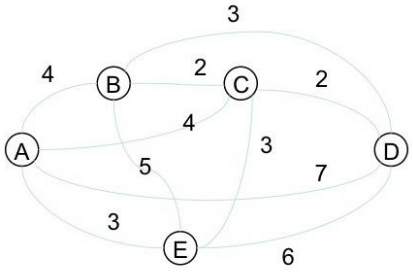
Το πρόβλημα του πλανόδιου πωλητή: Μια υλοποίηση με τη χρήση γενετικού αλγορίθμου σε Python

Αναστασία Μπουμπλίνη – Π19117

1. Περιγραφή Προβλήματος

Το πρόβλημα του πλανόδιου πωλητή πρόκειται για ένα από τα διασημότερα προβλήματα στην επιστήμη της πληροφορικής και ανάγεται στην εύρεση της σειράς με την οποία ένας πωλητής πρέπει να περάσει από όλες τις πόλεις ενός συνόλου πόλεων και να επιστρέψει στην αρχική, ώστε να έχει το μικρότερο δυνατό κόστος. Στην συγκεκριμένη υλοποίηση, το κόστος εκφράζει την απόσταση που απέχουν οι πόλεις μεταξύ τους. Η προσέγγιση που έχει γίνει στο πρόβλημα βασίζεται στο παρακάτω γράφημα, το οποίο αποτελείται από πλήρως διασυνδεδεμένες πόλεις. Επιπλέον, για τις ανάγκες του αλγορίθμου γίνεται χρήση των βιβλιοθηκών NumPy και random της python.



2. Δεδομένα προβλήματος

Η επεξήγηση του τρόπου δράσης του αλγορίθμου θα γίνει σταδιακά και αποσπασματικά, επομένως σε κάθε παράγραφο θα γίνεται λόγος για ένα συγκεκριμένο κομμάτι, ξεκινώντας από την εισαγωγή των δεδομένων στον αλγόριθμο. Αρχικά, όπως φαίνεται και στο παραπάνω σχήμα, υπάρχουν πέντε πόλεις, οι οποίες αρχικοποιούνται σε έναν πίνακα της NumPy. Επιπλέον, για δική μας διευκόλυνση θεωρούμε πως οι πόλεις έχουν την παρακάτω αντιστοιχία σε αριθμούς.

A	0
B	1
C	2
D	3
E	4

Κομβικής σημασίας λειτουργία, είναι να μπορεί να υπολογιστεί η απόσταση μεταξύ δυο πόλεων και κατ' επέκταση η συνολική απόσταση μιας διαδρομής του πωλητή. Αυτή τη λειτουργία την παρέχει η μέθοδος compute_score(), η οποία δέχεται ως ορίσματα τους αριθμούς που αντιστοιχούν σε δύο πόλεις και επιστρέφει την απόσταση τους, χρησιμοποιώντας έναν πίνακα κόστους, ο οποίος δημιουργήθηκε με βάση το γράφημα της πρώτης παραγράφου. Τέλος, όσον αφορά την αρχικοποίηση του αρχικού πληθυσμού λύσεων επιλέχθηκε, αυθαίρετα, η τιμή 10.

3. Δημιουργία και αναπαράσταση λύσεων

Για την σωστή δράση του αλγορίθμου, απαιτείται να υπάρχει ένα αρχικό σύνολο υποψήφιων λύσεων. Για αυτό το σκοπό έχει δημιουργηθεί η μέθοδος create_population_set, η οποία δέχεται ως ορίσματα τα ονόματα των πόλεων (A, B, C, D, E) και το πλήθος του αρχικού πληθυσμού λύσεων (10) και επιστρέφει το αρχικό σύνολο λύσεων ως πίνακα της NumPy. Η μέθοδος, αυτή, καλείται μόνο κατά την πρώτη επανάληψη του γενετικού αλγορίθμου, καθώς από την δεύτερη επανάληψη και μετά, κάθε σύνολο υποψήφιων λύσεων προκύπτει από την διαδικασία αναπαραγωγής, η οποία θα επεξηγηθεί σε επόμενη παράγραφο.

