras, E., Cao, W., Hu, Y., & Pickert, V. (2015). Multiobjective Design Optimization of IGBT Power Modules Considering Power Cycling and Thermal Cycling. *IEEE Transactions on Power Electronics*, *30*(5), 2493–2504. https://doi.org/10.1109/TPEL.2014.2365531

[6] Lin, T.-L., Horng, S.-J., Kao, T.-W., Chen, Y.-H., Run, R.-S., Chen, R.-J., Lai, J.-L., & Kuo, I.-H. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications*, *37*(3), 2629–2636. https://doi.org/10.1016/j.eswa.2009.08.015

[7] Nguyen, S., Zhang, M., Johnston, M., & Tan, K. C. (2015). Automatic Programming via Iterated Local Search for Dynamic Job Shop Scheduling. *IEEE Transactions on Cybernetics*, *45*(1), 1–14. https://doi.org/10.1109/TCYB.2014.2317488

[8] Chernbumroong, S., Cang, S., & Yu, H. (2015). Genetic Algorithm-Based Classifiers Fusion for Multisensor Activity Recognition of Elderly People. *IEEE Journal of Biomedical and Health Informatics*, *19*(1), 282–289. https://doi.org/10.1109/JBHI.2014.2313473

[9] Juang, C.-F. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *34*(2), 997–1006. https://doi.org/10.1109/TSMCB.2003.818557

[10] *Surrogate-assisted clonal selection algorithms for expensive optimization problems | SpringerLink*. (n.d.). Retrieved June 24, 2023, from https://link.springer.com/article/10.1007/s12065-011-0056-1

[11] Jin, Y., & Sendhoff, B. (2009). A systems approach to evolutionary multiobjective structural optimization and beyond. *IEEE Computational Intelligence Magazine*, *4*(3), 62–76. https://doi.org/10.1109/MCI.2009.933094

[12] Jin, Y., Olhofer, M., & Sendhoff, B. (2002). A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, *6*(5), 481–494. https://doi.org/10.1109/TEVC.2002.800884

[13] Zhou, Z., Ong, Y. S., Lim, M. H., & Lee, B. S. (2007). Memetic algorithm using multi-surrogates for computationally expensive optimization problems. *Soft Computing*, *11*(10), 957–971. https://doi.org/10.1007/s00500-006-0145-8

[14] Yang, X. (2021). Genetic Algorithms. In Elsevier eBooks (pp. 91–100). https://doi.org/10.1016/b978-0-12-821986-7.00013-5

[15] Μπαλτζόγλου Δ.–Ε. (n.d.). Γενετικός αλγόριθμος στο πρόβλημα επιλογής μεταβλητών (Διπλωματική Εργασία). https://dspace.lib.ntua.gr/xmlui/bitstream/handle/123456789/42721/%CE%93%CE%95%CE%9D%CE%95%CE%A4%CE%99%CE%9A%CE%9F%CE%A3%20%CE%91%CE%9B%CE%93%CE%9F%CE%A1%CE%99%CE%98%CE%9C%CE%9F%CE%A3.pdf?sequence=1

[16] Roetzel, W., Luo, X., & Chen, D. (20202). Optimal design of heat exchanger networks. In Elsevier eBooks (pp. 231–317). https://doi.org/10.1016/b978-0-12-817894-2.00006-6

[17] The Basics of Genetic Algorithms in Machine Learning. (n.d.). Engineering Education (EngEd) Program | Section. https://www.section.io/engineering-education/the-basics-of-genetic-algorithms-in-ml/

[18] Roetzel, W., Luo, X., & Chen, D. (2020). Optimal design of heat exchanger networks. In Elsevier eBooks (pp. 231–317). https://doi.org/10.1016/b978-0-12-817894-2.00006-6

