**Kelompok 6:**

Aldo Jonathan Handaka (2201736971)

Alim Raharjo (2201748164)

Edward (2201741971)

Raffael Lucas Tatulus (2201742356)

Richard Geovani (2201734096)

1. **Genetic Algorithm Definition**

Genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection .

1. **Genetic Algorithm Flow Chart**

Diagram

Description automatically generated

1. **Genetic Algorithm Steps**
2. **Initial Population**

A population is a group of individuals, and each individual is a candidate solution to The problem.

An individual contains a set of parameters known as Genes. those Genes form called Chromosome.

1. **Genes**

**from** math **import** sqrt **,** **pow**

**class** **city:**

**def** \_\_init\_\_**(**self**,id,**x**,**y**):**

self**.id** **=** **id**

self**.**x **=** x

self**.**y **=** y

**def** \_\_str\_\_**(**self**):**

**return** "city {" **+** **str(**self**.id)** **+** " , " **+** **str(**self**.**x**)** **+**" , " **+** **str(**self**.**y**)** **+** "}"

**def** getDistance**(**self**,**c**):**

**if** self **!=** **None** **and** c **!=** **None** **:**

distance **=** sqrt**(** **pow(**self**.**x **-** c**.**x **,** 2**)** **+** **pow(**self**.**y **-** c**.**y **,** 2**))**

**return** distance

1. **Chromosome**

**class** **tour** **:**

**def** \_\_init\_\_**(**self **,** init**):**

self**.**nbrCities **=** getNbrCities**()**

self**.**cities **=** **[None]** **\*** self**.**nbrCities

**if** init **:**

self**.**initTour**()**

self**.**fitness **=** self**.**calculateFitness**()**

**def** \_\_str\_\_**(**self**):**

path **=** ""

**for** i **in** **range(**self**.**nbrCities**):**

**if** i **!=** self**.**nbrCities **-**1 **:**

path **+=** **str(**self**.**cities**[**i**].id)** **+** " -> "

**else** **:**

path **+=** **str(**self**.**cities**[**i**].id)** **+** " . "

**return** path

**def** initTourWithNone**(**self**):**

**for** i **in** **range(**self**.**nbrCities**):**

self**.**cities**.**append**(None)**

**def** initTour**(**self**):**

**for** i **in** **range(**self**.**nbrCities**):**

city **=** getRandomCity**()**

**while** self**.**contain**(**city**)** **:**

city **=** getRandomCity**()**

self**.**cities**[**i**]** **=** city

**def** contain**(**self**,**city**):**

**for** i **in** **range(**self**.**nbrCities**):**

**if** self**.**cities**[**i**]** **!=** **None** **:**

**if** self**.**cities**[**i**].id** **==** city**.id** **:**

**return** **True**

**return** **False**

**def** getIndexOf**(**self**,**city**):**

**for** i **in** **range(**self**.**nbrCities**):**

**if** self**.**cities**[**i**]** **==** city**.id** **:**

**return** i

**return** **-**1

**def** calculateFitness**(**self**):**

self**.**fitness **=** 0

**for** i **in** **range(**self**.**nbrCities **-**1**):**

self**.**fitness **+=** self**.**cities**[**i**].**getDistance**(**self**.**cities**[**i**+**1**])**

self**.**fitness **+=** self**.**cities**[len(**self**.**cities**)** **-**1**].**getDistance**(**self**.**cities**[**0**])**

