ERHVERVSAKADEMI AARHUS

Opgave om TSP og GA.

Sila.

Exercise – TSP og GA.

In groups of 2- 3.

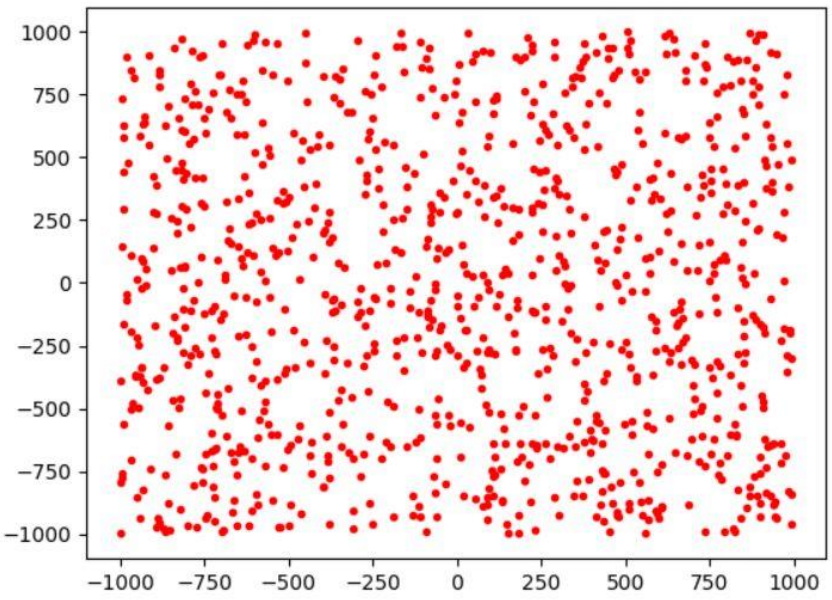
I denne opgave bliver du bedt om at se på ”The Traveling Salesman Problem” – og implementere en algoritme, der løser problemet v.hj.af en genetisk algoritme.

Der er 2 læringsmål i opgaven. Dels at arbejde selvstændigt med et lidt større projekt i Python, dels at benytte en genetisk algoritme til at løse et praktisk problem. Bemærk at en genetisk algoritme ikke nødvendigvis finder den absolut bedste løsning, men en approximation.

Traveling Salesman problemet er beskrevet her:

https://en.wikipedia.org/wiki/Travelling_salesman_problem

I denne opgave ønsker vi at finde en løsning, der besøger 1000 byer, spredt ud i et område, som angivet i dette plot.



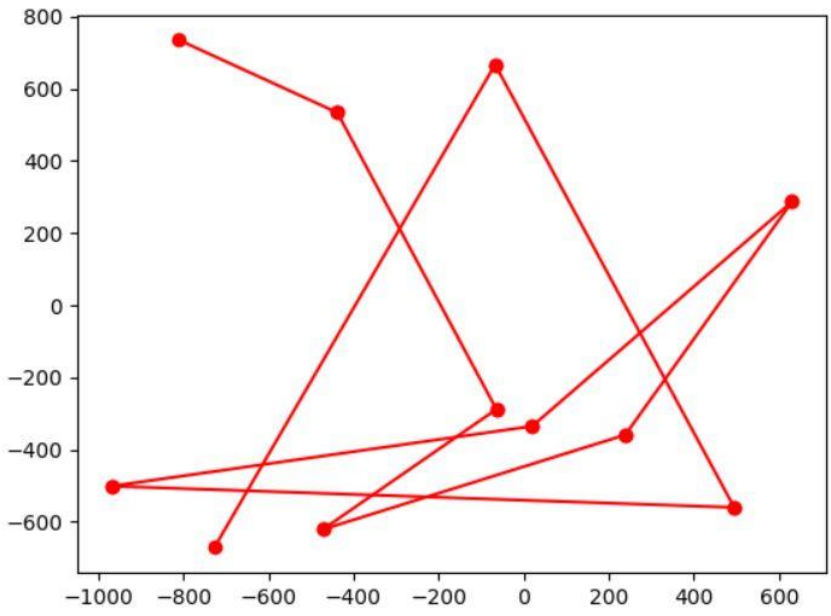
The Brute-Force Approach

The Brute Force approach, also known as the Naive Approach, calculates and compares all possible permutations of routes or paths to determine the shortest unique solution. To solve the TSP using the Brute-Force approach, you must calculate the total number of routes and then draw and list all the possible routes. Calculate the distance of each route and then choose the shortest one—this is the optimal solution.

Brute force er meget beregningstungt i dette problem. Og da vi ønsker at finde en god route der besøger 1000 byer i praksis umuligt.

Hvorfor vi her vil benytte GA, og derved komme frem til en rimelig god løsning, der besøger alle de 1000 byer.

