ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα

6ο εξάμηνο

2022-2023 Εργασία

Κωνσταντίνος Λοϊζίδης

Π20007

19/05/2023

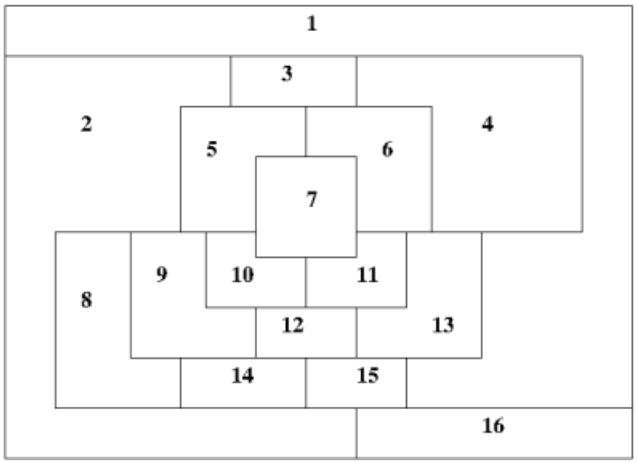
Περιεχόμενο

ΕΚΦΩΝΗΣΗ

Θέμα προαιρετικής εργασίας για το μάθημα «Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα». Bonus 2 βαθμοί.

<u>Η εργασία είναι ατομική. Παραδοτέο είναι αρχείο pdf με τεκμηριωμένο κώδικα και παραδείγματα εκτέλεσης, περίπου 5 σελίδων. Υποβολή αποκλειστικά μέσω gunet2. ΜΗ ΣΤΕΙΛΕΤΕ EMAIL. ΜΗ ΣΤΕΙΛΕΤΕ ΠΗΓΑΙΟ ΚΩΔΙΚΑ.</u>

Αναπτύξτε πρόγραμμα χρωματισμού του παρακάτω γράφου με χρήση γενετικών αλγορίθμων και γλώσσα προγραμματισμού της επιλογής σας. Τα διαθέσιμα χρώματα είναι 4: μπλε, κόκκινο, πράσινο, κίτρινο.



Χρησιμοποιείστε τυχαίο αρχικό πληθυσμό με πλήθος της δικής σας επιλογής. Χρησιμοποιείστε συνάρτηση καταλληλότητας και διαδικασία επιλογής γονέων σας της δικής σας επιλογής, επίσης. Χρησιμοποιείστε αναπαραγωγή με διασταύρωση ενός σημείου. Επιλέξτε αν θέλετε να κάνετε και μερική ανανέωση πληθυσμού σε κάποιο ποσοστό π.χ. 30% και μετάλλαξη ενός ψηφίου π.χ. στο 10% του πληθυσμού.

Παραδοτέα της εργασίας είναι ένα pdf (όχι zip, όχι πηγαίος κώδικας) που να περιλαμβάνει τον κώδικα και να τον εξηγεί, να εξηγεί τον τρόπο δράσης του υπολογιστή σύμφωνα με τον αλγόριθμο επίλυσης και να περιλαμβάνει παραδείγματα εκτέλεσης του προγράμματος που αναπτύξατε.

