Πανεπιστήμιο Πειραιώς 2017-2018

Τμήμα Πληροφορικής

Τεκμηρίωση Κώδικα Απαλλακτικής Εργασίας

Credits: Τριτσαρώλης Ανδρέας (Π15143), **University Of Piraeus - 2017-2018**

Contents

Μέρος Α' - Εισαγωγή στο Πρόβλημα του Περιοδεύοντος Πωλητή

Page: 03

Μέρος Β' - Ο Γενετικός Αλγόριθμος

1. Εισαγωγή Page: 03

2. Πληθυσμός (Population) Page: 04

3. Συνάρτηση Καταλληλότητας (Fitness Function) Page: 04

4. Επιλογή Γονέων (Selection) Page: 04

5. Δημιουργία Απογόνων (Crossover) Page: 04

6. Μετάλλαξη Γονιδιωμάτων (Mutation) Page: 05

7. Ελιτισμός Γονιδιωμάτων (Elitism) Page: 05

8. Σύγκλιση Αλγόριθμου (Algorithm Convergence) Page: 05

Μέρος Γ' - Υλοπόιηση Κώδικα

1. Κλάση Γράφου Page: 06

2. Κλάση Γενετικού Αλγόριθμου Page: 06

3. Εκτέλεση Κώδικα Page: 07

4. Παράδειγμα Εκτέλεσης Page: 08

Μέρος Α' - Εισαγωγή στο Πρόβλημα του Περιοδεύοντος Πωλητή

Το Πρόβλημα του Περιοδεύοντος Πωλητή (Travelling Salesman Problem – TSP) είναι κατ'ουσίαν, η εύρεση του ελάχιστου μονοπατιού που διασχίζει όλες τις πόλεις μονάχα μια φορά ουτώς ώστε ένας πωλητής να περάσει από αυτές με το ελάχιστο – δυνατό – κόστος. Η λύση σε αυτό το – κλασσικό στους αλγόριθμους – πρόβλημα μπορεί να επιτευχθεί μεσω μιας πληθώρας τεχνικών (π.χ: *Άπληστοι Αλγόριθμοι, Brute Force, Δυναμικός Προγραμματισμός, ...*). Ωστόσο αυτές οι τεχνικές "πάσχουν" από ένα πρόβλημα: **Ο χώρος αναζήτησης αυξάνεται εκθετικά όσο αυξάνεται το πλήθος των πόλεων.**

Με άλλα λόγια οι παραπάνω – παραδοσιακές – τεχνικές δεν μπορούν να χρησιμοποιηθούν για να λύσουν το TSP επειδή χρειάζονται χρόνο εκθετικής τάξης – ανέφικτο σε πραγματικές εφαρμογές.

Σε αυτό το σημείο, οι Γενετικοί Αλγόριθμοι χρησιμοποιούνται για να παρέχουν – προσεγγιστικά – βέλτιστες λύσεις σε εφικτό χρόνο.

Μέρος Β' - Ο Γενετικός Αλγόριθμος

1. Εισαγωγή

Οι Γενετικοί Αλγόριθμοι αποτελούν μέλος της οικογένειας των Πιθανοκρατικών Αλγορίθμων, εμπνευσμένοι από την Δαρβίνεια Θεωρία της Εξέλιξης.

Αναζητούν στο χώρο αναζήτησης του προβλήματος βρίσκοντας (σύμφωνα με κάποια κριτήρια) εφικτές λύσεις.

Για να χρησιμοποιήσει κανείς ένα Γενετικό Αλγόριθμο, θα πρέπει να ορίσει τα εξής:

- Πληθυσμό **(Population)**
- Συνάρτηση Καταλληλότητας **(Fitness Function)**
- Επιλογή Γονέων **(Selection)**
- Συνάρτηση Αναπαραγωγής **(Crossover)**
- Συνάρτηση Μετάλλαξης **(Mutation)**
- Ελιτισμό Γονιδιωμάτων **(Elitism)**

2. *Πληθυσμός (Population)*

Στο TSP επιλέγουμε να κωδικοποιήσουμε τον πληθυσμό ως μια λίστα από συμβολοσειρες (γονιδίωμα), όπου κάθε γράμμα (άλληλο) είναι ένας κόμβος του γράφου *(μια πόλη στον χάρτη).*

***Παραδείγματος Χάρη*:** Για ένα γράφο με κόμβους V = {a, b, c, d, e}, ένα άλληλο είναι ένα από τα στοιχεία στο V, ένα γονίδιωμα είναι κάθε μετάθεση των στοιχείων του V και ένας πληθυσμός είναι μια λίστα από N γονιδιώματα.

