Solving The Travelling Salesmen Problem

FLOCKS, HERDS & SCHOOLS BEHAVIOR MODEL

Aditya Iyer
Aaryan Choksi
Willam Zhang
Emily Dashefsky

Mathematics

Capstone

Table Of Contents

Overview

What is the travelling salesmen problem?

Overview

What is the travelling salesmen problem?

Travelling Salesman Problem (TSP)

and its applications

Overview

What is the travelling salesmen problem?

Travelling Salesman Problem (TSP)

and its applications

"Imagine you are a merchant travelling to 15 different cities. Which path would you take to reach all the cities and return to your home in the shortest distance?"

Overview

What is the travelling salesmen problem?

Travelling Salesman Problem (TSP)

and its applications

"Imagine you are a merchant travelling to 15 different cities. Which path would you take to reach all the cities and return to your home in the shortest distance?"

TRY IT YOURSELF !!

With The Paper Provided To You Draw Out The Most Optimal Route For The Salesmen To Go To All Cities Once and Return To The Point Of Origin.

Emily

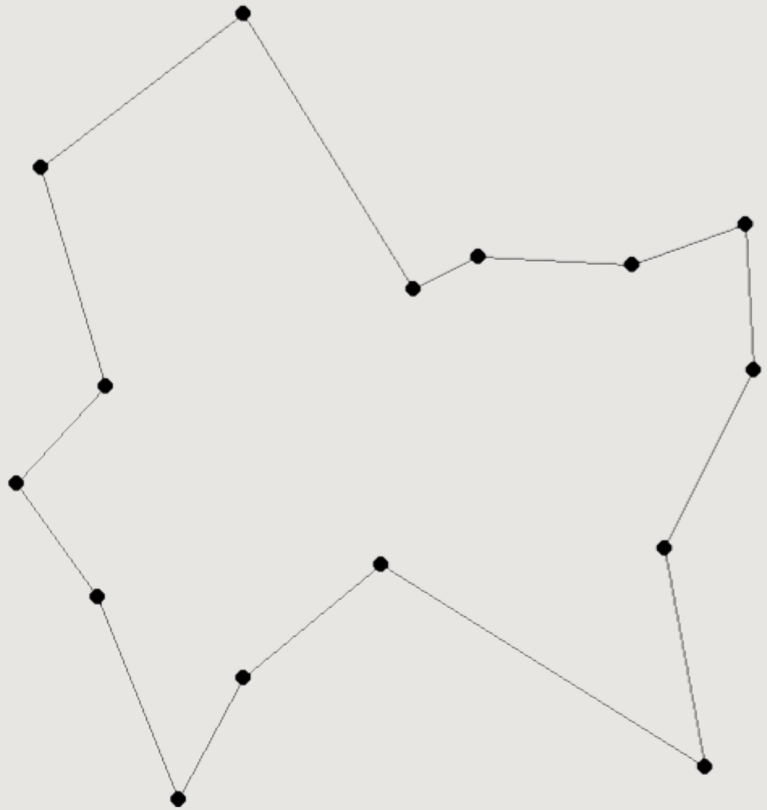
Overview

What is the travelling salesmen problem?

Travelling Salesman Problem (TSP)

and its applications

"Imagine you are a merchant travelling to 15 different cities. Which path would you take to reach all the cities and return to your home in the shortest distance?"



While humans have traditionally used cognitive thinking to solve the Traveling Salesman Problem (TSP) by intuitively finding the most optimal route, the problem becomes significantly more challenging when computed by a computer. Unlike humans, computers typically rely on brute force algorithms to solve the TSP, which involves checking all possible combinations of routes. As the number of cities increases, the computational complexity grows exponentially, making it increasingly difficult and time-consuming for a computer to find the optimal solution.

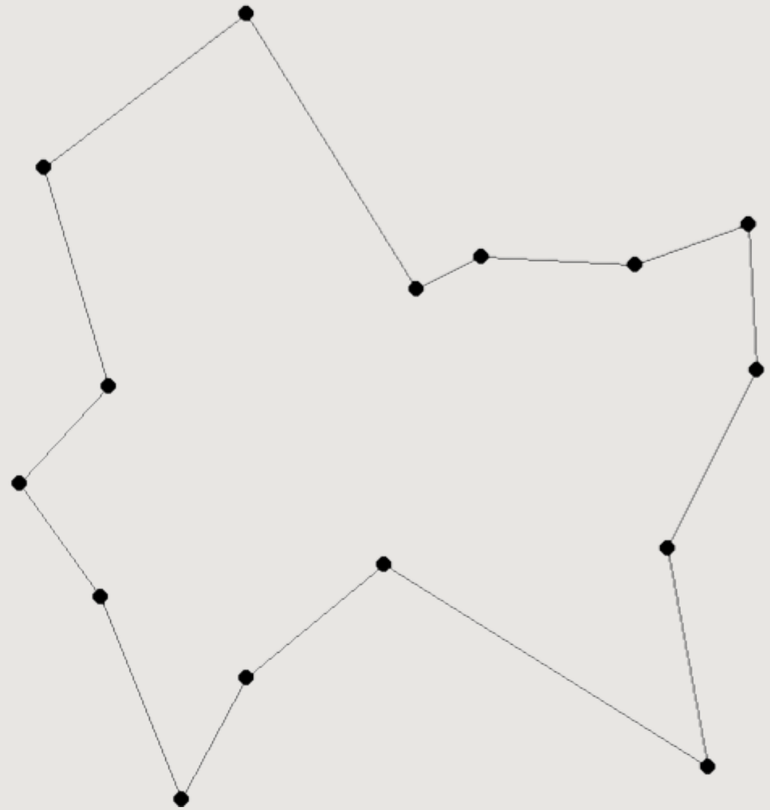
Aaryan

Overview

What is the travelling salesmen problem?

Key Terms

Nodes - Cities that fall in TSP
Edges - Paths/Routes with a certain weight (distance)
Factorial - Product of a natural number and all-natural number below it
O(?) - Time Complexity of an algorithm i.e. how the number of ticks increases with the size of data
(n-1)! /2- Number of possible routes in a perfect TSP



William

Overview

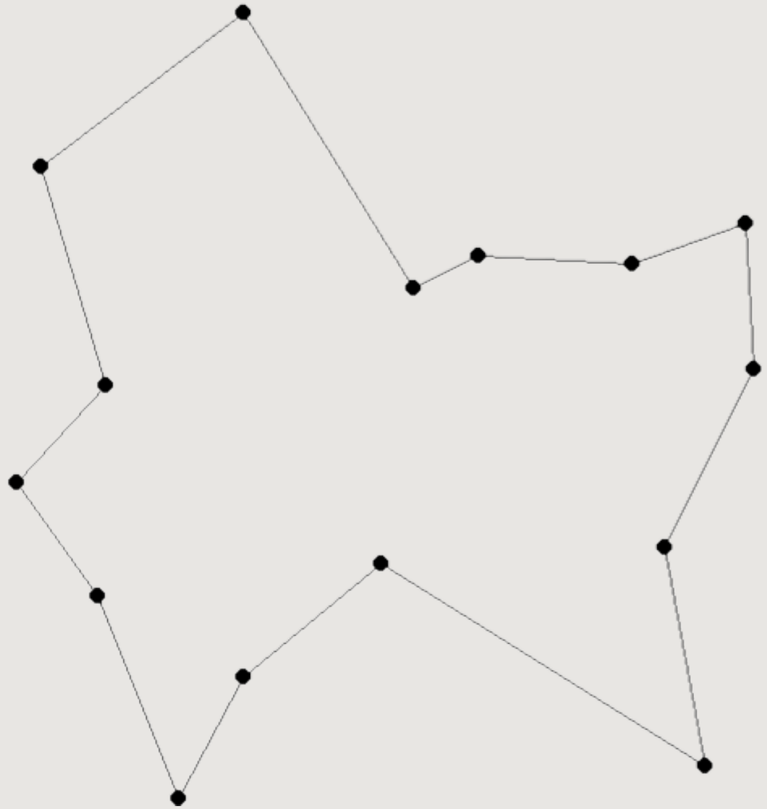
What is the travelling salesmen problem?

Travelling Salesman Problem (TSP)

and its applications

Math Behind The TSP - Lit Review 1

The Traveling Salesman Problem: Deceptively Easy to State; Notoriously Hard to Solve



Overview

What is the travelling salesmen problem?