ΚΩΔΙΚΑΣ

```
import random

graph = {...}

def create_population(population_size):...

def calculate_fitness(generated_strings,total_score):...

def pick_range(string_scores):...

def pick_index(random_value, my_list):...

def parent_selection(percentage,pick_ranges,total_score):...

def crossover_parents(parents):...

def mutate_parents(n):...

def population_renewal(n):...

def find_maximum_score(list_of_scores):...

def biggest_integers_indexes(input_list,n):...

def extract_strings_by_indexes(string_list, indexes):...

def genetic_algorithm(generated_strings,total_score):...

generated_strings = create_population(100)

total_score = 0

best_coloring = genetic_algorithm(generated_strings,total_score)

print(best_coloring)
```

```python
def create_population(population_size):
    characters = ["G", "B", "Y", "R"]
    strings = []
    for _ in range(population_size):
        string = ''.join(random.choice(characters) for _ in range(16))
        strings.append(string)
    return strings

def calculate_fitness(generated_strings,total_score):

    string_scores = []
    string_score = 0

    for colored_string in generated_strings:
        string_score = 0
        for colored_box, dic in graph.items():
            for neighbor in dic['neighbors']:
                graph[colored_box]['score'] = 0
        for colored_box, dic in graph.items():
            for neighbor in dic['neighbors']:
                if colored_string[neighbor-1] != colored_string[colored_box-1]:
                    graph[colored_box]['score'] += 1
            string_score += graph[colored_box]['score']
        total_score += string_score
        string_scores.append(string_score)
    return string_scores,total_score
```

```
def pick_range(string_scores):
    pick_range = []
    previous_score = string_scores[0]
    pick_range.append(previous_score)
    for i in range(100):
        if string_scores[i] != string_scores[0]:
            previous_score += string_scores[i]
            pick_range.append(previous_score)
    return pick_range

def pick_index(random_value, my_list):
    for i in range(len(my_list)):
        if random_value <= my_list[i]:
            return i
    return len(my_list) - 1

def parent_selection(percentage,pick_ranges,total_score):
    iterations = int((len(generated_strings)*percentage)/100)
    selected_parents = []
    for _ in range(iterations):
        random_value = random.randint(1, total_score)
        i = pick_index(random_value,pick_ranges)
        selected_parents.append(generated_strings[i])
    total_score = 0
    return selected_parents,total_score
```

```python
def crossover_parents(parents):
    num_parents = len(parents)
    if num_parents % 2 != 0:
        last_parent = parents[-1]
        random_parent = random.choice(parents[:-1])
        pairs = [(parents[i], parents[i+1]) for i in range(0, num_parents-1, 2)]
        pairs.append((last_parent, random_parent))
    else:
        pairs = [(parents[i], parents[i+1]) for i in range(0, num_parents, 2)]

    new_strings = []
    for pair in pairs:
        parent1, parent2 = pair
        n = random.randint(1, len(parent1)-1)
        new_string1 = parent1[:n] + parent2[n:]
        new_string2 = parent2[:n] + parent1[n:]
        new_strings.append(new_string1)
        new_strings.append(new_string2)

    return new_strings

def mutate_parents(n):
    list_of_parents = create_population(n)
    return list_of_parents

def population_renewal(n):
    list_of_parents = create_population(n)
    return list_of_parents
```

```python
def find_maximum_score(list_of_scores):
    max_score = max(list_of_scores)  # Find the maximum score in the list
    max_index = list_of_scores.index(max_score)  # Find the index of the maximum score
    return max_index

def biggest_integers_indexes(input_list,n):
    # Enumerate the input list to get both values and indexes
    enumerated_list = list(enumerate(input_list))

    # Sort the enumerated list based on the integer values in descending order
    sorted_list = sorted(enumerated_list, key=lambda x: x[1], reverse=True)

    # Extract the indexes of the n biggest integers
    indexes = [item[0] for item in sorted_list[:n]]

    return indexes

def extract_strings_by_indexes(string_list, indexes):
    new_list = []
    for index in indexes:
        if index < len(string_list):
            new_list.append(string_list[index])
    return new_list
```

```
def genetic_algorithm(generated_strings,total_score):
    i = 0
    string_scores,total_score = calculate_fitness(generated_strings,total_score)
    new_population = generated_strings
    if 87 in string_scores:
        index = string_scores.index(87)
        return generated_strings[index]
    else:
        while i < 100:
            if i != 0:
                string_scores,total_score = calculate_fitness(new_population,total_score)
                if 87 in string_scores:
                    index = string_scores.index(87)
                    return generated_strings[index]
            pick_ranges = pick_range(string_scores)
            selected_parents,total_score= parent_selection(60,pick_ranges,total_score)
            crossovered_parents = crossover_parents(selected_parents)
            mutated_parents = mutate_parents(20)
            new_parents = population_renewal(10)
            best_parents_indexes = biggest_integers_indexes(string_scores,10)
            best_parents = extract_strings_by_indexes(new_population, best_parents_indexes)

            new_population = crossovered_parents + mutated_parents + new_parents + best_parents

            i += 1
    index = find_maximum_score(string_scores)
    print(string_scores[index])
    return new_population[index]
```

```
total_score = 0

best_coloring = genetic_algorithm(generated_strings,total_score)

print(best_coloring)
```

Περιγραφή Γενετικού αλγόριθμου(Κώδικα)

Αρχικά το graph μου έχει την μορφή :

```
{
 1 :
{'neighbors' : [neighbor_1, ..., neighbor_ k],
 'score' : 0,},
 2:
{'neighbors' : [neighbor_1, ..., neighbor_ k],
 'score' : 0,},
 .
 .
.
16:
 {'neighbors' : [neighbor_1, ..., neighbor_ k],
  'score' : 0,},
}
```

όπου κάθε key-value pair έιναι ένας κόμβος μέσα στον γράφο.Το key είναι ο αριθμός του κόμβου και το value είναι είναι ένα dictionary, το οποίο περιέχει δύο keys το ‘neighbors’ που έχει σαν value μία λίστα απο τους γείτονες του και το ‘score’, το οποίο περιέχει το score του κάθε κόμβου.