3. Συνάρτηση Καταλληλότητας (Fitness Function)

Στο παρών πρόβλημα, ως συνάρτηση καταλληλότητας χρησιμοποιείται το άθροισμα των βαρών στις ακμές των κόμβων στον γράφο για ένα συγκεκριμένο μονοπάτι *(το μονοπάτι είναι κωδικοποιημένο στο γονιδίωμα – μια μετάθεση των στοιχείων του V).*

4. Επιλογή Γονέων (Selection)

Κάθε γενιά θα πρέπει να αποτελείται από N γονιδιώματα τη φορά. Για να επιτευχθεί η εξέλιξη αυτών θα πρέπει να επιλεχθούν κάποια γονίδια *(με κάποια Συνάρτηση Επιλογής – θα αναφερθεί στην συνέχεια),* να αναπαραχθούν *(με κάποια Συνάρτηση Αναπαραγωγής – θα αναφερθεί στην συνέχεια)* και να παράγουν απογόνους, οι οποίοι θα είναι μέλη της επόμενης γενιάς.

Για να επιλεχθουν αυτά τα γονίδια έχουν συζητηθεί πολλές τεχνικές (π.χ: Roulette Selection) αλλά η τεχνική που θα χρησιμοποιηθεί σε αυτή την υλοποίηση είναι η τεχνική Tournament Selection, η οποία λειτουργεί ως εξής:

> **choose K genomes from the population**

> **for each genome chosen:**
calculate its fitness with a Fitness function

> **choose the fittest genome among the K genomes**

5. Δημιουργία Απογόνων (Crossover)

Τώρα που έχουμε τα γονιδιώματα – γονείς από το στάδιο του Selection σειρά έχει η δημιουργία απογόνων (crossover) για την παραγωγή – δυνητικά – καλύτερων λύσεων. Όπως και στο προηγούμενο στάδιο έτσι και εδώ υπάρχει μια πληθώρα επιλογών, ωστόσο αυτή που επιλέγουμε είναι η εξής:

> **Select a subset from the first parent.**

> **Add that subset to the offspring.**

> **Any missing values are then added to the offspring from the second parent in order that they are found.**

Example:
Parent 1: a,[b, c],d, e
Parent 2: b, e, a, c, d
Offspring: e, b, c, a, d

6. Μετάλλαξη Γονιδιωμάτων (Mutation)

Μετά την δημιουργία των απογόνων, για την οριστικοποίηση του νέου πληθυσμού θα πρέπει να μεταλλάξουμε τα (νέα) γονιδιώματα (με κάποια πιθανότητα μετάλλαξης – mutationRate).

Για την μετάλλαξη ενός γονιδιώματος απλά επιλέγουμε δυο τυχαία άλληλα p, q με p ≠ q και τα μεταθέτουμε (swap) μεταξύ τους.

7. Ελιτισμός Γονιδιωμάτων (Elitism)

Στους γενετικούς αλγόριθμους *(ενίοτε)* τα ισχυρότερα γονιδιώματα της τωρινής γενιάς περνούν αυτούσια στην επόμενη, με σκοπό την "αναμέτρηση" με τους απογόνους και την αναπαραγωγή σε – δυνητικά – ισχυρότερα γονιδιώματα και κατ' επέκταση, σε καλύτερες λύσεις.

Το βήμα αυτό λαμβάνει χώρα κατα τον υπολογισμό της συνάρτησης καταλληλότητας για κάθε γονιδίωμα. Όταν όλες οι τιμές έχουν υπολογιστεί, τα γονιδιώματα με την μικρότερη τιμή περνούν κατευθείαν στον πληθυσμό της επόμενης γενιάς.

8. Σύγκλιση Αλγόριθμου (Algorithm Convergence)

Ο Αλγόριθμος σταματά (συγκλίνει) αν:

- Όλα τα γονιδιώματα του πληθυσμού είναι ίδια **ή**
- Αν ο αλγόριθμος έχει φθασει το μέγιστο επιτρεπόμενο πλήθος γενεών

Οι παραπάνω περιπτώσεις υποδεικνύουν πως ο αλγόριθμος έχει συγκλίνει σε ένα ελάχιστο (ολικό ή τοπικό).

Σημείωση: Για την διατήρηση της συνέπειας του αλγόριθμου προτείνεται να υπάρχει ένας αρκούντως μεγάλος πληθυσμός και ποσοστό μετάλλαξης και ελιτισμού.