4. Συνάρτηση Καταλληλότητας

Εφόσον έχει δημιουργηθεί το αρχικό πλήθος υποψήφιων λύσεων, χρειαζόμαστε μια μέθοδο η οποία θα καθορίσει την καταλληλότητα κάθε λύσης σε κάθε πλήθος λύσεων, που θα προκύψει, στην πορεία εκτέλεσης του γενετικού αλγορίθμου. Με τον όρο καταλληλότητα εννοείται πως όσο μικρότερη είναι η συνολική απόσταση μιας διαδρομής τόσο πιο κατάλληλη και ανώτερη θεωρείται η υποψήφια λύση. Συγκεκριμένα για τον υπολογισμό

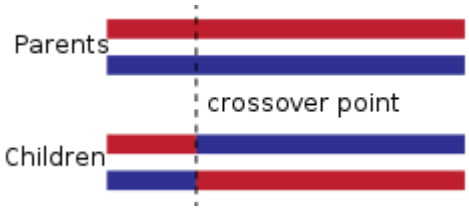
της συνολικής απόστασης μεταξύ των πόλεων που περιέχονται σε μια λύση χρησιμοποιείται η μέθοδος fitness_function(), η οποία δέχεται ως όρισμα το σύνολο των υποψήφιων λύσεων και επιστρέφει έναν μονοδιάστατο πίνακα με τις συνολικές αποστάσεις όλων των λύσεων. Στην συνέχεια, για την αξιολόγηση των λύσεων χρησιμοποιείται η μέθοδος evaluation(), η οποία δέχεται ως όρισμα τον πίνακα των αποστάσεων, που αναφέρθηκε παραπάνω και επιστρέφει έναν νέο πίνακα πιθανότητας, του οποίου το κάθε στοιχείο αντιπροσωπεύει την πιθανότητα να επιλεχθεί μια συγκεκριμένη λύση ως γονέας.

5. Επιλογή Γονέων

Έχουμε φτάσει στο βήμα, στο το οποίο πρέπει να επιλεχθούν οι γονείς, δηλαδή εκείνες οι υποψήφιες λύσεις που θα δημιουργήσουν την επόμενη γενιά, με την διαδικασία της αναπαραγωγής. Η μέθοδος parent_selection() επιλέγει οκτώ γονείς, χρησιμοποιώντας την τεχνική της ρουλέτας, με κριτήριο την πιθανότητα να επιλεχθεί κάθε λύση. Ο πίνακας NumPy που επιστρέφεται από την parent_selection() θα χρησιμοποιηθεί για την δημιουργία απογόνων με την τεχνική της διασταύρωσης ενός σημείου.

6. Αναπαραγωγή

Η αναπαραγωγή είναι η διαδικασία δημιουργίας απογόνων, δηλαδή είναι η δημιουργία της επόμενης γενιάς υποψήφιων λύσεων. Στην, εν λόγω, προσέγγιση του προβλήματος έχει χρησιμοποιηθεί η τεχνική της διασταύρωσης ενός σημείου (single point crossover). Γενικά, στη διασταύρωση ενός σημείου επιλέγεται ένα τυχαίο σημείο στους γονείς, το οποίο ονομάζεται «σημείο διασταύρωσης» και τα στοιχεία δεξιά αυτού του σημείου ανταλλάσσονται μεταξύ των δύο γονέων. Αυτό έχει ως αποτέλεσμα την δημιουργία δύο απογόνων, όπου ο καθένας φέρει πληροφορίες και από τους δύο γονείς, όπως φαίνεται και στην παρακάτω εικόνα.



Η παραπάνω διαδικασία, αν και πολύ αποτελεσματική, δεν θα λειτουργήσει για το συγκεκριμένο πρόβλημα, καθώς υπάρχει ο περιορισμός ότι τα ονόματα των πόλεων δεν μπορούν να επαναληφθούν. Επομένως, βασισμένοι στην διασταύρωση ενός σημείου, θα ακολουθήσουμε μια παρόμοια διαδικασία, της οποίας τα βήματα περιγράφονται παρακάτω:

1. Ορίζεται, αυθαίρετα, το σημείο διασταύρωσης και στους δυο γονείς.
2. Ο πρώτος γονέας χωρίζεται σε δύο κομμάτια.
3. Διαγράφονται από τον δεύτερο γονέα τα κοινά στοιχεία του δεύτερου γονέα με πρώτο κομμάτι του πρώτου γονέα.
4. Το πρώτο κομμάτι του πρώτου γονέα ενώνεται με τα στοιχεία που έχουν απομείνει από τον δεύτερο γονέα και, έτσι, έχει δημιουργηθεί ο πρώτος απόγονος.