**def** compare**(**self**,** other**):**

**return** 1 **if** self**.**fitness **<** other**.**fitness **else** **-**1

1. **Population**

**class** **population** **:**

**def** \_\_init\_\_**(**self**,**populationSize**,**init**):**

self**.**popSize **=** populationSize

self**.**tours **=** **[None]** **\*** self**.**popSize

**if** init **:**

self**.**initPopulation**()**

self**.**calculateFitnessForAll**()**

self**.**sortPopulation**()**

self**.**fittest **=** self**.**tours**[**0**]**

**def** initPopulation**(**self**):**

**for** i **in** **range(**self**.**popSize**):**

self**.**tours**[**i**]** **=** tour**(True)**

**def** initPopulationWithNone**(**self**):**

**for** i **in** **range(**self**.**popSize**):**

self**.**tours**.**append**(None)**

**def** sortPopulation**(**self**):**

**for** i **in** **range(**self**.**popSize**-**1**):**

index **=** i

**for** j **in** **range(**i**+**1 **,** self**.**popSize**):**

**if** self**.**tours**[**j**].**compare**(**self**.**tours**[**i**])** **>** 0 **:**

index **=** j

tmp **=** self**.**tours**[**i**]**

self**.**tours**[**i**]** **=** self**.**tours**[**index**]**

self**.**tours**[**index**]** **=** tmp

**def** getFittestTour**(**self**):**

**return** self**.**fittest

**def** getNFittestTour**(**self**,**n**):**

**return** self**.**tours**[:**n**]**

**def** calculateFitnessForAll**(**self**):**

**for** i **in** **range(**self**.**popSize**):**

self**.**tours**[**i**].**calculateFitness**()**

1. **Calculate Fitness Function**

A fitness function is a particular type of objective function which takes as input a candidate solution and outputs the quality of this solution, therefore the fitness function makes it possible to evaluate the candidate solutions.

**def** calculateFitness**(**self**):**

self**.**fitness **=** 0

**for** i **in** **range(**self**.**nbrCities **-**1**):**

self**.**fitness **+=** self**.**cities**[**i**].**getDistance**(**self**.**cities**[**i**+**1**])**

self**.**fitness **+=** self**.**cities**[len(**self**.**cities**)** **-**1**].**getDistance**(**self**.**cities**[**0**])**

1. **Do Selection Methods**

Selection is the process of selecting parents to generate the next generation.

**Elitism Selection**

**def** elitismSelection**(**self**,**pop**):**

pop**.**sortPopulation**()**

elitismSubPopulation **=** pop**.**tours**[:**self**.**elitismSize **+** 1**]**

**return** elitismSubPopulation

**Tournament Selection**

**def** touranmentSelection**(**self**,**pop**):**

pool **=** **[None]** **\*** self**.**poolSize

**for** i **in** **range(**self**.**poolSize**):**

index **=** random**.**randint**(**0**,**self**.**popSize **-**1**)**

pool**[**i**]** **=** pop**.**tours**[**index**]**

self**.**sortSubPopulation**(**pool**)**

**return** pool**[**0**]**

1. **Do CrossOver Methods**

The crossing operation is the reproduction of new chromosomes from the parent chromosomes.

**def** Ox1CrossOver**(**self**,**parent1**,**parent2**):**

child **=** tour**(False)**

start **=** random**.**randint**(**0**,**parent1**.**nbrCities**)**

end **=** random**.**randint**(**0**,**parent1**.**nbrCities**)**

**while** start **>=** end **:**

start **=** random**.**randint**(**0**,**parent1**.**nbrCities**)**

end **=** random**.**randint**(**0**,**parent1**.**nbrCities**)**

**for** i **in** **range(**start**,**end**):**

child**.**cities**[**i**]** **=** parent1**.**cities**[**i**]**

**for** i **in** **range(**parent2**.**nbrCities**):**

**if** **not** child**.**contain**(**parent2**.**cities**[**i**])** **:**

**for** j **in** **range(**parent2**.**nbrCities**):**

**if** child**.**cities**[**j**]** **is** **None** **:**

child**.**cities**[**j**]** **=** parent2**.**cities**[**i**]**

**break**

**return** child

1. **Do Mutation Methods**

The mutation can be defined as a small random modification of the chromosome, to obtain a new solution. It is used to maintain and introduce diversity in the genetic population and is generally applied with a low probability Pm.

**Swap Mutation**

**def** SwapMutation**(**self**,**child**):**

**for** i **in** **range(**child**.**nbrCities**):**

mutationProbability **=** random**.**random**()**

**if** mutationProbability **<** self**.**mutationRate **:**

mutationPoint **=** random**.**randint**(**0 **,** child**.**nbrCities **-**1**)**

