Εργασία Τεχνητής Νοημοσύνης και Έμπειρων Συστημάτων

Ματίνα Παπαδάκου

Π21127

Εισαγωγή:

Οι γενετικοί αλγόριθμοι (Genetic Algorithms) αποτελούν μια τάξη ευριστικών μεθόδων βελτιστοποίησης που είναι εμπνευσμένες από τις αρχές της βιολογικής εξέλιξης. Χρησιμοποιούνται για την επίλυση προβλημάτων αναζήτησης και βελτιστοποίησης, τα οποία συχνά είναι δύσκολο να αντιμετωπιστούν με συμβατικές μεθόδους. Οι γενετικοί αλγόριθμοι βασίζονται σε τρεις κύριες διεργασίες: επιλογή, διασταύρωση (crossover) και μετάλλαξη. Σε αυτές της διεργασίες στηριχθήκαμε και εμείς ώστε να λυθεί το πρόβλημα του πλανόδιου πωλητή. Ο πωλητής στο πρόβλημα αυτό θα πρέπει να ξεκινάει από την Πόλη 1 να πηγαίνει σε όλες τις υπόλοιπες πόλεις 1 φορά και να επιστρέφει πάλι στην Πόλη 1, χρησιμοποιώντας την καλύτερη διαδρομή.

Κώδικας (γλώσσα προγραμματισμού Python):

```
import random

# Ορισμός των αποστάσεων
distances = [
    [0, 10, 20, 5, 10],
    [10, 0, 2, 10, 6],
    [20, 2, 0, 7, 1],
    [5, 10, 7, 0, 20],
    [10, 6, 1, 20, 0]
]

# Δημιουργία αρχικού πληθυσμού
def initial_population(pop_size):
    population = []
    for _ in range(pop_size):
        individual = list(range(2, 6)) # Δημιουργία λίστας πόλεων από 2 έως 5
        random.shuffle(individual) # Τυχαία ανακατεύουμε τη σειρά των πόλεων
        individual.insert(0, 1)  # Προσθήκη της πόλης 1 στην αρχή
        individual.append(1)     # Προσθήκη της πόλης 1 στο τέλος
        population.append(individual) # Προσθέτουμε την πλήρη διαδρομή στον
πληθυσμό
    return population
```

Η Μέθοδος initial_population(pop_size):

Αυτή η μέθοδος έχει σκοπό την δημιουργία του αρχικού πληθυσμού του γενετικού αλγορίθμου. Ο πληθυσμός αυτός δημιουργείται τυχαία και για να λύσουμε το πρόβλημα θα πρέπει ο «πωλητής» να ξεκινάει πάντα από την πόλη 1, να καταλήγει σε αυτή και να περνάει από όλες τις υπόλοιπες πόλεις 1 φορά.

```
# Υπολογισμός της συνολικής απόστασης μιας διαδρομής
def calculate_distance(route):
    distance = 0
    for i in range(len(route) - 1):
        distance += distances[route[i] - 1][route[i + 1] - 1]  # Υπολογισμός
απόστασης
    return distance
```

Η Μέθοδος def calculate_distance(route):

Υπολογίζει την συνολική απόσταση κάθε διαδρομής, ώστε στην συνέχεια χρησιμοποιώντας το να βρούμε την καλύτερη.

```
# Συνάρτηση καταλληλότητας
def fitness(route):
    return 1 / float(calculate_distance(route))
```

Η Μέθοδος def fitness(route):

Την χρησιμοποιούμε ώστε να βρούμε τα ποσοστά καταλληλότητας της κάθε διαδρομής.

```
# Επιλογή γονέων
def selection(population, fitnesses):
    selected = random.choices(population, weights=fitnesses, k=2) # To weight
επιλέγει με βάση την πιθανότητα (ρουλέτα) και το k δείχνει πόσοι γονείς θα
επιλεγούν
    return selected
```

Η Μέθοδος def selection(population, fitnesses):

Με βάση την τεχνική της ρουλέτας, δηλαδή με βάση την πιθανότητα επιλογής της κάθε διαδρομής, επιλέγουμε τους δύο καλύτερους γονείς.

```

```

```
# Αναπαραγωγή με διασταύρωση ενός σημείου
def crossover(par1, par2):
    start, end = sorted(random.sample(range(len(par1)), 2))  # Επιλέγονται δύο τυχαία σημεία από το χρωμόσωμα
    child = [0] * len(par1) # Δημιουργείται ένα κενό παιδί (με μηδενικά)
    child[start:end] = par1[start:end]  # Αντιγράφεται τμήμα από τον πρώτο γονέα στο παιδί
    pointer = 0
    for i in range(len(par1)):
        if not par2[i] in child: # Εάν το στοιχείο του δεύτερου γονέα δεν υπάρχει ήδη στο παιδί
            while child[pointer] != 0: # Βρες το πρώτο κενό (0) σημείο στο παιδί
                pointer += 1
            child[pointer] = par2[i] # Αντιγραφή του στοιχείου από τον δεύτερο γονέα στο παιδί
    child[5] = 1 # Θέτουμε το τελευταίο στοιχείο του παιδιού ως 1
    return child
```

Η Μέθοδος def crossover(par1, par2):

