**ΠΑΠΠΑΣ ΑΘΑΝΑΣΙΟΣ**

**Π19212**

# Εργασία στο μάθημα "Τεχνητή Νοημοσύνη και Έμπειρα συστήματα"

## Κωδικοποίηση προβλήματος

Το σχήμα που δώθηκε μπορεί να γίνει αντιληπτό και ως ένας γράφος, με τις περιοχές να αποτελούν τους κόμβους και τα σύνορα να αποτελούν τις ακμές. Έτσι έφτιαξα 2 λίστες, μία για τους κόμβους και μία για τις ακμές. Στην συνέχεια έφτιαξα μια κλάση που έχει 4 attributes, τις 2 παραπάνω λίστες, μία λίστα που έχει τα διαθέσιμα χρώματα και μια λίστα που αποτελείται από μέλη μιας άλλης κλάσης, της Parent (θα εξηγηθεί στην συνέχεια).

## Αρχικοποίηση πληθυσμού

Ο αρχικός μου πληθυσμός κατασκευάστηκε επιλέγοντας ένα τυχαίο color map που μετά το περνούσα στον constructor της κλάσης Parent. Αυτή έχει 2 attributes, το color map και ένα ranking που προκύπτει από την fitness function. Αυτή η διαδικασία επαναλήφθηκε 100 φορές, που είναι το μέγεθος του αρχικού μου πληθυσμού.

## Συνάρτηση καταλληλότητας

Για κάθε γείτονα ενός κόμβου που είχε διαφορετικό χρώμα από αυτόν, το ranking του parent αυξάνεται κατά 10. Παραθέτω και τον σχετικό κώδικα

```
def fitness_function(self, edge_lst):
    for i in range(1, len(self.color_map)):
        for j in edge_lst:
            if j[0] == i:
                if self.color_map[i-1] != self.color_map[j[1] - 1]:
                    self.ranking += 10
    return self.ranking
```

## Επιλογή γονέων

Σε κάθε τέτοια διαδικασία επέλεγα 10 γονείς, οι οποίοι μετά ζευγάρωναν μεταξύ τους και μου έδιναν πάλι 100 πληθυσμό για την επόμενη γενιά. Αρχικά δοκίμασα να τους επιλέγω όλους με την μέθοδο της ρουλέτας, όμως αυτό δεν είχε τα επιθυμητά αποτελέσματα, καθώς δεν υπήρχε συνεπή βελτίωση μεταξύ γενεών και η τελευταία γενιά ήταν συνήθως μεταξύ του max και min ranking του αρχικού πληθυσμού. Έτσι, δοκίμασα να περνάω αυτούσιο τον γονέα που είχε το καλύτερο ranking σε κάθε γενιά (προφανώς έβαλα και ένα break loop σε περίπτωση που max score είχαν περισσότεροι του ενός) και τους υπόλοιπους 9 τους επέλεγα με την τεχνική της ρουλέτας. Την υλοποίησα με αυτόν τον τρόπο (το ranking_list έχει τα rankings όλων των γονέων και το tmp_parent_lst1 αποθηκεύει προσωρινά τους γονείς που θα σταλθούν για αναπαραγωγή, είναι απλά δύο helper lists).

```
#roullete technique
total = 0
while total < 9:
    ball = randrange(1,sum+1)
    cnt = 0
    for i in range(len(ranking_list)):
        cnt += ranking_list[i]
        if ball <= cnt:
            if self.parents[i] not in tmp_parent_lst1:
                tmp_parent_lst1.append(self.parents[i])
                total+=1
                break
            continue
self.parents = tmp_parent_lst1
```

## Αναπαραγωγή

Με διασταύρωση μονού σημείου, υπήρχε πρόβλημα με τα παιδιά να μηνξεπερνάνε τους γονείς τους. Μετά από τις 1-2 πρώτες γενιές, το καλύτερο score κόλλαγε και δεν ξεπερνιόταν ποτέ. Είχα δηλαδή premature convergence και ψάχνοντας για το τι μπορώ να κάνω σε αυτήν την περίπτωση, δοκίμασα να κάνω uniform αναπαραγωγή:

```
def reproduction(self):
    tmp_parent_lst2 = []
    for i in self.parents:
        for j in self.parents:
            tmp_parent_lst2.append(Parent([i.color_map[m] if randrange(0,2) == 0 else j.color_map[m] for m in range(16)]))
    self.parents = tmp_parent_lst2
```

Το πρόβλημα λύθηκε σχεδόν εντελώς και ο αλγόριθμος έφτανε σε λύσεις που ήταν πάρα πολύ κοντά στο βέλτιστο (δηλαδή μόνο 2-3 ακμές είχαν κόμβους του ίδιου χρώματος), παρόλο και πάλι ύπηρχε σύγκλιση νωρίς (μετά απο 7-8 γενιές). Χρειαζόταν λοιπόν λίγο ακόμα genetic variation, και έτσι

αποφάσισα να χρησιμοποιήσω και mutation στο 10% του πληθυσμού της κάθε γενιάς. Επέλεγα τυχαία αυτό το 10%, τυχαία ένα χρώμα στο color map του για να το αντικαταστήσω, και τυχαία το χρώμα με το οποίο θα το αντικαθιστούσα. Πέτυχε το αναμενόμενο αποτέλεσμα, καθώς πλέον η σύγκλιση δεν συνέβαινε τόσο νωρίς. Μετά από κάποια τρεξίματα του αλγόριθμου έφτασα στην βέλτιστη λύση, δηλαδή κανένας κόμβος δεν συνόρευε με κόμβο ίδιου χρώματος.

## Οπτικοποίηση αποτελεσμάτων

Χρησιμοποίησα την networkx για να σχεδιάσω τον γράφο και την matplotlib για να τον οπτικοποιήσω. Αυτή είναι η βέλτιστη λύση στην οποία έφτασα