ΠΡΟΣΟΧΗ: Δίνοντας πολύ μεγάλες τιμές στο ποσοστό μετάλλαξης αυξάνει την πιθανότητα αναζήτησης περισσότερων περιοχών στο χώρο αναζήτησης, εμποδίζοντας ωστόσο τον αλγόριθμο από το να συγκλίνει σε μια βέλτιστη λύση. [Πηγή] Ομοίως, δίνοντας πολύ μεγάλες τιμές στο ποσοστό ελιτισμού μειώνει την πιθανότητα την πιθανότητα αναζήτησης περισσότερων περιοχών στο χώρο αναζήτησης, βοηθώντας τον αλγόριθμο να συγκλίνει ταχύτερα σε μια λύση, αλλά (πιθανότατα) δεν θα είναι η βέλτιστη *(με άλλα λόγια, αυξάνεται το ρίσκο να εγκλωβιστεί ο αλγόριθμος σε ένα τοπικό ελάχιστο).*

Μέρος Γ' - Υλοποίηση Κώδικα

1. Κλάση Γράφου

Για την υλοποίηση του TSP χρησιμοποιούνται δύο κλάσεις:

- **Graph (Graph2.py):** Διατηρεί πληροφορίες και οδηγίες σχετικά με τον γράφο προς υπολογισμό. Είναι υπεύθυνη για:
  - Την Δημιουργία ενός γράφου
  - Την Προσθήκη ενός κόμβου (και των δεσμών του με τους άλλους κόμβους ενός γράφου)
  - Την Λήψη πληροφορίας για έναν κόμβο του γράφου
  - Την Λήψη πληροφορίας για τους δεσμούς ενός κόμβου στο γράφο

  Οι μέθοδοι που καθιστούν τα παραπάνω δυνατά είναι τα εξής:
  - **__init__**(graph={}) *(graph: The Graph Dictionary **{'vertex':{'adjacent':edge_weight}}**)*
  - **setVertex**(vertex) *(given a vertex name – in string format – set up entry in graph dictionary – if it doesn't exist)*
  - **setAdjacent**(vertex, adj, weight=0) *(given a node – and it's adjacent – along with the edge's weight set up an entry in the graph dictionary; – if they do not exist already – then set the weight like the pattern stated in the __init__ function)*
  - **getVertices**() *(return all nodes on the graph in a list of strings)*
  - **getAdjacent**(vertex) *(given a vertex name return all its adjacent nodes)*
  - **getPathCost**(path) *(given a path – in string format e.g: 'abcde' - return it's cost as the sum of the vertex weights in the path)*

2. Κλάση Γενετικού Αλγόριθμου

- **Genetic Algorithm (GeneticAlgorithmTSP.py):** Διατηρεί πληροφορίες και οδηγίες σχετικά με την λύση του TSP με την βοήθεια των γενετικών αλγορίθμων. Είναι υπεύθυνη για:
  - Την αρχικοποίηση του προβλήματος, δοθέντων ορισμένων παραμέτρων
  - Την εκκίνηση της λύσης του προβλήματος και την επιστροφή της λύσης.

  Οι μέθοδοι που καθιστούν τα παραπάνω δυνατά είναι τα εξής:
  - __init__(self, generations=100, population_size=10, tournamentSize=4, mutationRate=0.1, elitismRate=0.1), **όπου:**
    - **generations**: Ο μέγιστος επιτρεπτός αριθμός εξελίξεων του πληθυσμού
    - **population_size**: Το μέγεθος του πληθυσμού
    - **tournamentSize**: Το μέγεθος των "διαγωνιζόμενων" γονιδιωμάτων στο στάδιο της επιλογής γονέων μεσω Tournament Selection
    - **mutationRate:** Ο ρυθμός μετάλλαξης *(το ποσοστό μετάλλαξης)* των – μη ελιτιστικών – j γονιδιωμάτων του πληθυσμού
    - **elitismRate:** Το ποσοστό ελιτισμού γονιδιωμάτων του πληθυσμού *(ποσο % του πληθυσμού θα ελιτιστεί)*
  - optimize(self, graph) *(graph: The Graph Dictionary **{'vertex':{'adjacent': edge_weight}}**)*

3. Εκτέλεση Κώδικα

Σημείωση: Τα scripts που αναφέρθηκαν στη προηγούμενη ενότητα έχουν δοκιμαστεί με Python 3.5.4 μέσω του περιβάλλοντος Anaconda3.