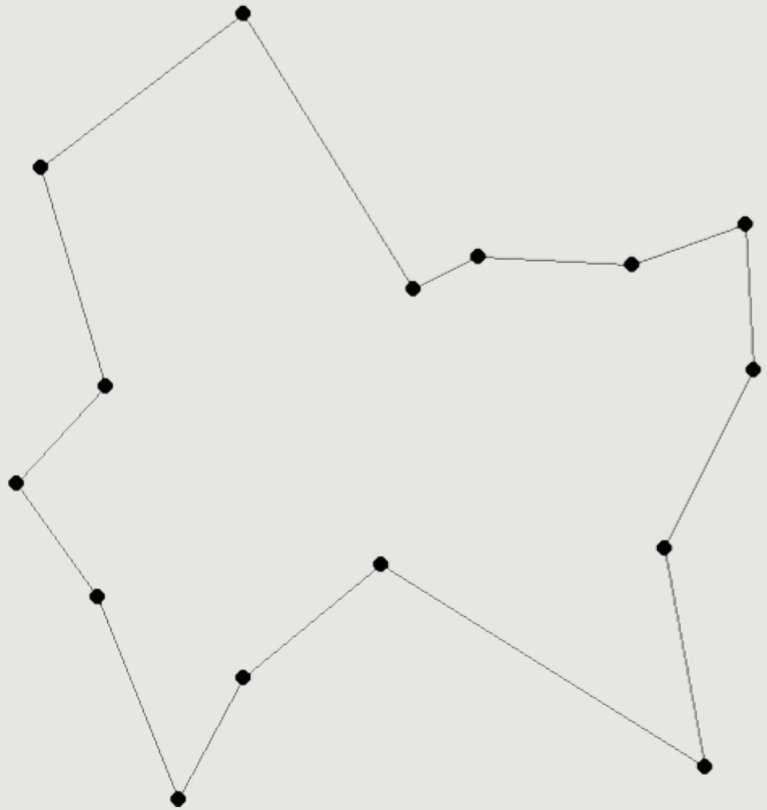
Travelling Salesman Problem (TSP)

and its applications

Math Behind The TSP - Lit Review 1

The Traveling Salesman Problem: Deceptively Easy to State; Notoriously Hard to Solve

This paper covers various methods of solving The Traveling Salesman Problem (TSP) The complexity arises from the factorial growth of possible routes as the number of cities increases. The mathematical notation "n!" denotes the factorial of a number, representing the total number of routes to consider. The expression (n-1)!/2 estimates the number of distinct routes by accounting for the problem's symmetry. These mathematical concepts help in understanding and approximating solutions for the TSP.



Aditya

Overview

What is the travelling salesmen problem?

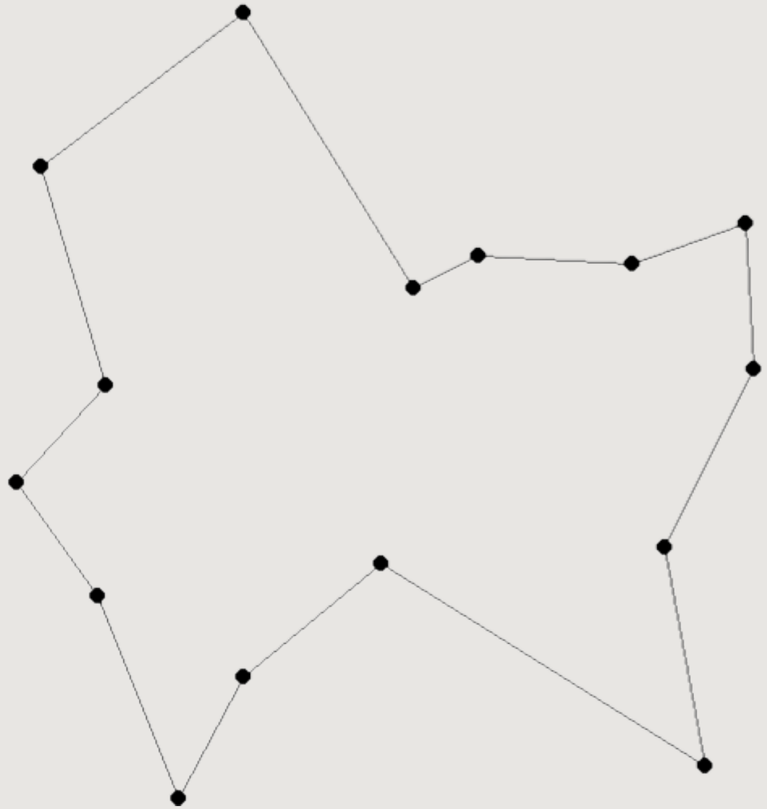
Travelling Salesman Problem (TSP)

and its applications

Applications Of TSP

The Travelling Salesmen Problem has various applications across multiple fields. In its pure form, TSP is used in planning, logistics, and even the manufacturing of microchips. Modified versions of this problem show up in fields such as DNA sequencing and computer science.

With the application of optimised TSP, many fields would be revolutionised. For example, chips would be able to store large amounts of data without waste of excess space.



Aaryan

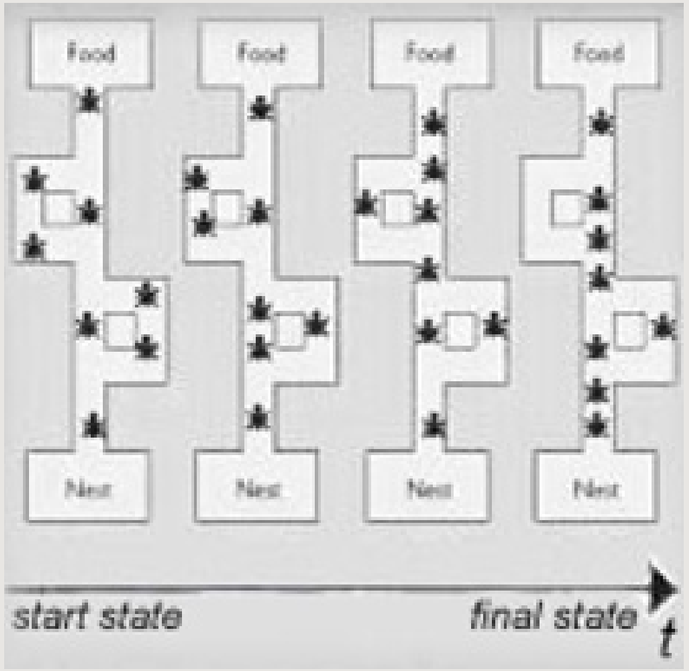
Conceptual Basis Behind The ACO

Conceptual Basis Behind The ACO

- What is Ant Colony Optimization (ACO)
- The biological basis for ACO
- How ACO is better than Brute Force
- Limitations Of ACO

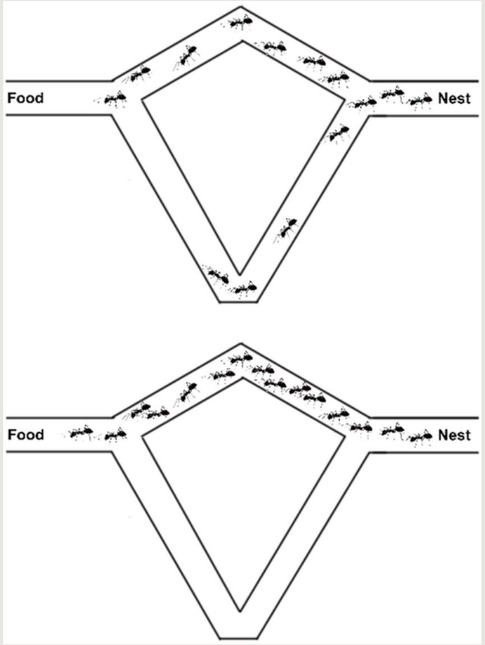
Conceptual Basis Behind The ACO

- What is Ant Colony Optimization (ACO)
- The biological basis for ACO
- How ACO is better than Brute Force
- Limitations Of ACO



What is Ant Colony Optimisation Algorithm (ACO)

