Πανεπιστήμιο Πειραιώς

Τμήμα Πληροφορικής

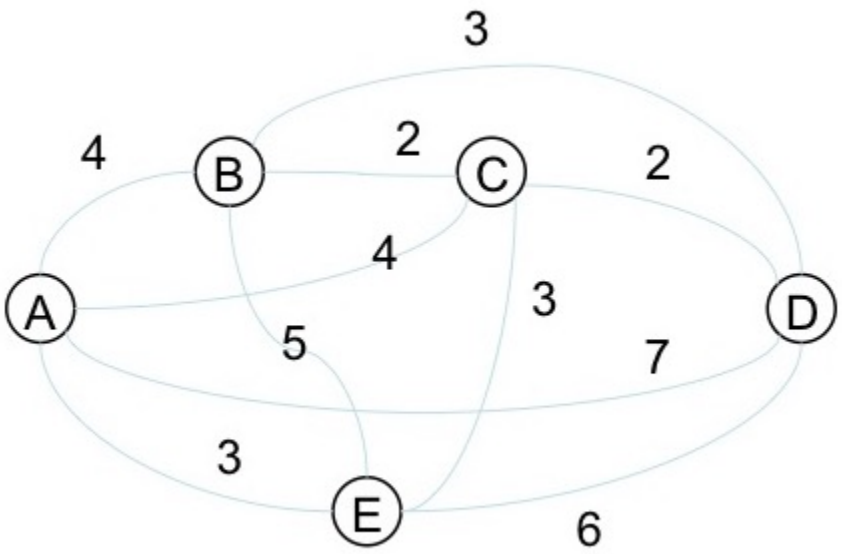
2021-2022

Ονοματεπώνυμο: Κωνσταντίνος Μαυρέλης
Αριθμός Μητρώου: Π19101

Τεχνητή Νοημοσύνη

Εκφώνηση εργασίας

Θέμα 1: Αναπτύξτε πρόγραμμα επίλυσης του Traveling Salesman Problem με χρήση γενετικών αλγορίθμων και γλώσσα προγραμματισμού της επιλογής σας. Ο γράφος αποτελείται από πλήρως διασυνδεδεμένες πόλεις όπως φαίνεται στο παρακάτω σχήμα.



Γλώσσα προγραμματισμού : JavaScript

Αναπαράσταση πόλεων

Για την αναπαράσταση των πόλεων χρησιμοποιήθηκε array/λίστα με Objects/αντικείμενα τα οποία περιέχουν το όνομα της πόλης, τη θέση στην οποία βρίσκεται η κάθε πόλη με βάση το κόστος της και ένα ακόμα array/λίστα το οποίο αναπαριστά τα κόστη της κάθε πόλης με όλες τις άλλες.

```
let cities = [
   {name:"A",cityIndex: 0,cost: [0,4,4,7,3]},
   {name:"B",cityIndex: 1,cost: [4,0,2,3,5]},
   {name:"C",cityIndex: 2,cost: [4,2,0,2,3]},
   {name:"D",cityIndex: 3,cost: [7,3,2,0,7]},
   {name:"E",cityIndex: 4,cost: [3,5,3,6,0]},
]
```

Ο χρήστης μπορεί να ορίσει τον αρχικό πληθυσμό κατά την εκκίνηση του προγράμματος και με την παρακάτω συνάρτηση δημιουργούνται τα τυχαία Paths.

```
function createRandomPaths(howManyPaths){
   let paths = _.fill(Array(howManyPaths),0).map(x=> {
       let shuffledPath = _.shuffle(cities)
       let firstElement = _.first(shuffledPath)
       shuffledPath.push(firstElement)
       return shuffledPath
   })
   return paths
}
```

Συνάρτηση Καταλληλότητας/Fitness

Για να υπολογίσουμε το Fitness κάθε διαδρομής χρειαζόμαστε Το κόστος της εκάστοτε διαδρομής δηλαδή τη συνολική απόσταση. Αυτή υπολογίζεται από τη συνάρτηση calculate cost.

```
function calculatePathCost(path){
   let timesRun = 0
   let costTotal = 0
   path.forEach(city =>{
       let indexOfCity = path.indexOf(city)
       //Prevent add cost of last element with itself
       if(timesRun+1 < path.length){
           costTotal+= path[indexOfCity+1].cost[city.cityIndex]
       }
       timesRun+=1
   })
   return costTotal
}
```

Η καταλληλότητα με τη σειρά της είναι μία συνάρτηση η οποία επιστρέφει έναν αριθμό αντιστρόφως ανάλογο με το κόστος της διαδρομής δηλαδή το καλύτερο path είναι αυτό με τη μικρότερη απόσταση. Συνεπώς όσο μεγαλύτερο το Fitness score τόσο καλύτερο και το path.

```
function calculateFitnessFromCost(cost){
   return 100/cost
}
```