[19] Καραγιαννίδη, Χ. (2007). Βελτιστοποίηση βιομηχανικής γραμμής παραγωγής με χρήση γενετικών αλγορίθμων (Διπλωματική Εργασία). https://hellanicus.lib.aegean.gr/bitstream/handle/11610/9215/file0.pdf?sequence=1

Παράρτημα – Documentation

Εισαγωγή βιβλιοθηκών:

```
# %%
import matplotlib.pylab as plt
import math
import random as rand
import numpy as np
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET
import pathlib
```

Άνοιγμα και εξαγωγή του XML περιεχομένου

(συνάρτηση readPoints(fileName))**:**

```
def readPoints(fileName):
  #open and parser kml file
  file_path = pathlib.Path(fileName)
  tree = ET.parse(file_path)
  folder = tree.getroot()[0][1]
  names = []
  size = len(folder) - 1
  #points = np.random.randint(-1000, 1000, size * 2)
  points = np.zeros(size * 2)
  points = np.reshape(points, [size, 2])
  for i in range(1, len(folder)):
    node = folder[i]
    names.append(node[0].text)
    tokens = node[2][0].text.split(',')
    long = math.radians(float(tokens[0]))
    lat = math.radians(float(tokens[1]))
    alt = math.radians(float(tokens[2]))
    points[i-1, 0] = long
    points[i-1, 1] = lat
    #print(i-1, long, lat, alt)
  return points, names, size
```

- Δημιουργεί ένα αντικείμενο Path χρησιμοποιώντας pathlib.Path για να αποθηκεύσει το path του αρχείου ως συμβολοσειρά.
- Αναλύει το kml αρχείο χρησιμοποιώντας xml.etree.ElementTree.parse() και αποθηκεύει το δέντρο που προκύπτει στη μεταβλητή tree από το XML περιέρχόμενο του φακέλου.
- Παίρνει το πρώτο στοιχείο XML του φακέλου.

- Αρχικοποιεί μια κενή λίστα names για την αποθήκευση των ονομάτων των σημείων.
- Υπολογίζει τον αριθμό των συνολικών σημείων του φακέλου.
- Δημιουργεί έναν πίνακα μηδενικών points με μέθεγος διπλάσιο του size χρησιμοποιώντας numpy.zeros().
- Αναδιαμορφώνει τον πίνακα points σε έναν δισδιάστατο πίνακα όπου έχουμε γραμμή για το όνομα του σημείου και δύο στήλες για τις συντεταγμένες χρησιμοποιώντας numpy.reshape().
- Στην επανάληψη, γεμίζει τον πίνακα points. Τρέχει για κάθε παιδί του φακέλου και εξάγει το όνομα, το γεωγραφικό πλάτος (latitude), το γεωγραφικό μήκος (longitude) και την υψόμετρο (altitude) για κάθε σημείο. Συγκεκριμένα, αποθηκεύει το όνομα του κάθε σημείου στη λίστα names, μετατρέπει το γεωγραφικό πλάτος, το γεωγραφικό μήκος και το υψόμετρο σε ακτίνες με αντιστόιχηση του π αντί για μοίρες και εκχωρεί μόνο το πλάτος και το μήκος στις κατάλληλες θέσεις στον πίνακα points.
- Επιστρέφει τον πίνακα points, την λίστα names και τον αριθμό των σημείων size

Δημιουργία γράφου G με τις αποστάσεις μεταξύ των σημείων

(συνάρτηση `calcDistances(points, size)`)**:**

```
def calcDistances(points, size):
  G = np.zeros(size * size)
  G = np.reshape(G, [size, size])
  R = 6373.0  #radius of Earth in km
  for i in range(0, size):
    for j in range(i + 1, size):
      long1 = points[i,0]
      lat1 = points[i,1]
      long2 = points[j,0]
      lat2 = points[j,1]
      dlong = abs(long2 - long1)
      dlat = abs(lat2 - lat1)
      a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlong /
2) ** 2
      c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
      d = c * R   #distance between points in km
      G[i,j] = d
      G[j,i] = d
      #print(long1, lat1, long2, lat2, d)
  return G
```