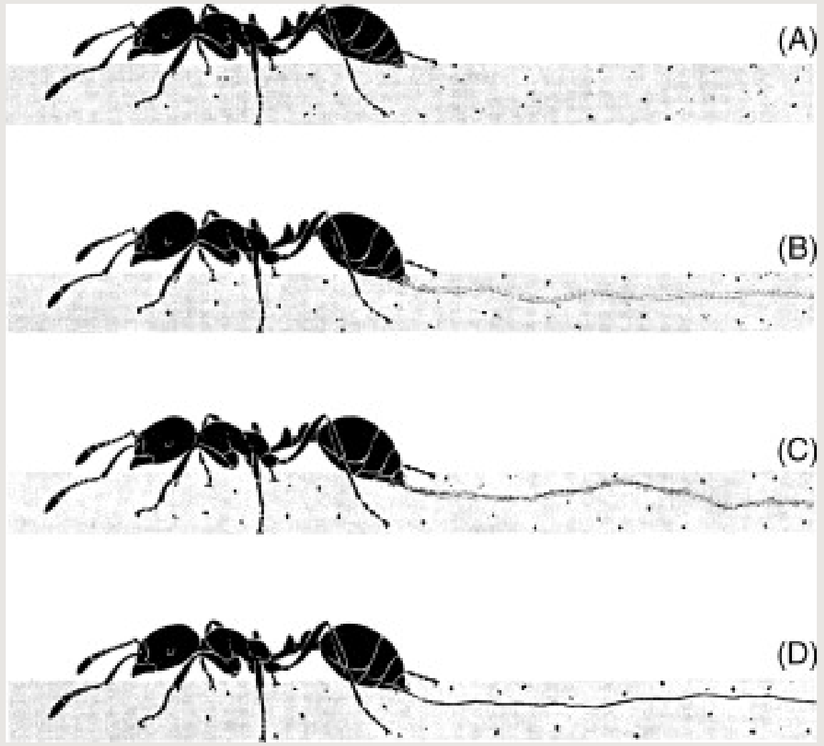
- proposed by Marco Dorigo
- a branch of Swarm Intelligence Methods
- used to find good paths in graphs
- inspired by the behavior of real ants
- simulated ants will record the quality of solutions
- more ants locate better solutions
- the Double Bridge experiment



William

Conceptual Basis Behind The ACO

- What is Ant Colony Optimization (ACO)
- The biological basis for ACO
- How ACO is better than Brute Force
- Limitations Of ACO



The biological basis for ACO: Pheromone

- a secreted chemical factor that triggers a social response in members of the same species
- trail pheromone: leads members of its own species towards a food source
- can evaporate easily
- after food is depleted, the pheromone trail decays

william

Conceptual Basis Behind The ACO

- What is Ant Colony Optimization (ACO)
- The biological basis for ACO
- How ACO is better than Brute Force
- Limitations Of ACO

How ACO is better than Brute Force.

Conceptual Basis Behind The ACO

- What is Ant Colony Optimization (ACO)
- The biological basis for ACO
- How ACO is better than Brute Force
- Limitations Of ACO

How ACO is better than Brute Force.

- **Efficiency:** ACO is more efficient than brute force for solving TSP due to its focused exploration of the solution space.
- **Scalability:** ACO exhibits better scalability, making it suitable for larger TSP instances where brute force becomes impractical.
- **Near-Optimal Solutions:** ACO can find near-optimal solutions by utilizing pheromone trails and iterative improvements, providing high-quality results with less computational effort

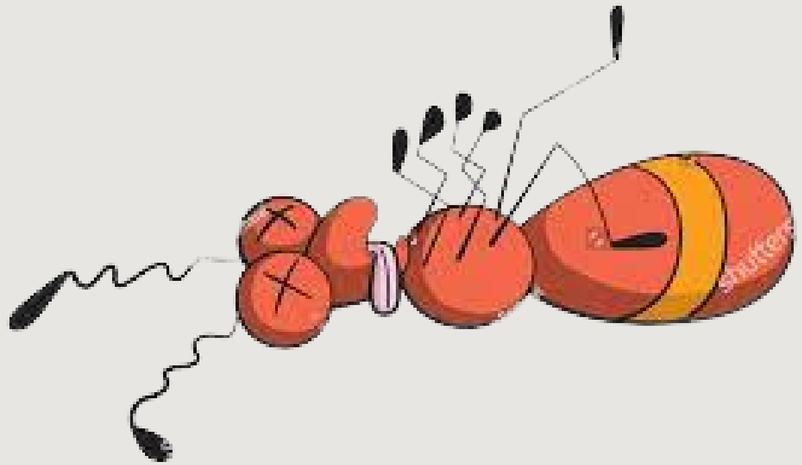
aaryan

Conceptual Basis Behind The ACO

- What is Ant Colony Optimization (ACO)
- The biological basis for ACO
- How ACO is better than Brute Force
- Limitations Of ACO

Limitations Of ACO

- **Parameter sensitivity**: ACO's performance is sensitive to parameter choices, requiring careful tuning to achieve optimal results.
- **Local optima trapping:** ACO can struggle to escape local optima, limiting its ability to find the global optimal solution in complex TSP instances.
- **Problem dependency:** ACO's effectiveness varies depending on the characteristics of the TSP problem, potentially leading to suboptimal performance in certain scenarios.



Aditya

Research Question

Can Craig W. Reynolds' Flock Herds and Schools Behaviour model applied to Ant Colony Optimization (ACO) algorithms improve the convergence speed in solving the Traveling Salesman Problem (TSP)?

Research Question

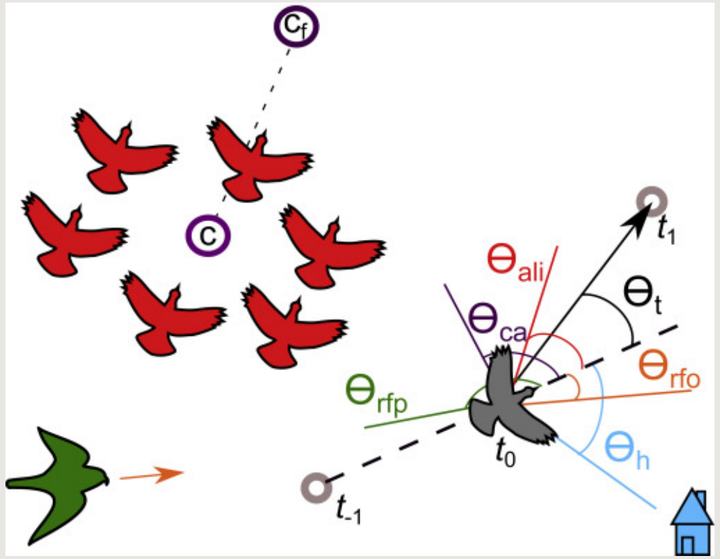
Can Craig W. Reynolds' Flock Herds and Schools Behaviour model applied to Ant Colony Optimization (ACO) algorithms improve the convergence speed in solving the Traveling Salesman Problem (TSP)?

Flock Herds and Schools Behaviour Model

Boid Theory - Lit Review Flocks, Herds, and Schools: A Distributed Behavioral Model by Craig W. Reynolds

Rule 1: Collision Avoidance - Avoid Collisions With Nearby Flockmates
Rule 2: Velocity Matching - Attempt to match velocity with nearby flockmates
Rule 3: Flock Centering - Attempt to stay close to nearby flockmates
These rules simulate the flocking patterns of birds which are used to render multi-organism groups in video games. This concept can be applied to Ant Colony Optimization Algorithm to eliminate unfavourable solutions and decrease ACO's reliance on pure randomness.

Emily



Hypothesis

Integrating analogues of the three Boid rules in the Ant Colony Optimization Algorithm will reduce the average number of ticks required to find the best path by up to 85%.

Emily

Hypothesis

Integrating analogues of the three Boid rules in the Ant Colony Optimization Algorithm will reduce the average number of ticks required to find the best path by up to 85%.

What are Game Ticks?