Μερική ανανέωση πληθυσμού

Για την μερική ανανεώση πληθυσμού χρησιμοποιήθηκε μία συνάρτηση mutate path η οποία καλείται με βάση μιας πιθανότητας που ορίζει ο χρήστης κατά την έναρξη του προγράμματος

```
function mutatePath(path){
   //Mutate cities
   let mutationArray = path
   let arrayToShuffle = mutationArray.slice(1,-1)
   arrayToShuffle = _.shuffle(arrayToShuffle)
   arrayToShuffle.unshift(mutationArray[0])
   arrayToShuffle.push(mutationArray[0])
   return arrayToShuffle
}
```

Single point Crossover/ Αναπαραγωγή με διασταύρωση ενός σημείου

Για την αναπαραγωγή με διασταύρωση ενός σημείου χρησιμοποιήθηκε μία συνάρτηση η οποία επιλέγει δύο τυχαίες διαδρομές/γονείς από το σύνολο τον paths και τους κόβει και τους δύο σε ένα τυχαίο σημείο. Στη συνέχεια ανταλλάσσονται τα σημεία μεταξύ των δύο γονιών δημιουργώντας δύο νέους απογόνους. Στην περίπτωση του συγκεκριμένου προγράμματος διότι ο αριθμός των πόλεων είναι πολύ μικρός, η πιθανότητα να υπάρχουν διπλότυπες πόλεις είναι μεγάλη , για αυτό το λόγο χρησιμοποιήθηκε άλλη μία συνάρτηση για την αντικατάσταση των διπλότυπων πόλεων σε μοναδικών.

```
function singlePointCrossOver(paths){
   //Pick 2 Random Parents
   let randomPaths = _.shuffle([...paths])
   let parent1 = randomPaths.pop()
   let parent2 = randomPaths.shift()
   let randomCrossPoint = _.random(1,parent1.length-1)
   let child1 =
[...parent1.slice(0,randomCrossPoint),...parent2.slice(randomCrossPoint,parent2.length)]
   let child2 =
[...parent1.slice(randomCrossPoint,parent1.length),...parent2.slice(0,randomCrossPoint)]
   //Make arrays unique and first and last points same
   child1[child1.length-1] = child1[0]
   child2[child2.length-1] = child2[0]

   //Remove duplicates and replace them with left over
   child1 = child1.map(((s, i) => v => {
       if (!s.has(v)) {
           s.add(v);
           return v;
       }
       while (s.has(cities[i])) ++i;
       s.add(cities[i]);
       if(!cities[i]) return v
       return cities[i];
   })(new Set, 0));

   child2 = child2.map(((s, i) => v => {
       if (!s.has(v)) {
           s.add(v);
           return v;
       }
       while (s.has(cities[i])) ++i;
       s.add(cities[i]);
       if(!cities[i]) return v
       return cities[i];
```

```
    })(new Set, 0));
    paths[paths.indexOf(parent1)] = child1
    paths[paths.indexOf(parent2)] = child2

    return paths
}
```

Συνάρτηση γενετικών αλγορίθμων

Αυτή η συνάρτηση είναι υπεύθυνη να τρέξει και να λύσει το πρόβλημα με το γενετικό αλγόριθμο. Επαναλαμβάνεται έως ότου βρεθεί πας με Fitness μεγαλύτερο ίσο του Fitness limits

```
async function runGeneticAlgorithm(howManyPaths,FitnessLimit){
    let isFitnessLimitReached = false
    let bestPath = null
    let timesRun = 1
    //Create random Paths
    let paths = createRandomPaths(howManyPaths)

    while (!isFitnessLimitReached){
        console.log(`----------------------[${timesRun}]----------------------`)
        paths.forEach(path=>{
            let cost = calculatePathCost(path)
            let fitness = calculateFitnessFromCost(cost)
            if( fitness >= FitnessLimit){
                isFitnessLimitReached = true
                bestPath = path
            }
            console.log(`Cost: ${cost}\nFitness: ${fitness}\n Path: ${path.map(city => `${city.name} ->
`).join("").slice(0,-3)}`)
            console.log("------------------------------------------------------------")
        })
        //Mutate some with chance
        paths.forEach(path=>{
            let random = Math.random();
            if(random < mutationChance/100){
                let mutatedPath = mutatePath(path)
                paths[paths.indexOf(path)] = mutatedPath
                console.log(` Path: ${path.map(city => `${city.name} -> `).join("").slice(0,-3)}
Mutated into path ${mutatedPath.map(city => `${city.name} -> `).join("").slice(0,-3)}`)

            }
        })
```

```
        //CrossOver some with Chance
        let random = Math.random()
        if(random < crossOverChance/100){
            paths = singlePointCrossOver(paths)
        }
        timesRun+=1
    }
    console.log("------------------------------------------------------------")
    console.log("Fitness Limit Reached:")
    console.log(`Cost: ${calculatePathCost(bestPath)}\nFitness:
${calculateFitnessFromCost(calculatePathCost(bestPath))}\n Path: ${bestPath.map(city => `${city.name}
-> `).join("").slice(0,-3)}`)
    console.log("------------------------------------------------------------")
}
```

