Hibridni genetski algoritam primenjen na problem trgovačkog putnika

Profesor: Mladen Nikolić

Asistent: Stefan Mišković

Autor: Aleksandra Ilić

Problem trgovačkog putnika

- Za zadati skup gradova i zadate udaljenosti među njima potrebno je pronaći najkraći put kojim će se proći kroz sve gradove tačno jedanput i vratiti se u početni položaj
- NP težak problem
- Jedan od najčešćih problema optimizacije
- Primena: planiranje, logistika, proizvodnja mikročipova
- U ovom slučaju gradovi su predstavljeni skupom tačaka u koordinantnom sistemu, a udaljenosti medju njima euklidskim rastojanjem odgovarajućih tačaka

Genetski algoritam

- Genetski algoritam je heuristička metoda optimizacije koja imitira prirodni evolucioni proces
- Kompjuterska simulacija evolucije se prvi put primenjuje 1954. godine na Prinstonu, Nju Džersi. Simulacije su uključivale sve osnovne elemente modernih genetskih algoritama
- Popularnosti GA posebno doprinosi Holand ranih ’70- tih knjigom “*Adaptacija u prirodnim i veštačkim sistemima*”

Zadatak genetskih algoritama

- Zadatak GA je da se pretraži prostor hipoteza kako bi se pronašla najbolja hipoteza.
- Najbolja hipoteza je ona koja optimizuje unapred definisanu numeričku meru za dati problem zvanu prilagođenost hipoteze (fitness)

Osnovni koraci genetskog algoritma

```
GenerateInitialPopulation()
ComputeFitness()
While termination condition is not met
        Selection()
        Breeding()
        Mutation()
        ComputeFitness()
```

Osnovni pojmovi

- Gen – jedan grad
- Jedinka (hromozom) - jedna od mogućih putanja kroz sve čvorove
- Populacija – skup mogućih putanji
- Roditelji – dve putanje koje se kombinuju I kreiraju novu putanju
- Mating pool – skup roditelja koji se koriste da kreiraju sledeću generaciju
- Fitness – funkcija koja definiše koliko je neka putanja dobra (kratka)
- Mutacija - način da se promeni putanja menjanjem mesta dva grada u putanji
- Elitizam - način da se najbolje jedinke prenesu u sledeću generaciju

Lokalna pretraga

- Lokalna pretraga je heuristički metod koji se zasniva na poboljšavanju vrednosti jednog rešenja.
- Na početku algoritma se proizvoljno ili na neki drugi način generiše početno rešenje i izračuna vrednost njegove funkcije cilja
- Vrednost najboljeg rešenja se najpre inicijalizuje na vrednost početnog
- Zatim se algoritam ponavlja kroz nekoliko iteracija. U svakom koraku se razmatra rešenje u okolini trenutnog
- Ukoliko je vrednost njegove funkcije cilja bolja od vrednosti funkcije cilja trenutnog rešenja, ažurira se trenutno rešenje

Delovi implementacije

- Kreirana je klasa City koja sadrži x i z koordinate tačke
- distance - računa udaljenost do nekog drugog grada.
- Klasa Fitness se kreira od putanje
- RouteDistance - računa dužinu puta
- RouteFitness - računa fitness kao 1/routeDistance
- Dakle tražimo što veći fitness

- CityList sadrži 25 random tačaka
- Inicijalne putanje se kreiraju random biranjem tačaka
- def createRoute(cityList):
  route = random.sample(cityList, len(cityList))
  return route
- def initialPopulation(popSize,cityList):
- population = []
-
- for i in range(0, popSize):
- population.append(createRoute(cityList))
- return population

```
for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate, maxLocalSearchIterations)
```

Evolucija prolazi kroz 500 generacija

```
def nextGeneration(currentGen, eliteSize, mutationRate, maxLocalSearchIterations):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children, mutationRate)
    bestRouteIndex = rankRoutes(nextGeneration)[0][0]
    bestRoute = nextGeneration[bestRouteIndex]
    nextGeneration[bestRouteIndex] = localSearch(bestRoute, maxLocalSearchIterations)
    return nextGeneration
```

```
def rankRoutes(population):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] = Fitness(population[i]).routeFitness()
    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)

def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults
```

```
def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool
```

Postoji više načina kako možemo da odaberemo roditelje. Najčešći su:

- Selekcija proporcionalna fitness-u (ovde implementirana)
  - Svakoj jedinki se dodeli verovatnoća da će biti izabrana proporcionalna veličini fitness-a
- Turnir
  - Nasumično se izabere grupa jediniki I za prvog roditelja se izabere ona jedinka koja ima najveći fitness. Isto se ponovi I za drugog roditelja

Umnožavanje jedinki

```
def breedPopulation(matingpool, eliteSize):
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))

    for i in range(0,eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children
```

Mutacije

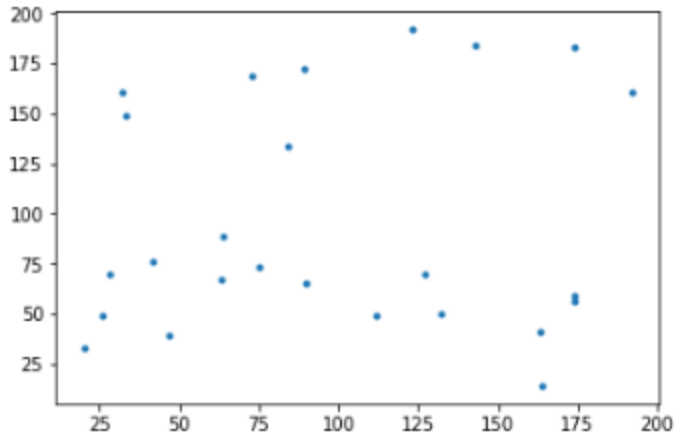
```
def mutate(individual, mutationRate):
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1
    return individual
```

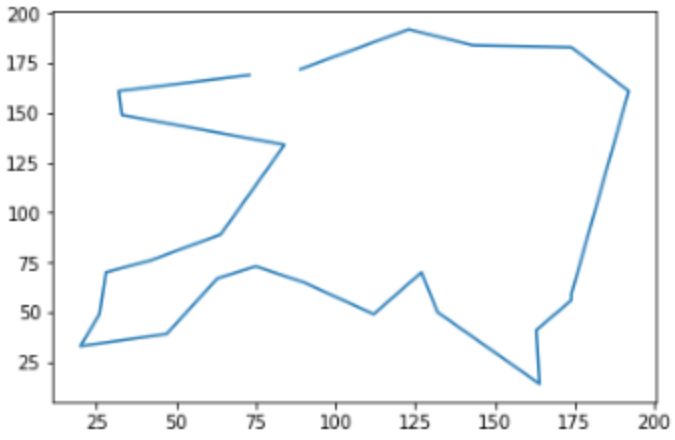
Lokalna pretraga

```
def localSearch(route, maxIterations):
    for i in range(0, maxIterations):
        start = int(random.random() * len(route))
        end = int(random.random() * len(route))
        newRoute = swap(route, start, end)
        currentFitness = Fitness(route).routeFitness()
        newFitness = Fitness(newRoute).routeFitness()
        if newFitness > currentFitness:
            route = newRoute
    return route
```

Rezultat

Initial distance: 1961.2731340962132
Final distance: 736.2063108952311





Literatura

- https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35
- Stefan Mišković, Hibridni genetski algoritam za prost lokacijski problem neograničenih kapaciteta
- Stefan Mišković, Lokalna pretraga za prost lokacijski problem neograničenih kapaciteta