- Δημιουργεί έναν πίνακα μηδενικών G με σχήμα size * size χρησιμοποιώντας numpy.zeros().
- Αναδιαμορφώνει το G σε δισδιάστατο πίνακα με διπλάσιο μέγεθος size με numpy.reshape().
- Αρχικοποιεί την ακτίνα της Γης (R) από το κέντρο της
- Στην επανάληψη, υπολογίζει τη διαφορά του γεωγραφικού μήκους και πλάτους μεταξύ δύο σημείων και υπολογίζει και γεμίζει τον πίνακα με τα βάρη (κόστος/απόσταση) μεταξύ των σημείων.
- Επιστρέφει τον γράφο G

Υπολογισμός των μονοπατιών

(συνάρτηση `calcPathDistance(G, path)`)**:**

```
def calcPathDistance(G, path):
  dist = 0
  for j in range(len(path) - 1):
    dist += G[int(path[j]), int(path[j+1])]
  return dist
```

- Προσθέτει τις μεταξύ αποστάσεις των σημείων
- Επιστρέφει τον πίνακα αποστάσεων με τα συνολικές αποστάσεις

Διασταύρωση:

(συνάρτηση `calcPathDistance(G, path)`)**:**

```
def crossover(p1, p2):
  cut = rand.randrange(1, len(p1) - 2)
  child = np.zeros(len(p1))
  S = {}
  for j in range(cut):
    child[j] = p1[j]
    S[p1[j]] = 1
  for j in range(len(p2)):
    if not p2[j] in S:
      child[cut] = p2[j]
      S[p2[j]] = 1
      cut += 1
  return child
```

- Επιλέγει τυχαία ένα σημείο τομής (cut) εντός του πρώτου μονοπατιού
- Δημιουργεί ένα απόγονο (child) του ίδιου μήκους με το γονικό μονοπάτι
- Αντιγράφει τα πρώτα στοιχεία από το p1 στο child μέρχι το σημείο τομής
- Επαναλαμβάνει τα στοιχεία του p2 και προσθέτει αυτά που δεν είναι ήδη στο child
- Επιστρέφει τον απόγονο

Μετάλλαξη

(συνάρτηση `calcPathDistance(G, path)`):

```
#bitswap
def mutation(path):
  r1 = rand.randrange(len(path))
  r2 = rand.randrange(len(path))
  temp = path[r1]
  path[r1] = path[r2]
  path[r2] = temp
  return path
```

- Επιλέγει τυχαία δύο θέσεις (r1και r2) από
- Ανταλλάσσει τα bits στις θέσεις r1και r2 στο μονοπάτι
- Επιστρέφει το μονοπάτι

Υλοποίηση Γενετικού Αλγόριθμου

(συνάρτηση `calcPathDistance(G, path)`):

```
def genetic_algorithm(G, size, pop_size, gen_limit):
  next_pop_limit = math.floor(pop_size / 2)
  #initialize population
  paths = []
  for i in range(pop_size):
    path = np.random.permutation(size)
    paths.append(path)

  gen = 1
  bestPath = []
  bestDistance = float('inf')
  while gen <= gen_limit:
    #calculate path distance
    distances = np.zeros(pop_size)
```

```
    for i in range(len(paths)):
      distances[i] = calcPathDistance(G, paths[i])
    #sort distance ascended
    paths, distances = (list(x) for x in zip(*sorted(zip(paths,distances), key=lambda
pair:pair[1])))
    #choose best solutions
    if bestDistance > distances[0]:
      bestDistance = distances[0]
      bestPath = paths[0]
    gen += 1

    #crossover
    for i in range(next_pop_limit, pop_size):
      p1 = paths[rand.randrange(next_pop_limit)]
      p2 = paths[rand.randrange(next_pop_limit)]
      paths[i] = crossover(p1, p2)

    #mutation
    for i in range(next_pop_limit, pop_size):
      if rand.random() < 0.5:
        paths[i] = mutation(paths[i])

  return bestPath, bestDistance, gen
```