Παραδείγματα εκτέλεσης του προγράμματος

5 διαδρομών με πιθανότητα μετάλλαξης των πόλεων 40% και πιθανότητα διασταύρωσης ενός σημείου 10%

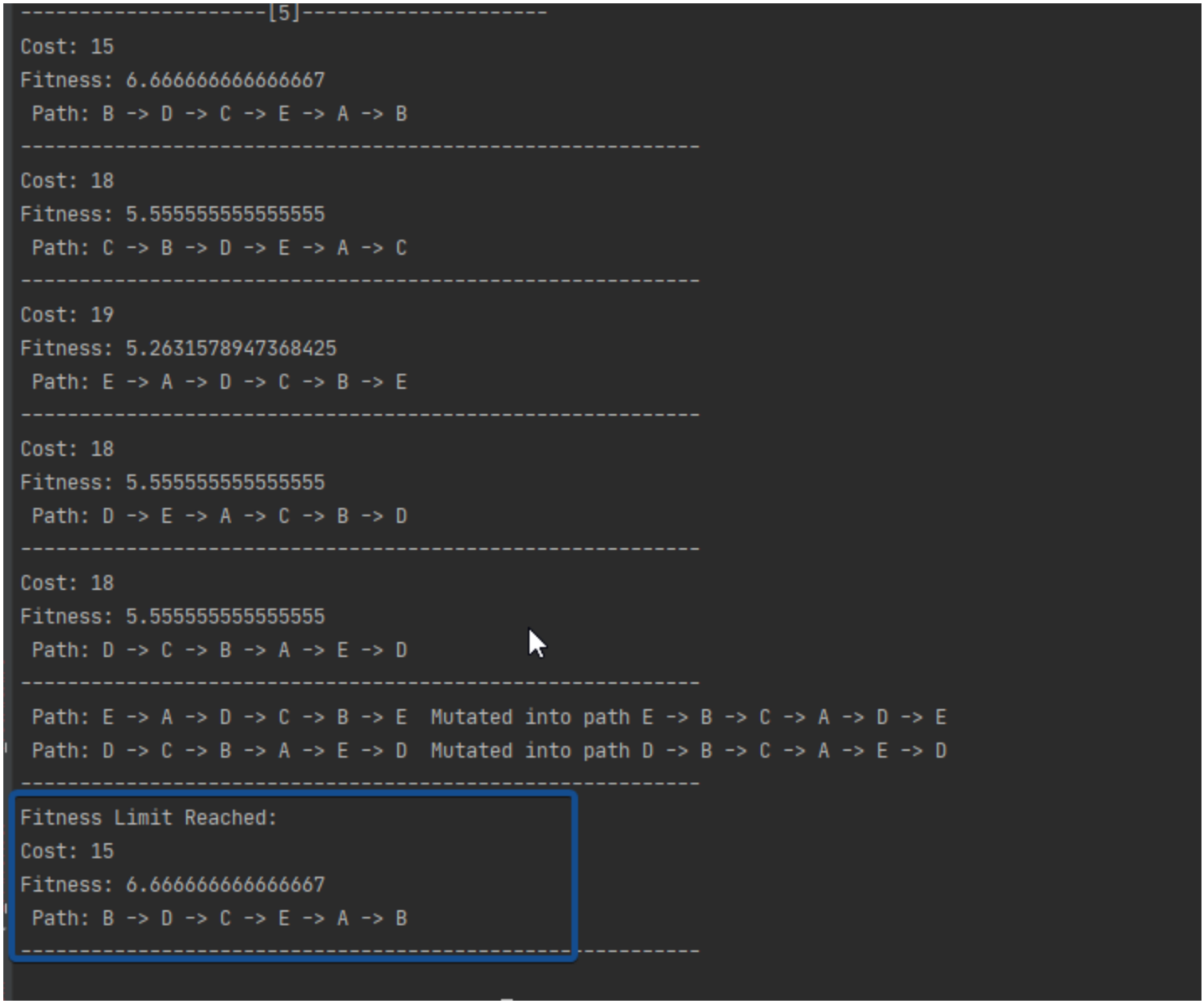
```
runGeneticAlgorithm( howManyPaths: 5, FitnessLimit: 6)
```

```
let mutationChance = 40
let crossOverChance = 10
```

```
----------------------[1]----------------------
Cost: 22
Fitness: 4.545454545454546
 Path: C -> A -> B -> E -> D -> C
------------------------------------------------------------
Cost: 20
Fitness: 5
 Path: C -> A -> B -> D -> E -> C
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: E -> A -> D -> B -> C -> E
------------------------------------------------------------
Cost: 20
Fitness: 5
 Path: D -> E -> C -> A -> B -> D
------------------------------------------------------------
Cost: 20
Fitness: 5
 Path: D -> E -> C -> A -> B -> D
------------------------------------------------------------
 Path: C -> A -> B -> E -> D -> C  Mutated into path C -> D -> A -> E -> B -> C
 Path: C -> A -> B -> D -> E -> C  Mutated into path C -> B -> D -> E -> A -> C
 Path: D -> E -> C -> A -> B -> D  Mutated into path D -> B -> C -> E -> A -> D
----------------------[2]----------------------
Cost: 19
Fitness: 5.2631578947368425
 Path: C -> D -> A -> E -> B -> C
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: C -> B -> D -> E -> A -> C
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: E -> A -> D -> B -> C -> E
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: D -> B -> C -> E -> A -> D
------------------------------------------------------------
Cost: 20
Fitness: 5
 Path: D -> E -> C -> A -> B -> D
```

```
----------------------[4]----------------------
Cost: 21
Fitness: 4.761904761904762
 Path: C -> E -> B -> A -> D -> C
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: C -> B -> D -> E -> A -> C
------------------------------------------------------------
Cost: 19
Fitness: 5.2631578947368425
 Path: E -> A -> D -> C -> B -> E
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: D -> E -> A -> C -> B -> D
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: D -> C -> B -> A -> E -> D
------------------------------------------------------------
Single Point cross Over [CrossPoint: 4]
Parent1: D -> E -> A -> C -> B -> D
Parent2: C -> E -> B -> A -> D -> C


Child1: D -> E -> A -> C -> B -> D
Child2: B -> D -> C -> E -> A -> B
```