Σε αυτή τη μέθοδο υλοποιείται η αναπαραγωγή με διασταύρωση ενός σημείου. Αρχικά επιλέγονται τυχαία δύο σημεία start και end από το μήκος του πρώτου γονέα par1. Τα σημεία αυτά καθορίζουν τη θέση στην οποία θα γίνει η διασταύρωση. Στην συνέχεια δημιουργείται ένα παιδί child με το ίδιο μήκος όσο ο γονέας par1, γεμίζοντας το με μηδενικά. Το τμήμα από τον πρώτο γονέα par1 που βρίσκεται μεταξύ των σημείων start και end αντιγράφεται απευθείας στο παιδί child. Για κάθε στοιχείο par2[i] του δεύτερου γονέα par2, ελέγχεται αν υπάρχει ήδη στο παιδί child. Αν δεν υπάρχει, το στοιχείο par2[i] αντιγράφεται στο παιδί child στην πρώτη διαθέσιμη θέση (pointer). Τέλος ορίζεται το τελευταίο στοιχείο του παιδιού ως 1, αυτό αποτελεί την τελική πόλη στην διαδρομή του παιδιού, και επιστρέφεται το παιδί που δημιουργήθηκε.

```
# Μετάλλαξη
def mutate(individual, mutation_rate):
    for i in range(1,4):
        if random.random() < mutation_rate: # Ελέγχει αν θα γίνει μετάλλαξη στην τρέχουσα θέση, με βάση το mutation_rate
            j = random.randint(1,4) # Επιλέγει τυχαία μια άλλη θέση από 1 έως 4 για την ανταλλαγή
            individual[i], individual[j] = individual[j], individual[i] # Ανταλλάσσει τις θέσεις i και j στην τρέχουσα διαδρομή (individual)
    return individual
```

Η Μέθοδος def mutate(individual, mutation_rate):

Μέσω αυτής της μεθόδου καθορίζεται εάν θα γίνει μετάλλαξη σε κάποιο άτομο. Ελέγχεται αν θα γίνει μετάλλαξη στην τρέχουσα θέση, με βάση το mutation_rate(στο παράδειγμα είναι 20%) με τυχαία σύγκριση. Στην συνέχεια γίνεται τυχαία πάλι μία ανταλλαγή.

```
# Γενετικός Αλγόριθμος
def genetic_algorithm(pop_size, generations, mutation_rate):
    # Δημιουργία αρχικού πληθυσμού
    population = initial_population(pop_size)
    # Επανάληψη για τον καθορισμένο αριθμό γενιών
    for _ in range(generations):
        # Υπολογισμός καταλληλότητας για κάθε άτομο στον πληθυσμό
        fitnesses = [fitness(individual) for individual in population]
        # Δημιουργία νέου πληθυσμού
        new_population = []
        # Δημιουργία νέων ατόμων μέχρι να φτάσουμε το μέγεθος του πληθυσμού
        for _ in range(pop_size):
            # Επιλογή δύο γονέων με βάση την καταλληλότητα
            par1, par2 = selection(population, fitnesses)
            # Δημιουργία νέου ατόμου με διασταύρωση των γονέων
            child = crossover(par1, par2)
            # Εφαρμογή μετάλλαξης στο νέο άτομο
            child = mutate(child, mutation_rate)
            # Προσθήκη του νέου ατόμου στον νέο πληθυσμό
            new_population.append(child)
        # Ανανέωση του πληθυσμού με το νέο πληθυσμό
        population = new_population
    # Εύρεση της καλύτερης διαδρομής στον τελικό πληθυσμό
    best_route = max(population, key=lambda x: fitness(x))
    # Επιστροφή της καλύτερης διαδρομής και της συνολικής απόστασής της
    return best_route, calculate_distance(best_route)
```

Η Μέθοδος genetic_algorithm(pop_size, generations, mutation_rate):

Μέσω αυτής της μεθόδου εκτελούνται όλες οι άλλες μέθοδοι ώστε να εκτελεστεί ο γενετικός αλγόριθμος. Αρχικά δημιουργείται ο αρχικός πληθυσμός. Στην συνέχεια εκτελείται επανάληψη με βάση των καθορισμένο αριθμό των γενιών (στην περίπτωση μας 5). Έπειτα υπολογίζεται η καταλληλότητα για κάθε άτομο στον πληθυσμό. Δημιουργούμε έναν αρχικό, κενό, νέο πληθυσμό. Ξεκινάμε μία επανάληψη ώστε να δημιουργήσουμε νέα άτομα στον πληθυσμό. Η επανάληψη τερματίζει όταν φτάσουμε το μέγεθος του πληθυσμού(στο παράδειγμά μας είναι ο αριθμός 10). Μέσα σε αυτή την επανάληψη επιλέγονται δύο γονείς με βάση την καταλληλότητα, μέσω της διασταύρωσης ενός σημείου των γονέων δημιουργείται το παιδί, το παιδί υποβάλλεται στην διαδικασία της μετάλλαξης. Στην συνέχεια προστίθεται το παιδί στον νέο πληθυσμό. Με το τέλος της 2^{ης} επανάληψης ανανεώνεται ο πληθυσμός με τον νέο

πληθυσμό. Τέλος γίνεται η εύρεση της καλύτερης διαδρομής και ο υπολογισμός της συνολικής απόστασης της.

```
# Παράμετροι
pop_size = 10
generations = 5
mutation_rate = 0.2 #20% πιθανότητα μετάλλαξης

# Εκτέλεση του αλγορίθμου
best_route, best_distance = genetic_algorithm(pop_size, generations,
mutation_rate)
print(f"Best route: {best_route}")
print(f"Best distance: {best_distance}")
```

Παραδείγματα Εκτέλεσης Κώδικα:

```
Best route: [1, 2, 5, 3, 4, 1]
Best distance: 29
```

```
Best route: [1, 5, 3, 2, 4, 1]
Best distance: 28
```

```
Best route: [1, 5, 2, 3, 4, 1]
Best distance: 30
```