Για την δημιουργία του δεύτερου απογόνου ακολουθείται η ίδια διαδικασία αντιστρέφοντας τους γονείς. Η μέθοδος crossover() εκτελεί τα παραπάνω βήματα ανα δύο γονείς και δημιουργεί, κάθε φορά, δύο απογόνους. Δέχεται ως όρισμα τον πίνακα με τους γονείς και επιστρέφει έναν νέο πίνακα, που περιλαμβάνει τους απογόνους, δηλαδή την επόμενη γενιά υποψήφιων λύσεων.

7. Μερική Ανανέωση Πληθυσμού

Η μερική ανανέωση πληθυσμού είναι μια διαδικασία, κατά την οποία επιλέγεται, τυχαία, ένα ποσοστό λύσεων της προηγούμενης γενιάς, το οποίο θα περάσει αυτούσιο και χωρίς καμία αλλαγή στην επόμενη γενιά. Αξίζει να σημειωθεί ότι μια λύση μπορεί να επιλεχθεί για γονέας και, ταυτόχρονα, να έχει επιλεχθεί για μερική ανανέωση πληθυσμού. Οι μέθοδοι renewal_elements() και population_renewal() είναι υπεύθυνες για μια μερική ανανέωση της νέας γενιάς υποψήφιων λύσεων σε ένα ποσοστό 20%. Αρχικά, πριν την επιλογή γονέων επιλέγονται, με τη χρήση της renewal_elements(), δύο τυχαίες υποψήφιες λύσεις και στη συνέχεια, εφόσον έχει γίνει η αναπαραγωγή και η δημιουργία απογόνων, αυτές οι λύσεις προστίθενται στην επόμενη γενιά, με τη χρήση της population_renewal().

8. Μετάλλαξη

Η μετάλλαξη του πληθυσμού είναι το τελευταίο στάδιο του γενετικού αλγορίθμου και ορίζεται ως η εναλλαγή δύο στοιχείων μιας υποψήφιας λύσης. Για αυτό το σκοπό έχουν δημιουργηθεί οι μέθοδοι mutate_child() και mutation(). Συγκεκριμένα η μέθοδος mutation(), για κάθε απόγονο του νέου πληθυσμού υποψήφιων λύσεων, καλεί την mutate_child(), η οποία ορίζει τυχαία δύο πόλεις, που θα μεταλλαχθούν.

9. Γενετικός Αλγόριθμος

Όπως αναφέρθηκε και σε προηγούμενη παράγραφο, ο γενετικός αλγόριθμος ανάγεται στην εύρεση της σειράς με την οποία ένας πωλητής πρέπει να περάσει από όλες τις πόλεις ενός συνόλου πόλεων και να επιστρέψει στην αρχική, ώστε να έχει το μικρότερο δυνατό κόστος. Αρχικά ο αλγόριθμος εκτελείται είκοσι φορές, αποθηκεύοντας σε κάθε επανάληψη τα χαρακτηριστικά της, μέχρι στιγμής, καλύτερης λύσης. Συνδυάζοντας όλα τα παραπάνω ο τρόπος δράσης του αλγορίθμου περιγράφεται ως ακολούθως:

1. Δημιουργείται η πρώτη γενιά υποψήφιων λύσεων
2. Εκτελείται η συνάρτηση καταλληλότητας
3. Επιλέγονται οι γονείς
4. Επιλέγονται οι υποψήφιες λύσεις που θα κάνουν την μερική ανανέωση πληθυσμού
5. Γίνεται αναπαραγωγή και δημιουργούνται οι απόγονοι
6. Γίνεται μερική ανανέωση πληθυσμού
7. Γίνεται μετάλλαξη και, πλέον έχει δημιουργηθεί η επόμενη γενιά
8. Επανάληψη διαδικασίας από το βήμα 2 μέχρι να ολοκληρωθούν και οι 20 επαναλήψεις

Τέλος, εμφανίζονται οι πληροφορίες για την γενιά με την καλύτερη λύση, καθώς και ένας πίνακας, που περιέχει το σύνολο των αποστάσεων όλων των λύσεων κάθε γενιάς.