----------[5]----------
Cost: 15
Fitness: 6.666666666666667
Path: B -> D -> C -> E -> A -> B
Cost: 18
Fitness: 5.555555555555555
Path: C -> B -> D -> E -> A -> C
Cost: 19
Fitness: 5.2631578947368425
Path: E -> A -> D -> C -> B -> E
Cost: 18
Fitness: 5.555555555555555
Path: D -> E -> A -> C -> B -> D
Cost: 18
Fitness: 5.555555555555555
Path: D -> C -> B -> A -> E -> D
Path: E -> A -> D -> C -> B -> E Mutated into path E -> B -> C -> A -> D -> E
Path: D -> C -> B -> A -> E -> D Mutated into path D -> B -> C -> A -> E -> D
Fitness Limit Reached:
Cost: 15
Fitness: 6.666666666666667
Path: B -> D -> C -> E -> A -> B

Παραδείγματα εκτέλεσης του προγράμματος

10 διαδρομών με πιθανότητα μετάλλαξης των πόλεων 20% και πιθανότητα διασταύρωσης ενός σημείου 20%

```
----------[1]----------
Cost: 18
Fitness: 5.555555555555555
 Path: C -> B -> D -> E -> A -> C
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: C -> E -> A -> D -> B -> C
------------------------------------------------------------
Cost: 15
Fitness: 6.666666666666667
 Path: B -> D -> C -> E -> A -> B
------------------------------------------------------------
Cost: 22
Fitness: 4.545454545454546
 Path: B -> E -> C -> A -> D -> B
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: B -> A -> E -> D -> C -> B
------------------------------------------------------------
Cost: 20
Fitness: 5
 Path: C -> A -> B -> D -> E -> C
------------------------------------------------------------
Cost: 19
Fitness: 5.2631578947368425
 Path: A -> E -> D -> B -> C -> A
------------------------------------------------------------
Cost: 18
Fitness: 5.555555555555555
 Path: B -> D -> E -> A -> C -> B
------------------------------------------------------------
Cost: 21
Fitness: 4.761904761904762
 Path: B -> A -> C -> D -> E -> B
------------------------------------------------------------
Cost: 22
Fitness: 4.545454545454546
 Path: C -> A -> D -> B -> E -> C
------------------------------------------------------------
 Path: A -> E -> D -> B -> C -> A  Mutated into path A -> D -> E -> C -> B -> A
Single Point cross Over [CrossPoint: 5]
Parent1: B -> A -> C -> D -> E -> B
Parent2: B -> A -> E -> D -> C -> B


Child1: B -> A -> C -> D -> E -> B
Child2: B -> A -> C -> E -> D -> B
------------------------------------------------------------
Fitness Limit Reached:
Cost: 15
Fitness: 6.666666666666667
 Path: B -> D -> C -> E -> A -> B
------------------------------------------------------------
```

Κώδικας Εργασίας: https://github.com/CyberGl1tch/TSP_GPA