- Υπολογίζει το επόμενο όριο του πληθυσμού (next_pop_limit) ως το ήμισυ του αρχικού.
- Αρχικοποιεί μια κενή λίστα paths.
- Επαναλαμβάνει για τον pop_size και κάνει permutation για κάθε μονοπάτι.
- Αρχικοποιεί τις μεταβλητές: gen ως 1 για την τρέχον γενιά, bestPath ως κενή λίστα και bestDistance ως θετικό άπειρο.
- Εισάγει έναν while βρόχο, όπου: αρχικοποιεί έναν μηδενικό πίνακα distances
- Υπολογίζει τις αποστάσεις όλων των μονοπατιών στον πληθυσμό χρησιμοποιώντας την calcPathDistance συνάρτηση.
- Ταξινομεί και τους πίνακες paths και distances σε αύξουσα σειρά με βάση τις αποστάσεις.
- Εάν η καλύτερη απόσταση σε αυτήν τη γενιά (distances[0]), τότε είναι καλύτερη από την τρέχουσα καλύτερη απόσταση και ενημερώνει την καλύτερη απόσταση και το καλύτερο μονοπάτι.
- Αυξάνει τον μετρητή γενιάς.
- Επιλέγει από το μισό πληθυσμό δύο τυχαία μονοπάτια (p1 και p2) για διασταύρωση.
- Με πιθανότητα 50%, επιλέγει το μονοπάτι για μετάλλαξη από το μισό πληθυσμό.
- Επιστρέφει το καλύτερο μονοπάτι, την καλύτερη απόσταση και τον αριθμό των γενεών.

Κάλεσμα συναρτήσεων:

```
points, names, size = readPoints(r'Άγιος Γεώργιος.kml')
G = calcDistances(points, size)
gen_limit = 500
pop_size = 200
print("Running for", size ,"points")
print("Generations:", gen_limit)
print("Population:", pop_size)
print("Please wait...")
bestPath, bestDistance, gens = genetic_algorithm(G, size, pop_size, gen_limit)

print("Best Distance:", bestDistance, "km")  #fitness
print("With path: ")
for i in range(len(bestPath)):
  print(names[int(bestPath[i])], end = ' ')

print()

long = np.zeros(len(bestPath))
lat = np.zeros(len(bestPath))

for i in range(0, len(bestPath)):
    long[i] = points[int(bestPath[i]),0]
    lat[i] = points[int(bestPath[i]),1]

plt.figure(figsize=(23,12))
plt.plot(long, lat, '-o', c='green')
#plt.scatter(long,lat)
plt.xlabel('Longitude (γεωγραφικό μήκος)')
plt.ylabel('Latitude (γεωγραφικό πλάτος)')
#plt.grid(True)
plt.show

# %%
```

- Καλεί τη συνάρτηση readPoints για να αποκτήσει τα σημεία, τα ονόματα και τον αριθμό των σημείων
- Καλεί τη συνάρτηση calcDistances για να υπολογίσει τον πίνακα απόστασης
- Ορίζει το γενίες σε 500 και το μέγεθος του πληθυσμού σε 200
- Καλεί τη συνάρτηση genetic_algorithm με τον πίνακα αποστάσεων, τον αριθμό των σημείων, το μέγεθος του πληθυσμού και τον αριθμό των γενεών
- Εμφανίζει την καλύτερη απόσταση και το αντίστοιχο μονοπάτι

- Δημιουργεί δύο πίνακες με μέγεθος όσα τα στοιχεία του καλύτερου μονοπατιού
- Στην επανάληψη, γεμίζει τους πίνακες τις συντεταγμένες x και y κάθε σημείου
- Σχεδιάζει ένα plot με το καλύτερο μονοπάτι