Plot for en random rute, der besøger de 10 første byer:



Real world applications:

Despite the complexity of solving the Travelling Salesman Problem, it still finds applications in many real world scenarios.

For instance, efficient solutions found through the TSP are being used in the last mile delivery. Last mile delivery refers to the movement of goods from a transportation hub, such as a depot or a warehouse, to the end customer's choice of delivery. Last mile delivery is the leading cost driver in the supply chain. Companies usually shoulder some of the costs to better compete in the market. Where efficient solutions are very important in order to compete.

Crossover og Mutation i TSP

Da man kun må besøge en by en gang i Traveling Salesman Problemet kan GA skridtene **Crossover** og **Mutation** på en løsning (en route) være problematiske. Der er mange veje omkring problemet, hvor folk på bl.a. Github beskriver hvad de synes er den gode løsning her.

Ofte laver man **crossover** ved at tage gener fra først den ene parent, og så fylder man op med gener fra den anden parent. I.e. gener (byer) der endnu ikke er brugt (besøgt).

Koden vil typisk se ca. ud som følger – præcis hvordan kommer an på ens konkrete implementation, og en del af opgaven her er at få det til at virke.

```
def fill_with_parent1_genes(child, parent, genes_n):
    start_at = randint(0, len(parent.genes)-genes_n-1)
    finish_at = start_at + genes_n
    for i in range(start_at, finish_at):
        child.genes[i] = parent_1.genes[i]


def fill_with_parent2_genes(child, parent):
    j = 0
    for i in range(0, len(parent.genes)):
        if child.genes[i] == None:
            while parent.genes[j] in child.genes:
                j += 1
            child.genes[i] = parent.genes[j]
            j += 1
```

Tilsvarende er mutation operationen i TSP med GA ikke helt triviel. Da man igen skal være opmærksom på ikke at komme til at lave en invalid solution.

Typisk laver man en swap operation på ens løsning.

```
def mutate(route_to_mut):
    '''
    Route() --> Route()
    Swaps two random indexes in route_to_mut.route. Runs k_mut_prob*100 % of the time
    '''
    # k_mut_prob %
    if random.random() < k_mut_prob:

        # two random indices:
        mut_pos1 = random.randint(0,len(route_to_mut.route)-1)
        mut_pos2 = random.randint(0,len(route_to_mut.route)-1)

        # if they're the same, skip to the chase
        if mut_pos1 == mut_pos2:
            return route_to_mut

        # Otherwise swap them:
        city1 = route_to_mut.route[mut_pos1]
        city2 = route_to_mut.route[mut_pos2]

        route_to_mut.route[mut_pos2] = city1
        route_to_mut.route[mut_pos1] = city2
```

Igen vil den præcise implementation afhænge af ens samlede løsning.

De 2 kode fragmenter findes i filen Snippets_TSP_Crosover_Mutation.txt på Canvas. Som man så evt. selv kan arbejde videre på?

Kode

Der ønskes nu en løsning af TSP problemet for de byer der er angivet i filen TSPcities1000.txt på Canvas.

I filen TSP.py (Canvas) er der givet kode, der læser data ind fra filen (TSPcities1000.txt), og laver plots af byer og ruter, samt skabelon for plot af progress af den genetiske algoritme.

Distance mellem 2 byer ønskes beregnet ved denne formel:

```
# calculate distance between cities
def distancebetweenCities(city1x, city1y, city2x, city2y):
    xDistance = abs(city1x-city2x)
    yDistance = abs(city1y-city2y)
    distance = math.sqrt((xDistance * xDistance) + (yDistance * yDistance))
    return distance
```

Hvor man så selv skal summere over en hel rute. Hvilket forventes at være en del af en ens fitness funktion, hvor lavere fitness (længde af rute) er bedre.

Opgaven

Det er nu en del af opgaven at vælge populations størrelse for den genetiske algoritme, mutation_rate, antal af generationer den genetiske algoritme skal køre mm. Og derefter at kode en løsning i Python.

Det er helt legitimt at søge inspiration på internettet, inklusivt at finde kode der hjælper én på vej.

Submit

Denne opgave er en afleveringsopgave i kurset (men ikke en af de obligatoriske opgaver).

Når man afleverer skal man derfor aflevere dels ens Python kode, samt en lille beskrivelse af ens genetiske algoritme. Dvs. hvad har man lavet selv, hvad har man fundet på internettet, hvad er størrelsen af ens population, mutation_rate mm. ***Meget gerne en liste over de ideer du har prøvet.*** Endelig en beskrivelse af hvilke resultater du har opnået.

I beskrivelses teksten skal det også fremgå hvem der er i den gruppe der har afleveret den pågældende opgave, eller om man har arbejdet alene på denne løsning. Disse to filer zippes sammen til en project.zip fil, som afleveres på Canvas.