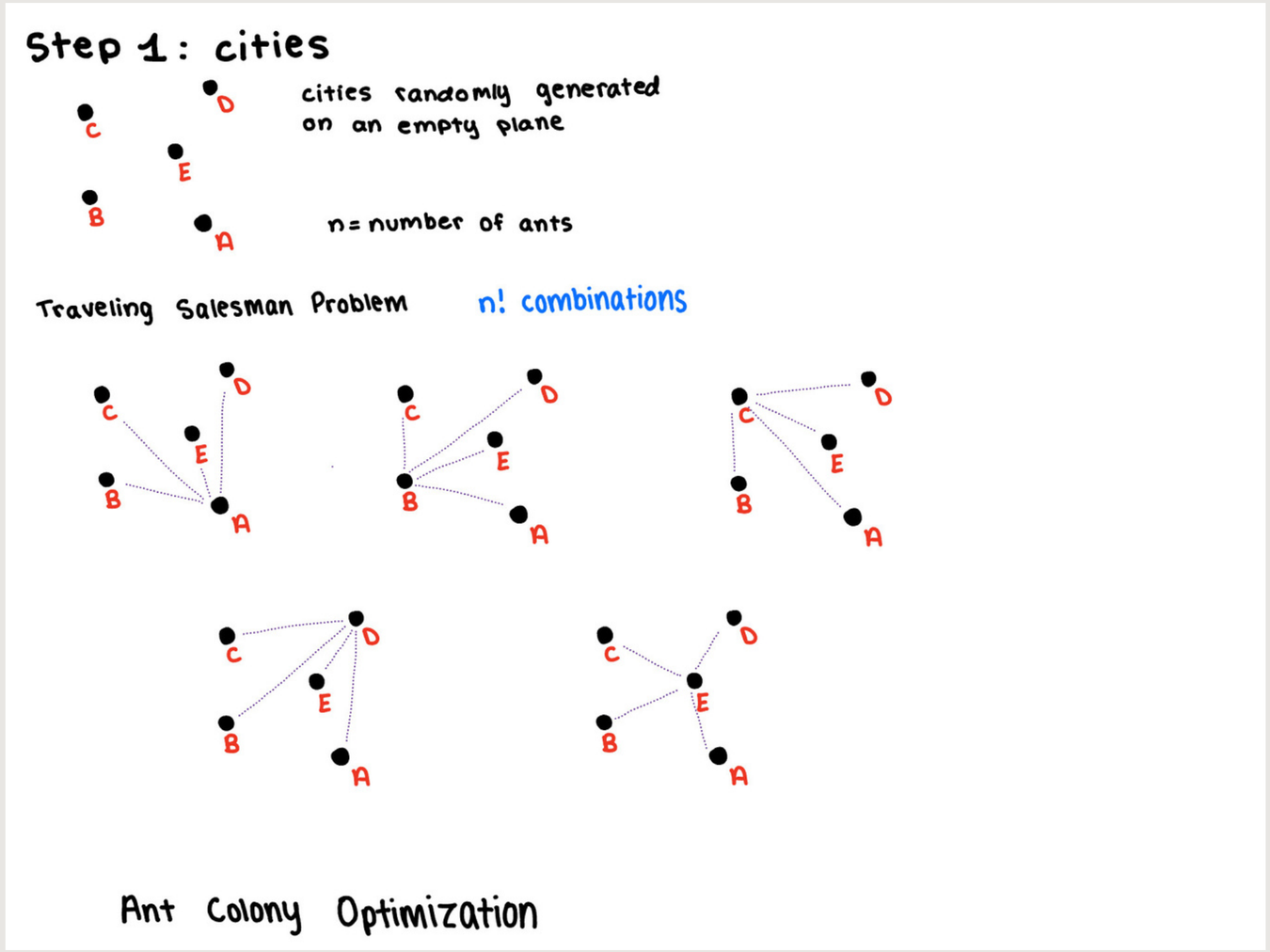
In Computation, a "Tick" refers to a single iteration of the main game loop. It is used as a standard measure of time in algorithms since it is independent of factors such as CPU Power or available memory.

In the ACO, one tick refers to every Ant making a single city-to-city movement. Thus a TSP problem with N cities will require a minimum of N ticks to solve.