tmp **=** child**.**cities**[**mutationPoint**]**

child**.**cities**[**mutationPoint**]** **=** child**.**cities**[**i**]**

child**.**cities**[**i**]** **=** tmp

1. **Do Reproduction if doesn’t meet the Criteria**

**def** reproduction**(**self**,**pop**):**

newpop **=** population**(**pop**.**popSize**,False)**

elitismSubPopulation **=** self**.**elitismSelection**(**pop**)**

**for** index **in** **range(**self**.**elitismSize**):**

newpop**.**tours**[**index**]** **=** elitismSubPopulation**[**index**]**

**for** i **in** **range(**index **,** pop**.**popSize**):**

parent1 **=** self**.**touranmentSelection**(**pop**)**

parent2 **=** self**.**touranmentSelection**(**pop**)**

child **=** self**.**Ox1CrossOver**(**parent1**,** parent2**)**

self**.**SwapMutation**(**child**)**

child**.**calculateFitness**()**

newpop**.**tours**[**i**]** **=** child

newpop**.**calculateFitnessForAll**()**

newpop**.**sortPopulation**()**

newpop**.**fittest **=** newpop**.**tours**[**0**]**

**return** newpop

1. **Genetic Algorithm**

**def** \_\_init\_\_**(**self**,**nbrGenerations**,**popSize**,**elitismSize**,**poolSize**,**mutationRate**):**

self**.**nbrGenerations **=** nbrGenerations

self**.**popSize **=** popSize

self**.**elitismSize **=** elitismSize

self**.**poolSize **=** poolSize

self**.**mutationRate **=** mutationRate

self**.**initialPopulation **=** population**(**self**.**popSize **,** **True)**

self**.**fitnesses **=** np**.**zeros**(**self**.**nbrGenerations**)**

**print(**"Initial Fitness : " **,** self**.**initialPopulation**.**fittest**.**fitness**)**

**print(**"Best Tour : "**,**self**.**initialPopulation**.**fittest**)**

newPopulation **=** self**.**initialPopulation

generationCounter **=** 0

**for** i **in** **range(**self**.**nbrGenerations**):**

newPopulation **=** self**.**reproduction**(**newPopulation**)**

self**.**fitnesses**[**generationCounter**]** **=** newPopulation**.**fittest**.**fitness

generationCounter **+=** 1

**print(**"Generation : "**,** generationCounter **)**

**print(**"Fitness : "**,** newPopulation**.**fittest**.**fitness**)**

**print(**"Best Tour : "**,**newPopulation**.**fittest**)**

**print(**"\n\n"**)**

1. **Result of berlin52 Dataset Using Genetic Algorithm**

Application

Description automatically generated with low confidence

Chart, histogram

Description automatically generated

**References:**

<https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>

1. **Ant Colony Optimization Definition**

In the natural, an ant having ability or capability to find their food form nest in best way and some of the ants doesn’t have eyes, though they can communicate with other by using their smelling sense.

Usually the ants has organs which disperse pheromone. Basically, there are 10 to 20 types of pheromones presents and all these different pheromones smell convey different massage and ant are familiar with it.

When Ants are in search for food randomly in their colony, and upon finding food return to their colony while laying down pheromone trails. So, the only way is to communicate with other ants by releasing pheromone trails in their path.

As pheromone trails is chemical evaporate over a time, Thus its attractive strength decreases. So, ant which will follow the longer route to travel down and come back again, the more time the pheromone trails have to evaporate.

A short path, by comparison, gets followed over more frequently, and thus the pheromone density become higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution Thus path will be chosen by the ants which has the more pheromone level density. subsequently they find the optimal solution.

The overall result is that when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to many ants following a single path.

Ant Colony Optimization (ACO) was first introduced by Marco Dorigo in the 90s in his Ph.D. thesis. This algorithm is introduced based on the foraging behavior of an ant for seeking a path between their colony and source food.

1. **Ant Colony Optimization Pseudocode**

Graphical user interface, text, application

Description automatically generated

1. **Ant Colony Optimization Code**
2. **Read Dataset**

**import** numpy **as** np

**from** numpy **import** inf

**import** matplotlib**.**pyplot **as** plt

locations **=** **[]**

f **=** **open(**"berlin52Aco.txt"**,** "r+"**)**

u **=** f

**for** line **in** u**:**

line **=** line**.**split**(**" "**,** 1**)[**1**]**

y **=** line**.**split**()**

y**[**0**]** **=** **float(**y**[**0**])**

y**[**1**]** **=** **float(**y**[**1**])**

locations**.**append**(**y**)**

**print(**locations**)**

dist **=** np**.**zeros**((**52**,**52**))**

**for** i**,**city **in** **enumerate(**locations**):**

**for** j**,**city2 **in** **enumerate(**locations**):**

dist**[**i**][**j**]** **=** np**.**linalg**.**norm**(**np**.**array**(**city**)-**np**.**array**(**city2**))**