Code Depedencies: NumPy *(sudo pip3 install numpy)*

Για την εκτέλεση του γενετικού αλγόριθμου που επιλύει το TSP εκτελείτε την εξής εντολή:

python GeneticAlgorithmTSP.py

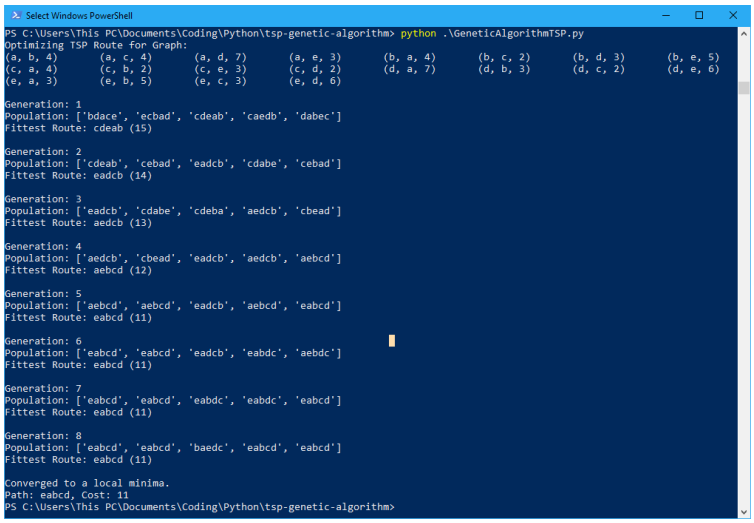
Το οποίο επιλύει το εν λόγω πρόβλημα σύμφωνα με την παρακάτω λογική:

```
> current_population = Initiate (pseudo)random population with N genomes
> elitismOffset = ceil(population_size * elitismRate)
> for every generation:
>    newPopulation = []
>    calculate fitness value for every genome in the current_population
>    newPopulation.append(elitismOffset fittest genomes)
>    for genome in population[elitismOffset:population_size):
>         select 2 parents with TournamentSelection
>         newPopulation.append(offspring)
>    for genome in population[elitismOffset:population_size):
>         select a random_number between 0 and 1
>         if random_number <= mutationRate:
>              mutate genome
>    current_population = new_population
>    if all genomes are the same:
>         break
> return fittest(current_population)
```

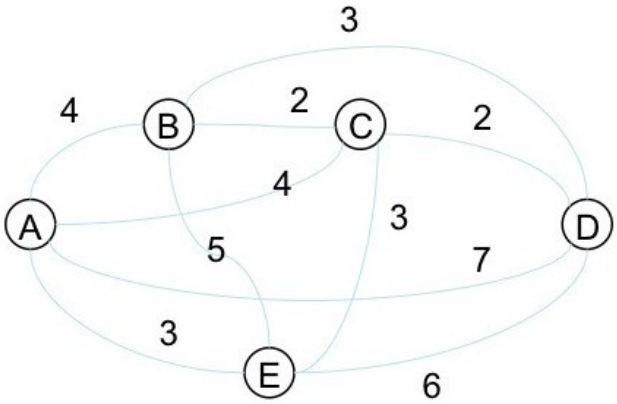
*(**Note**: to initiate the TSP for another graph or fine-tune the genetic algorithm's parameters, please change the code @__main__ function inside the GeneticAlgorithmTSP.py script.)*

4. Παράδειγμα Εκτέλεσης

Ένα στιγμιότυπο της εκτέλεσης που επιλύει το TSP για τον παρακάτω γράφο φαίνεται στο επόμενο screenshot



Εικόνα Γ.4.1 *– Στιγμιότυπο Εκτέλεσης*



Εικόνα Γ.4.2 *– Ο Γράφος τον οποίο ο Γενετικός Αλγόριθμος καλείται να λύσει*

Με παραμέτρους:

- **generations** = 20
- **population_size** = 7
- **tournamentSize** = 2
- **mutationRate** = 0.2
- **elitismRate** = 0.1

Όπως φαίνεται και στην εικόνα παρά το γεγονός ότι ο αλγόριθμος θα έτρεχε για 20 γενιές, σταμάτησε στις 8 διότι τα γονίδια του **νέου πληθυσμού** είναι ίδια μεταξύ τους, συνεπώς **ο αλγόριθμος έχει συγκλίνει σε ένα ελάχιστο, την διαδρομή *eabcd με κόστος 11***.

ΣΗΜΕΙΩΣΗ: Ο πληθυσμός που φαίνεται στην οθόνη είναι ο αμέσος προηγούμενος από αυτόν που υπολογίζεται και κατά συνέπεια όταν τα γονίδια εξισωθούν ο βρόγχος λύεται και γίνονται ορατά μόνο από τη “βέλτιστη” λύση που επιστρέφεται..