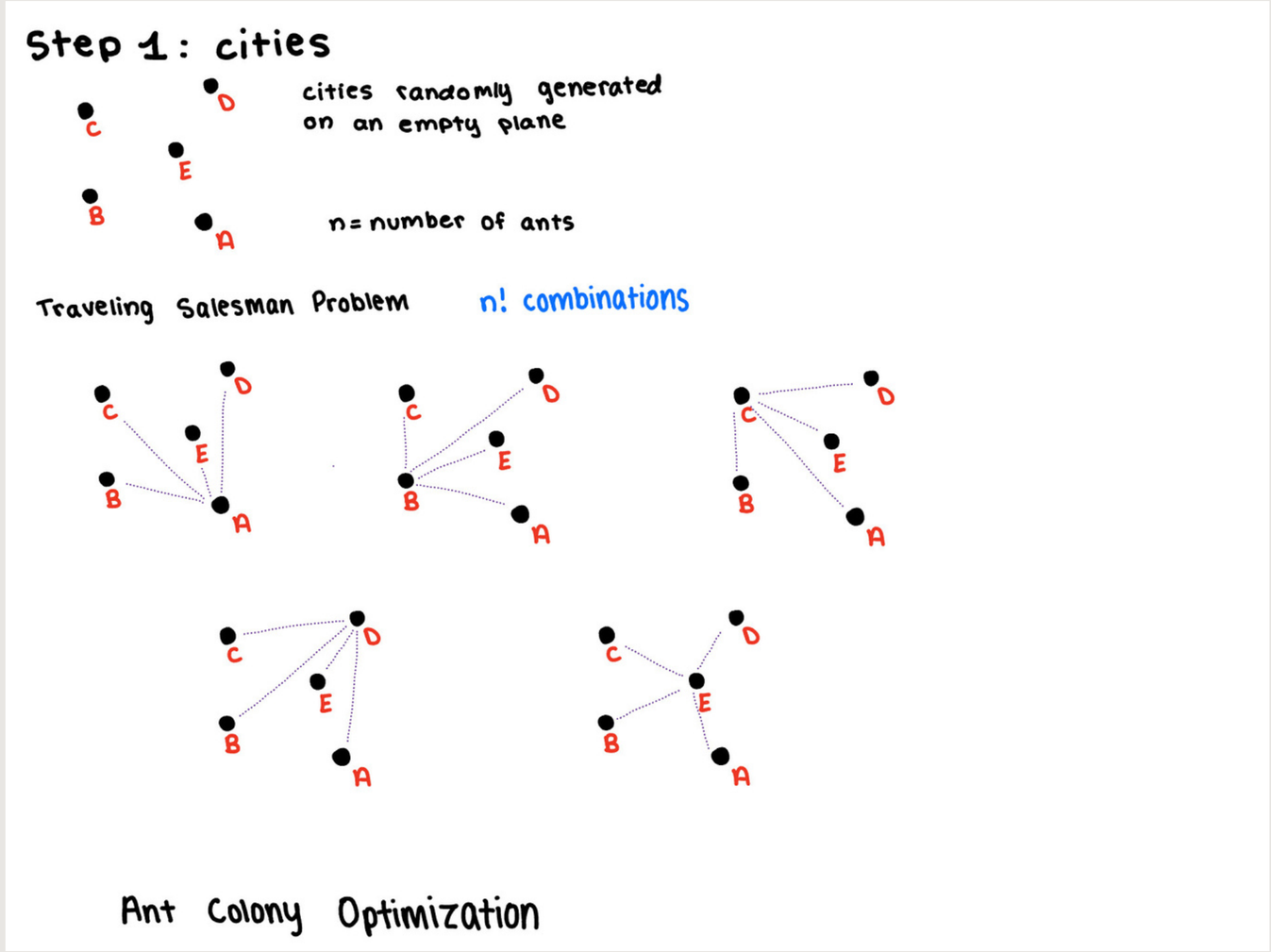
William

Math Behind The ACO Algorithm

Math Behind The ACO Algorithm



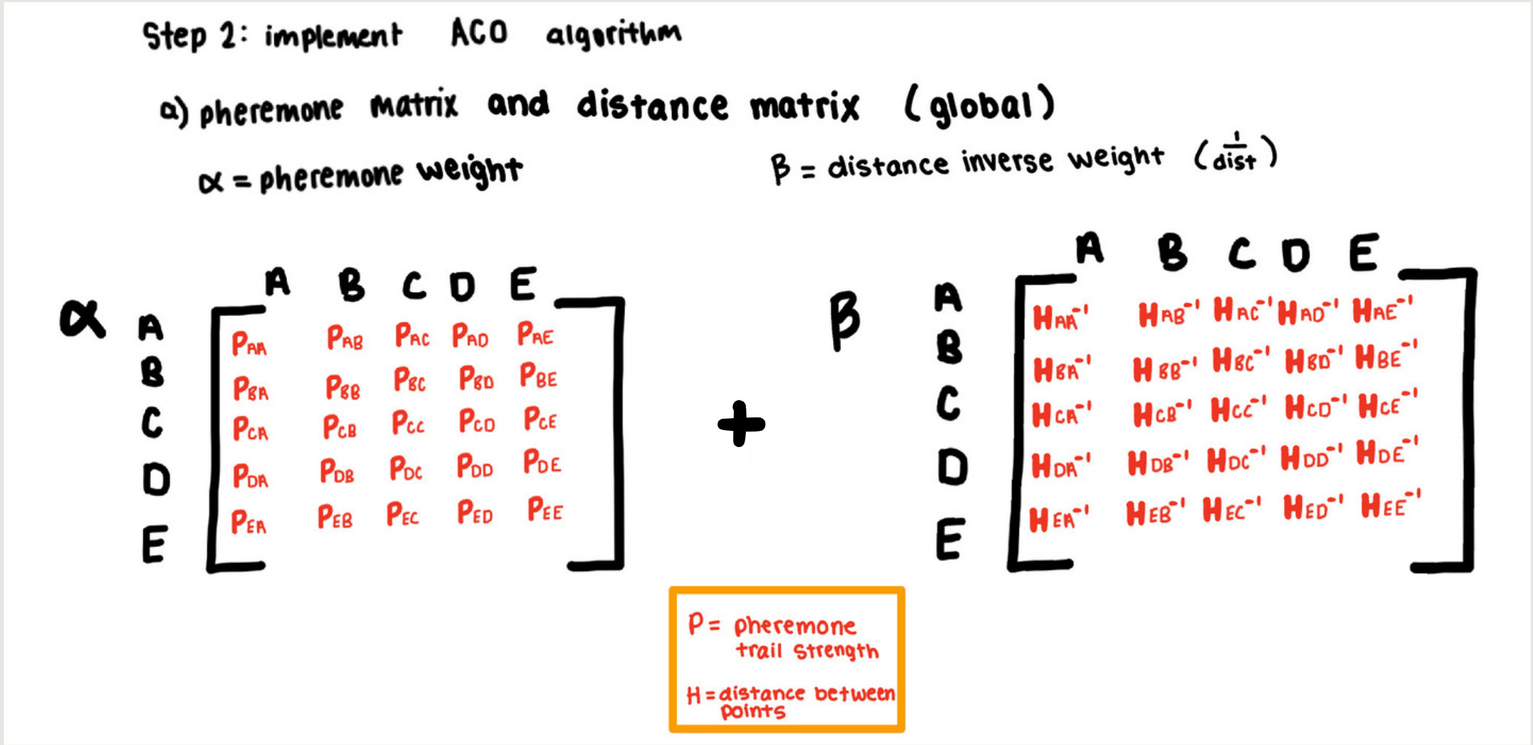
Math Behind The ACO Algorithm



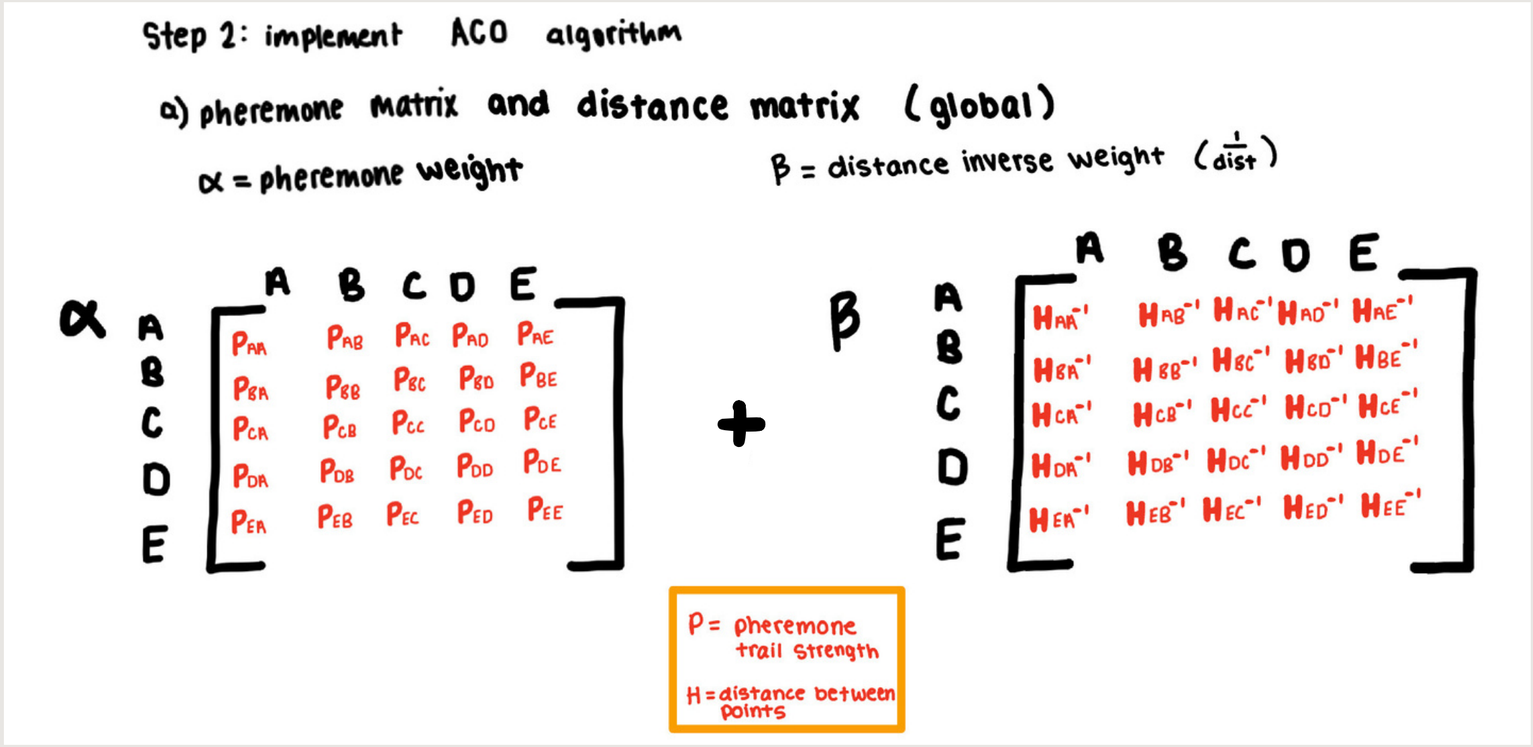
The Ant Colony Optimization (ACO) Algorithm starts with 'n' ants distributed randomly among the cities

Aaryan

Math Behind The ACO Algorithm



Math Behind The ACO Algorithm



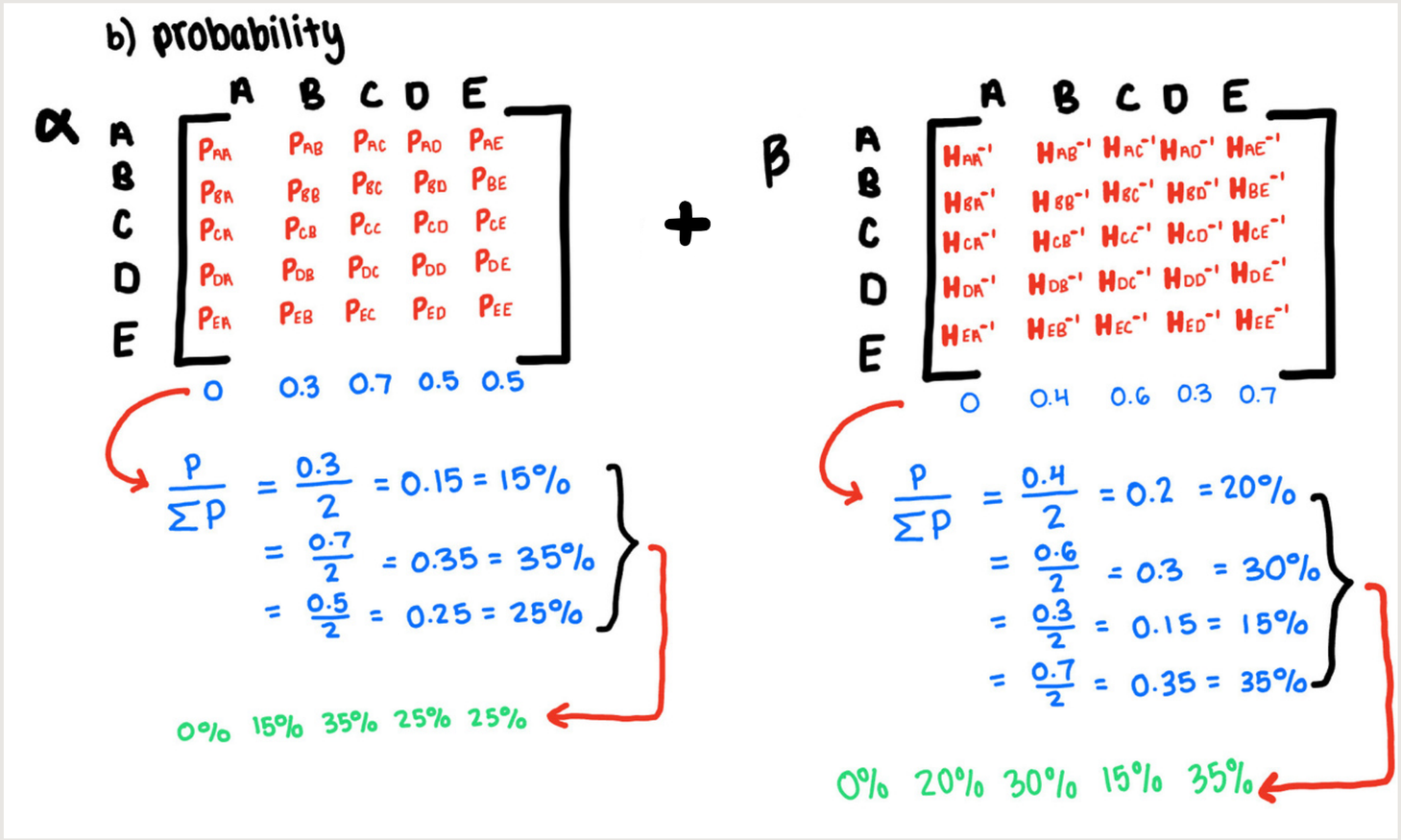
The ACO uses two matrices to store certain values - Pheromone Strength and the Inverse Distance

Pheromone Strength is a measure of how often a path is travelled and is used to reward travelling on the best path.