d **=** dist

1. **Initialize City, Ants, Pheromone, Alpha, Beta**

iteration **=** 1000

n\_city **=** 52

n\_ants **=** 52

total\_p **=** 1

pheromone\_per\_path **=** total\_p**/**d

pheromone\_per\_path**[**pheromone\_per\_path **==** inf**]** **=** 0

**print(**pheromone\_per\_path**)**

alpha **=** 0.85

beta **=** 1.7

ini\_p **=** np**.**ones**((**n\_ants**,** n\_city**))**

rute **=** np**.**ones**((**n\_ants**,** n\_city**+**1**))**

run **=** 0

bestp **=** **[]**

1. **Ant Colony Optimization Result for berlin52 Dataset**

**while** run **<** 1**:**

m **=** n\_ants

n **=** n\_city

pheromne **=** ini\_p

e **=** .2

alpha **=** 1

beta **=** 1.8

visibility **=** pheromone\_per\_path

**for** ite **in** **range(**iteration**):**

rute**[:,**0**]** **=** 1

**for** i **in** **range(**m**):**

temp\_visibility **=** np**.**array**(**visibility**)**

**for** j **in** **range(**n**-**1**):**

#print(rute)

combine\_feature **=** np**.**zeros**(**52**)**

cum\_prob **=** np**.**zeros**(**52**)**

cur\_loc **=** **int(**rute**[**i**,**j**]-**1**)**

# print("cur\_loc",cur\_loc)

temp\_visibility**[:,**cur\_loc**]** **=** 0

# print(temp\_visibility)

# print("pheromne",pheromne[cur\_loc,:])

p\_feature **=** np**.**power**(**pheromne**[**cur\_loc**,:],**beta**)**

v\_feature **=** np**.**power**(**temp\_visibility**[**cur\_loc**,:],**alpha**)**

p\_feature **=** p\_feature**[:,**np**.**newaxis**]**

v\_feature **=** v\_feature**[:,**np**.**newaxis**]**

combine\_feature **=** np**.**multiply**(**p\_feature**,**v\_feature**)**

total **=** np**.sum(**combine\_feature**)**

probs **=** combine\_feature**/**total

cum\_prob **=** np**.**cumsum**(**probs**)**

#print(cum\_prob)

r **=** np**.**random**.**random\_sample**()**

#print(r)

city **=** np**.**nonzero**(**cum\_prob**>**r**)[**0**][**0**]+**1

#print(city)

rute**[**i**,**j**+**1**]** **=** city

left **=** **list(set([**i **for** i **in** **range(**1**,**n**+**1**)])-set(**rute**[**i**,:-**2**]))[**0**]**

rute**[**i**,-**2**]** **=** left

rute\_opt **=** np**.**array**(**rute**)**

dist\_cost **=** np**.**zeros**((**m**,**1**))**

**for** i **in** **range(**m**):**

s **=** 0

**for** j **in** **range(**n**-**1**):**

s **=** s **+** d**[int(**rute\_opt**[**i**,**j**])-**1**,int(**rute\_opt**[**i**,**j**+**1**])-**1**]**

dist\_cost**[**i**]=**s

dist\_min\_loc **=** np**.**argmin**(**dist\_cost**)**

dist\_min\_cost **=** dist\_cost**[**dist\_min\_loc**]**

best\_route **=** rute**[**dist\_min\_loc**,:]**

pheromne **=** **(**1**-**e**)\***pheromne

**for** i **in** **range(**m**):**

**for** j **in** **range(**n**-**1**):**

dt **=** 1**/**dist\_cost**[**i**]**

pheromne**[int(**rute\_opt**[**i**,**j**])-**1**,int(**rute\_opt**[**i**,**j**+**1**])-**1**]** **=** pheromne**[int(**rute\_opt**[**i**,**j**])-**1**,int(**rute\_opt**[**i**,**j**+**1**])-**1**]** **+** dt

best\_dist **=** **int(**dist\_min\_cost**[**0**])** **+** d**[int(**best\_route**[-**2**])-**1**,**0**]**

bestp**.**append**(**best\_dist**)**

**print(**'route of all the ants at the end :'**)**

**print(**rute\_opt**)**

**print()**

**print(**'best path :'**,**best\_route**)**

**print(**'cost of the best path'**,int(**dist\_min\_cost**[**0**])** **+** d**[int(**best\_route**[-**2**])-**1**,**0**])**

plt**.**plot**(**bestp**)**

f **=** **open(**"berlin\_52\_E.txt"**,** "a"**)**

f**.**write**(**"Run {}/{} Length of best path: {}\n"**.format(**run**,** 10**,int(**dist\_min\_cost**[**0**])** **+** d**[int(**best\_route**[-**2**])-**1**,**0**]))**

run **+=**1

1. **Result of berlin52 Dataset Using Ant Colony Optimization**

Graphical user interface, application

Description automatically generated

**References:**

<https://towardsdatascience.com/the-inspiration-of-an-ant-colony-optimization-f377568ea03f>