10. Στιγμιότυπα Εκτέλεσης Αλγορίθμου

Αναπαράσταση αρχικού συνόλου υποψήφιων λύσεων και πρώτη επανάληψη του αλγορίθμου:

```
--------------------------------------------------------------------------------
Generation 0
Set is:
[['E' 'A' 'C' 'B' 'D']
 ['C' 'D' 'E' 'A' 'B']
 ['C' 'E' 'D' 'B' 'A']
 ['C' 'E' 'A' 'B' 'D']
 ['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'E' 'D']
 ['C' 'E' 'A' 'B' 'D']
 ['C' 'B' 'D' 'E' 'A']
 ['A' 'E' 'C' 'B' 'D']
 ['D' 'C' 'A' 'B' 'E']]

Fitness of set is:
[18, 17, 20, 15, 15, 22, 15, 18, 18, 21]

Probability list of set is:
[0.1005586592178771, 0.09497206703910614, 0.11173184357541899, 0.08379888268156424, 0.08379888268156424, 0.12290502793296089,

Selection of parents:
[7 5 8 4 1 8 3 9]

Parents:
['C' 'B' 'D' 'E' 'A']
['A' 'B' 'C' 'E' 'D']
['A' 'E' 'C' 'B' 'D']
['D' 'C' 'E' 'A' 'B']
['C' 'D' 'E' 'A' 'B']
['A' 'E' 'C' 'B' 'D']
['C' 'E' 'A' 'B' 'D']
['D' 'C' 'A' 'B' 'E']
```

```
Children:
[['C' 'B' 'D' 'A' 'E']
 ['A' 'B' 'C' 'D' 'E']
 ['A' 'E' 'C' 'D' 'B']
 ['D' 'C' 'E' 'A' 'B']
 ['C' 'D' 'E' 'A' 'B']
 ['A' 'E' 'C' 'D' 'B']
 ['C' 'E' 'A' 'D' 'B']
 ['D' 'C' 'A' 'E' 'B']]

--------------------------------------------------------------------------------
```

Ένατη επανάληψη του αλγορίθμου:

```
--------------------------------------------------------------------------------
Generation 8
Set is:
[['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'D' 'E']
 ['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'D' 'E']
 ['A' 'B' 'C' 'D' 'E']
 ['D' 'C' 'E' 'A' 'B']
 ['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'D' 'E']
 ['D' 'C' 'E' 'A' 'B']
 ['D' 'C' 'E' 'A' 'B']]

Fitness of set is:
[15, 17, 15, 17, 17, 15, 15, 17, 15, 15]

Probability list of set is:
[0.0949367088607595, 0.10759493670886076, 0.0949367088607595, 0.10759493670886076, 0.10759493670886076, 0.0949367088607595,

Selection of parents:
[6 7 6 4 4 2 0 7]

Parents:
['D' 'C' 'E' 'A' 'B']
['A' 'B' 'C' 'D' 'E']
['D' 'C' 'E' 'A' 'B']
['A' 'B' 'C' 'D' 'E']
['A' 'B' 'C' 'D' 'E']
['D' 'C' 'E' 'A' 'B']
['D' 'C' 'E' 'A' 'B']
['A' 'B' 'C' 'D' 'E']
```

```
Children:
[['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'D' 'E']
 ['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'D' 'E']
 ['A' 'B' 'C' 'D' 'E']
 ['D' 'C' 'E' 'A' 'B']
 ['D' 'C' 'E' 'A' 'B']
 ['A' 'B' 'C' 'D' 'E']]

--------------------------------------------------------------------------------
```

Αποτελέσματα κάθε γενιάς:

```
--------------------------------------------------------------------------------

Fitness of each generation:
Generation 0: 179
Generation 1: 169
Generation 2: 170
Generation 3: 178
Generation 4: 168
Generation 5: 164
Generation 6: 162
Generation 7: 160
Generation 8: 158
Generation 9: 158
Generation 10: 162
Generation 11: 164
Generation 12: 164
Generation 13: 164
Generation 14: 168
Generation 15: 170
Generation 16: 170
Generation 17: 170
Generation 18: 170
Generation 19: 170
--------------------------------------------------------------------------------
```

Καλύτερη λύση:

```
--------------------------------------------------------------------------------
Generation with best score: 8
Score: 158
Best Path: ['D' 'C' 'E' 'A' 'B']
Distance of best path: 15
```