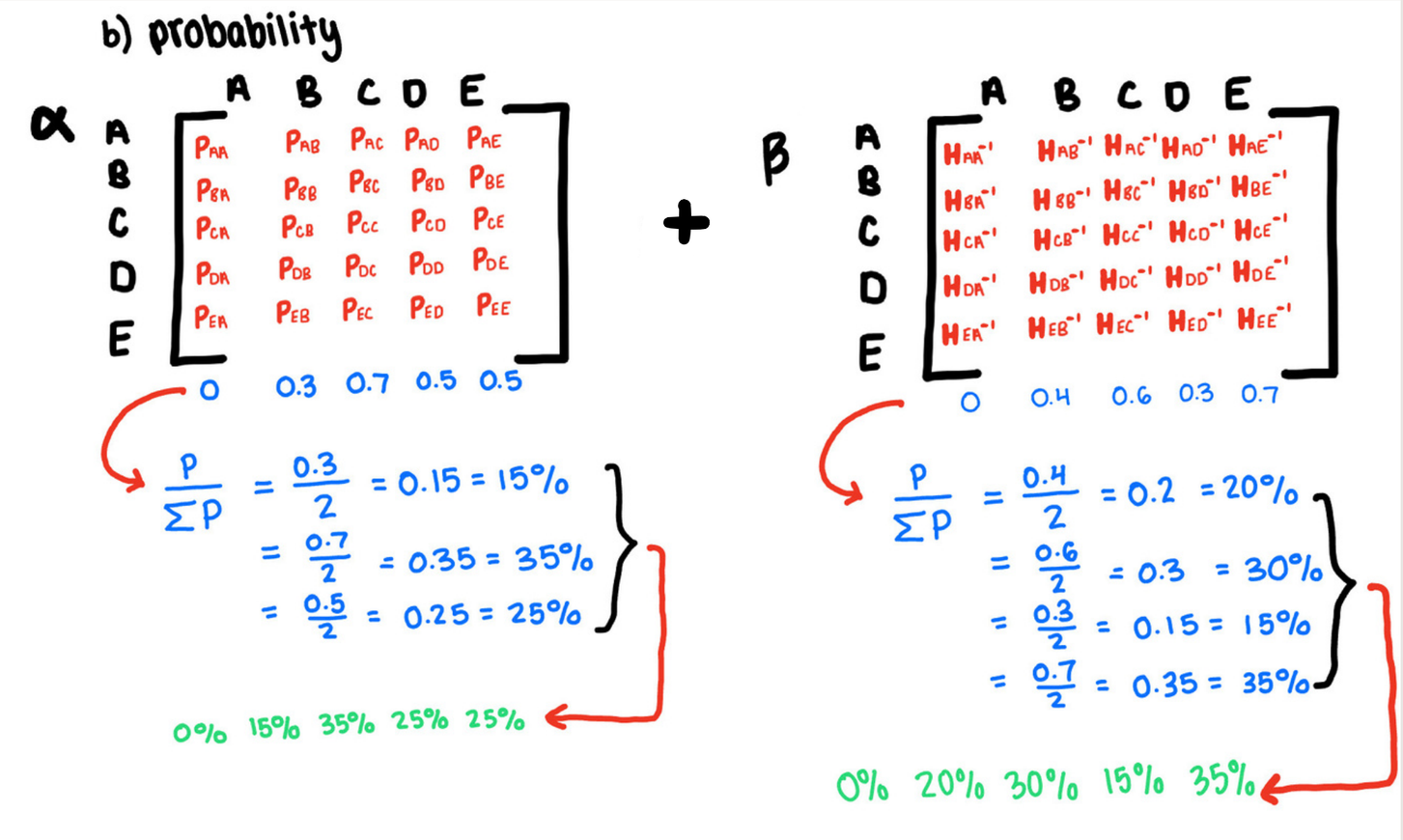
Inverse Distance, as the name suggests, is a value that decreases with increase in distance. This is used to prioritize moving to closer cities.

These matrices are then multiplied by constants Alpha and Beta.

Math Behind The ACO Algorithm



Math Behind The ACO Algorithm

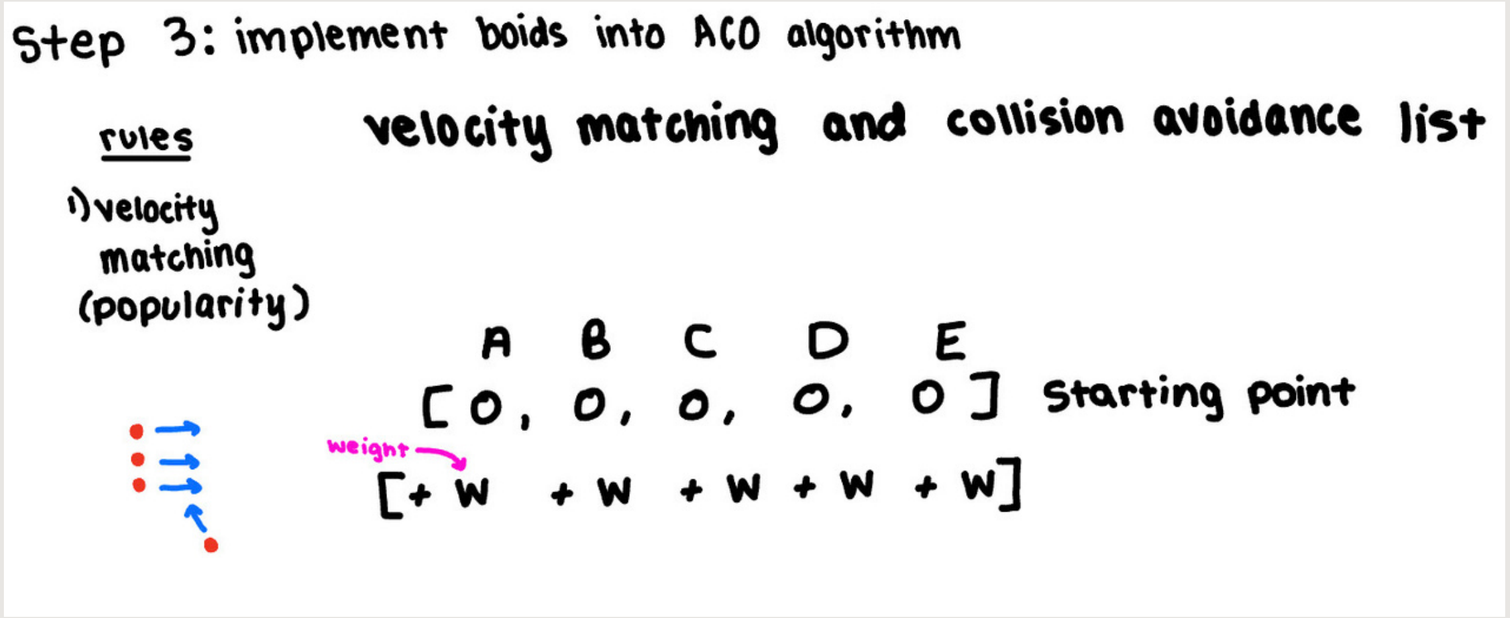


When a "Tick" is called, each Ant must decide which city to go to next. To do this, the ant chooses the rows from the matrices which correspond to its current city.