Η def create_population, δημιουργεί τον πληθυσμό μου όπου κάθε γονέας είναι ένα 16 length string το οποίο αοτελείται απο συνδιασμούς των χαρακτήρων:

["G", "B", "Y", "R"] (G = Green, B= Blue,Y = Yellow,R = Red)

Η def calculate_fitness,υπολογίζει το score του κάθε κόμβου και με αυτό υπολογίζουμε το score κάθε γονέα.Επίσης υπολογίζε το συνολικό άθροισμα όλων των γονέων.

Η def pick_range,χρησιμοποιεί τα score του κάθε γονέα που υπολγίσαμε ώστε να δημιουρήσει μία λίστα, όπου τα στοιχεία αυτά θα μας βοηθήσουν να εφαρμόσουμε την μέθοδο της ρουλέτας.

Η def pick_index,παίρνει σαν παράμετρο έναν random_integer και την λίστα που μας επέστρεψε η pick_range και ελέγχει σε ποιο range αριθμών βρίσκεται ο random_integer και μας επιστρέφει το index στο οποίο ανήκει αυτός ο αριθμός.

Η def parent_selection, με την βοήθεια της pick_index μπορεί να επιλέξει τους γονείς με την μέθοδο της ρουλέτας.Στους οποίους εφαρμόζουμε το crossover.

Η def crossover_parents, κάνει τους γονείς που επιλέχθηκαν crossover.

Η def mutate_parents, εφαρμόζει mutation σε ένα σύνολο των γονέων.

(δέχεται σαν είσοδο έναν ακέραιο αριθμό που καθορίζει σε πόσους γονείς θα εφαρμοστεί mutation)

Η def population_renewal, ξαναδημιουργά ένα ποσοστό πληθυσμού.

Η def find_maximum_score, βρίσκει το maximum_score

Η def biggest_integers_indexes,επιστρέφει τα index των μεγαλύτερων score(ουσιαστικά χρησιμοποιήται για ελιτισμό).

Η def extract_strings_by_indexes, μας επιστέφει τα string που βρίσκονται στα αντίστοιχα indexes.

Η genetic_algorithm,ουσιαστικά υλοποιά τον γενετικό μας αλγόριθμο με την χρήση των παραπάνω συναρτήσεων.

Δηλαδή,όπως βλέπουμε στον κώδικα δημιουργούμε τον αρχικό πληθυσμό μας όπου και τον παιρνάμε με παράμετρο στην συνάρτηση μας.Μετά καλούμε την calculate_fitness ώστε να υπολογιστούνε τα score,ελέγχουμε εάν έχουμε τον τέλειο γονέα που στην συγκεκριμένη υλοποίηση αυτό σημαίνει ότι πρέπει να έχει score 87.Εάν υπάρχει τον επιστρέφουμε αυτόν μαζί με το score του,αλλιώς τρέχουμε ένα while loop 100 επαναλήψεων στην περίπτωση μας.Το οποίο αν δεν είμαστε στην πρώτη επανάληψη καλεί την calculate_fitness.Μετά ελέγχουμε εάν βρήκαμε τον καλύτερο χρωματισμό και αν δεν τον βρήκαμε συνεχίζουμε με το να βρούμε τα ranges που βρίσκεται ο κάθε γονέας με την χρήση της pick_ranges.Με βάση αυτά καλούμε την parent_selection όπου με την μέθοδο της ρυλέτας επιλέγει τους γονείς μας.Τους οποίος κάνουμε crossover με την crossover_parents,μετά καλούμε την mutated_parents όπου κάνουμε mutate μερικούς γονείς.

Έπειτα χρησιμοποιούμε την population_renewal ώστε να δημιουργίσουμε μερικούς καινούριους γονείς.Μετά εφαρμόζουμε ελιτισμό σε ένα επιθυμειτώ ποσοστό.Τέλος βάζουμε σε μία λίστα μαζοί τους crossovered_parents τους mutated_parents τους new_parents και τους best_parents.Αν τελειώσουν όλες οι επαναλήψεις του while μας επιστρέφεται ο καλυτερος γονέας της τελευταίας επανάληψης μαζί με το score του!!

ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ

Τα συγκεκριμένα παραδείγματα έτρεξαν με 60% επιλογή γονέων με την μέθοδο της ρουλέτας,εφαρμόστηκε 20% mutation,10% population renewal και 10% elitism

Υπενθύμιση η βέλτιστη λύση είναι το 87