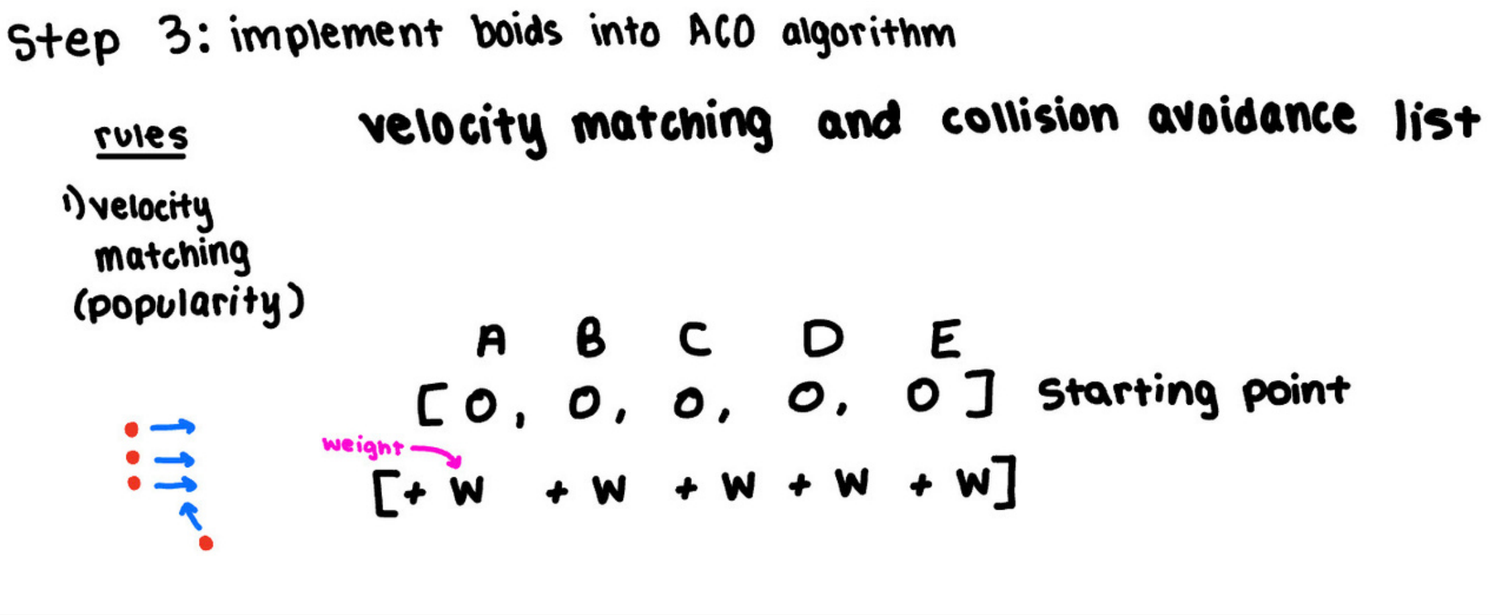
The values from the matrices are said to be "raw" and must be normalized into probabilities in the range [0,1]. To do this, each value is divided by the sum of the values.

Our Modifications to the ACO Algorithm

Our Modifications to the ACO Algorithm



Our Modifications to the ACO Algorithm

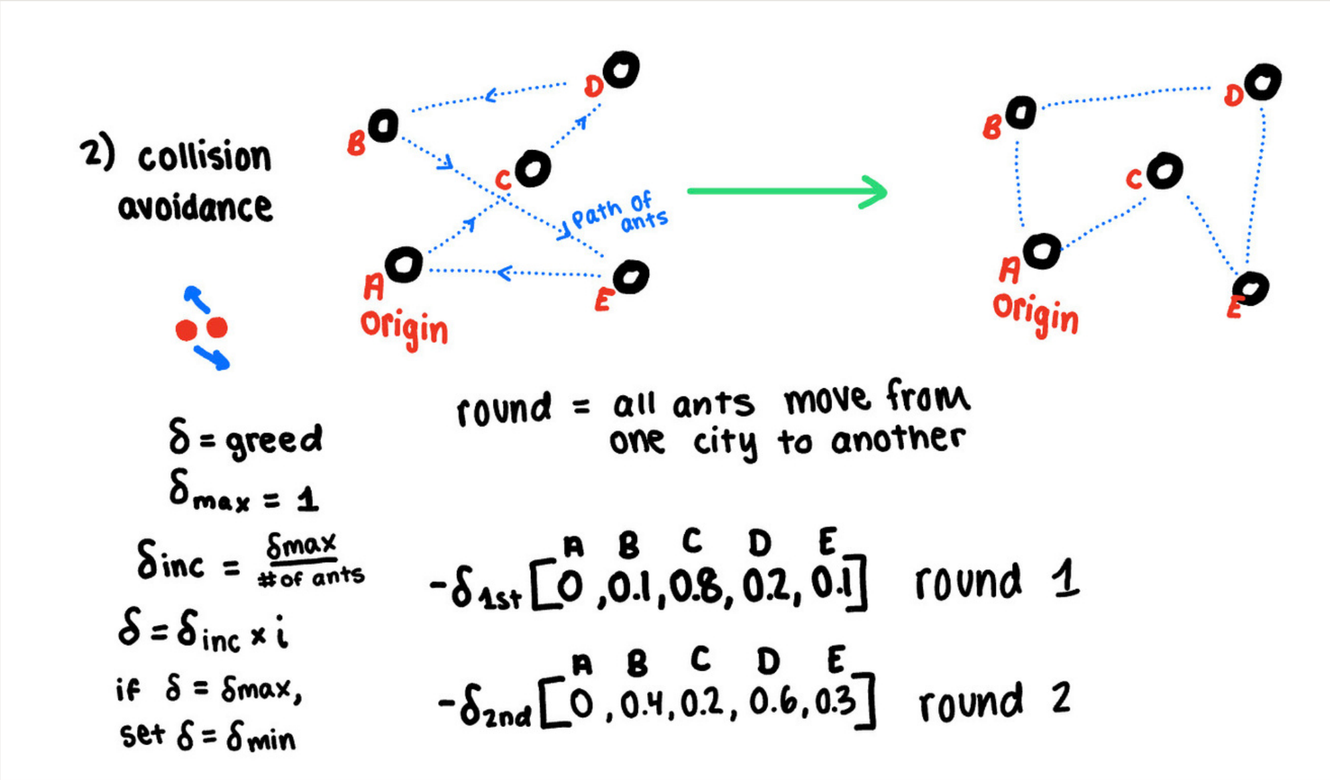


The first rule of the Boids paper is Velocity Matching. The direct analogue to this is called "Popularity", which is a measure of how often a city is visited (not to be confused with the Pheromone matrix, which is for *paths* rather that *cities*).

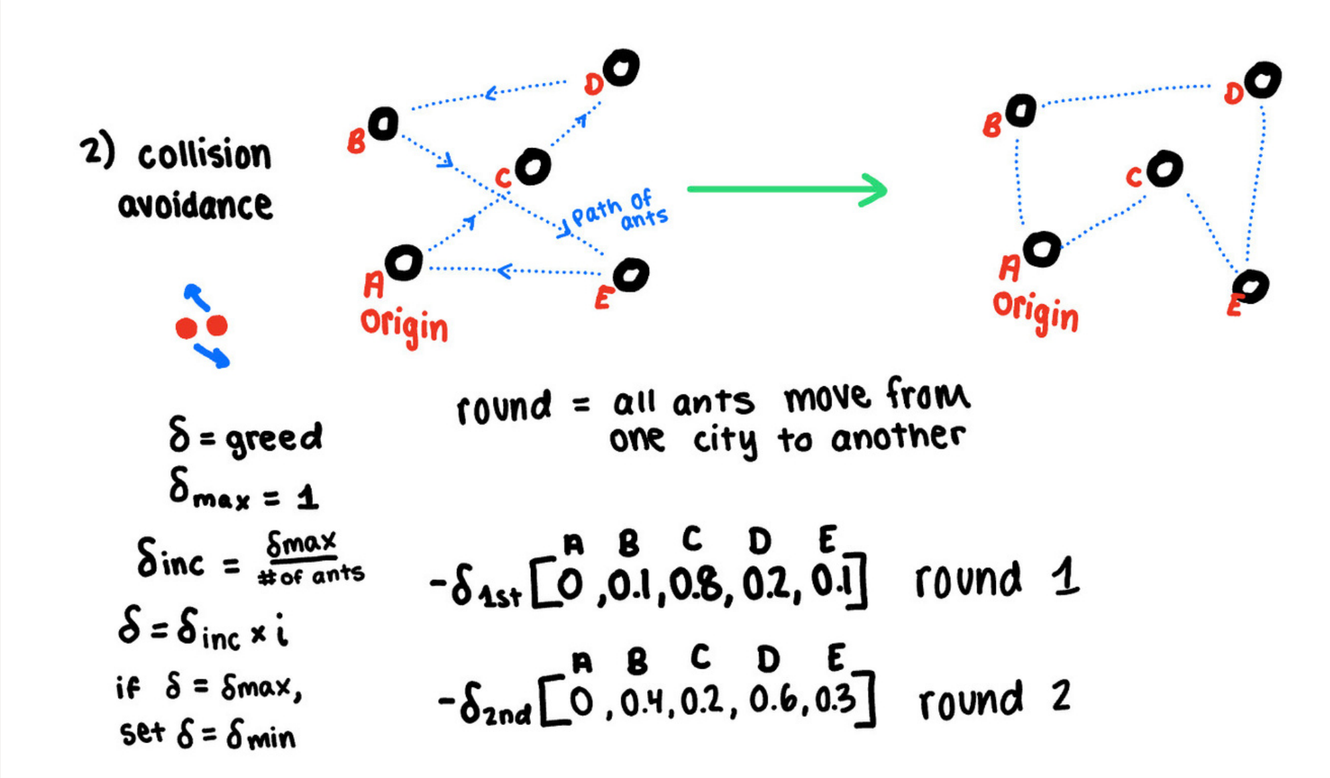
Every time an Ant chooses to go to a city, it does so by assigning a certain "weight" to the city.

To implement Popularity, the chosen weight is added to the city's Popularity value. The Popularity list is multiplied by a constant Gamma, which is then used by other Ants to make decisions.

Our Modifications to the ACO Algorithm



Our Modifications to the ACO Algorithm

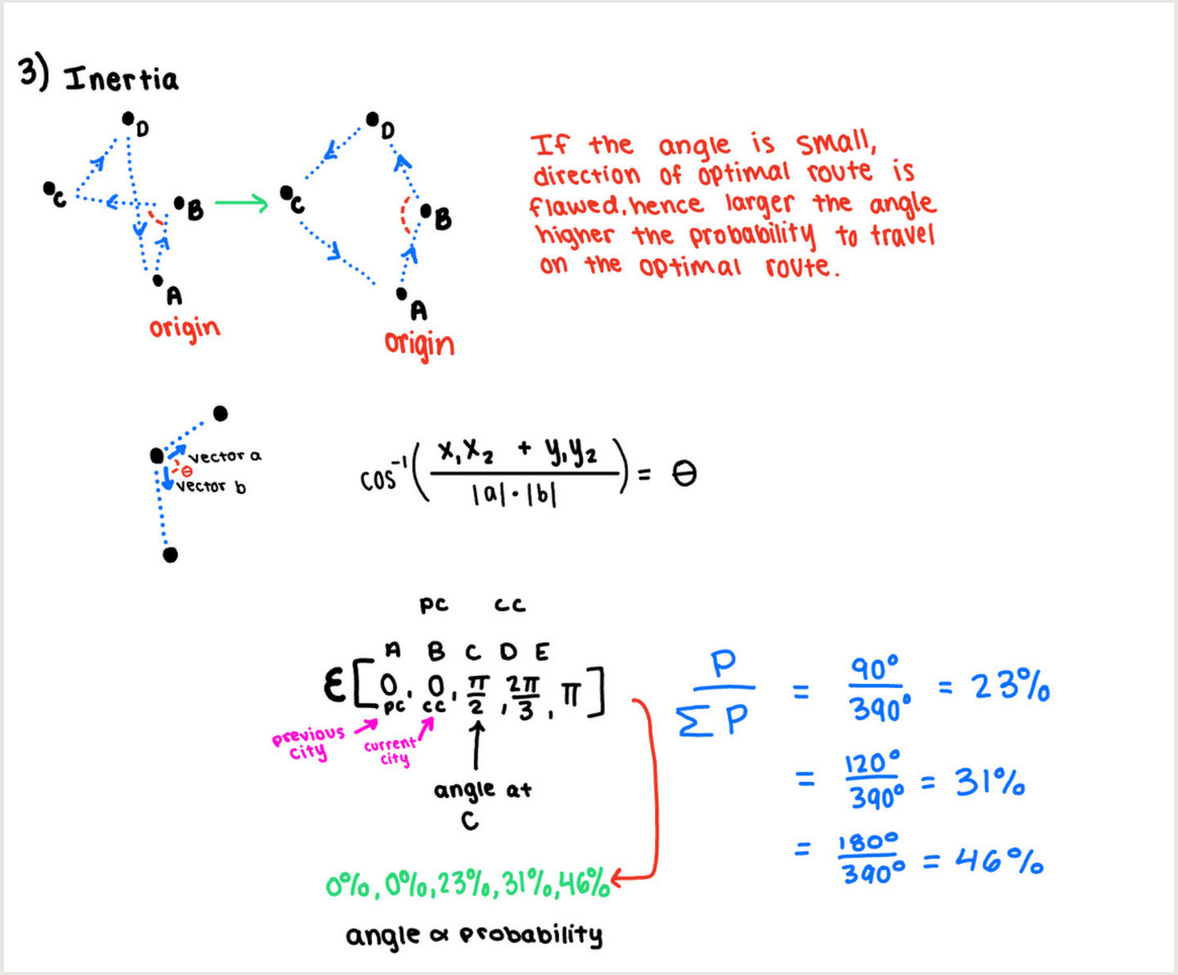


The second rule of the Boids paper is Collision Avoidance, and this is made analogous to "Greed", which is a measure of how many ants have travelled to a certain city *on that turn*. This is inspired by Ants' behaviour of finding better food sources.

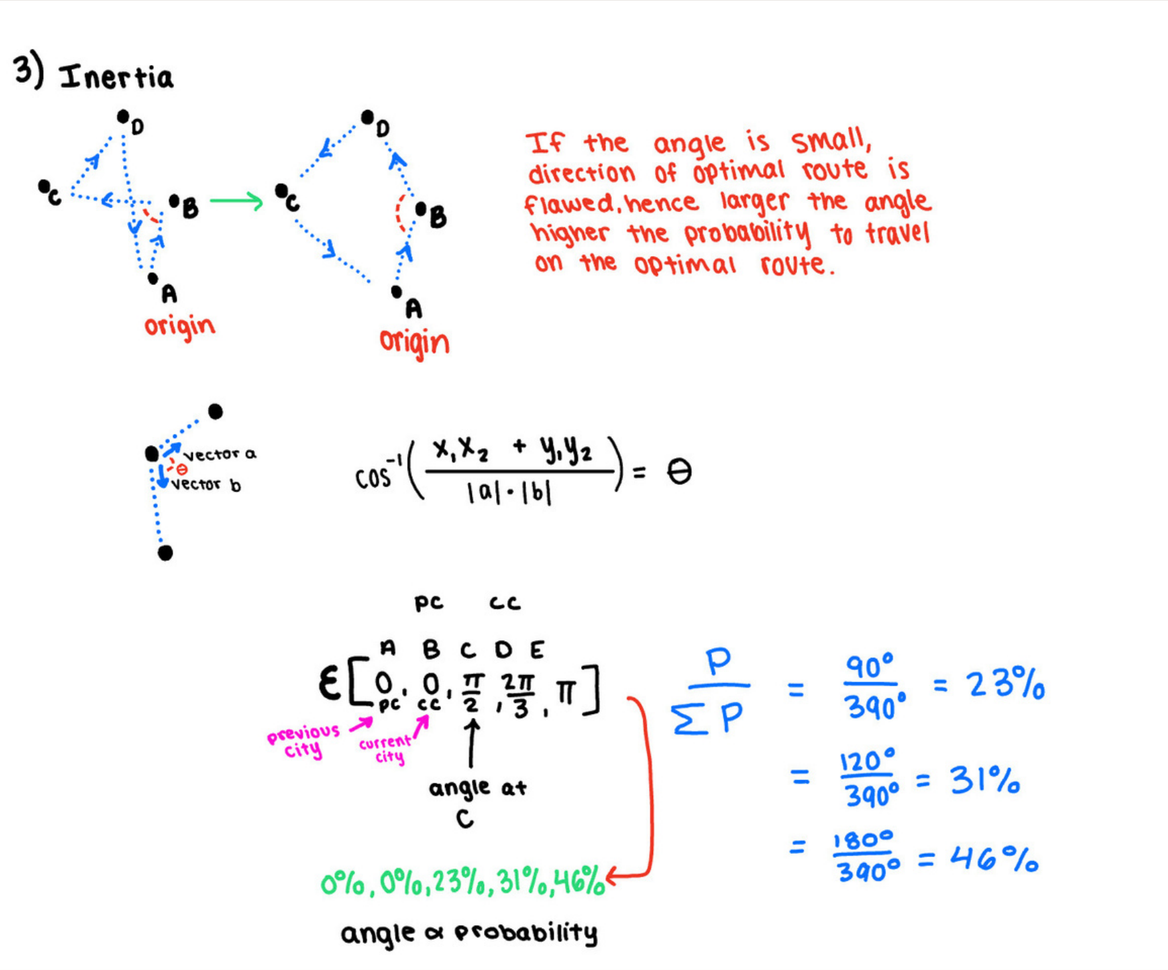
When an Ant travels to a city on a certain round, it increments the counter for that city at that specific round. This round-wise list is then multiplied by a *variable* number Delta.

Delta is special in two ways: One, it is negative, hence it pushes Ants *away* from commonly visited cities. Second, it changes for each Ant. The first Ant has a Delta = 0 (no greed) while the last Ant has a Delta = Delta_max (most greed).

Our Modifications to the ACO Algorithm



Our Modifications to the ACO Algorithm



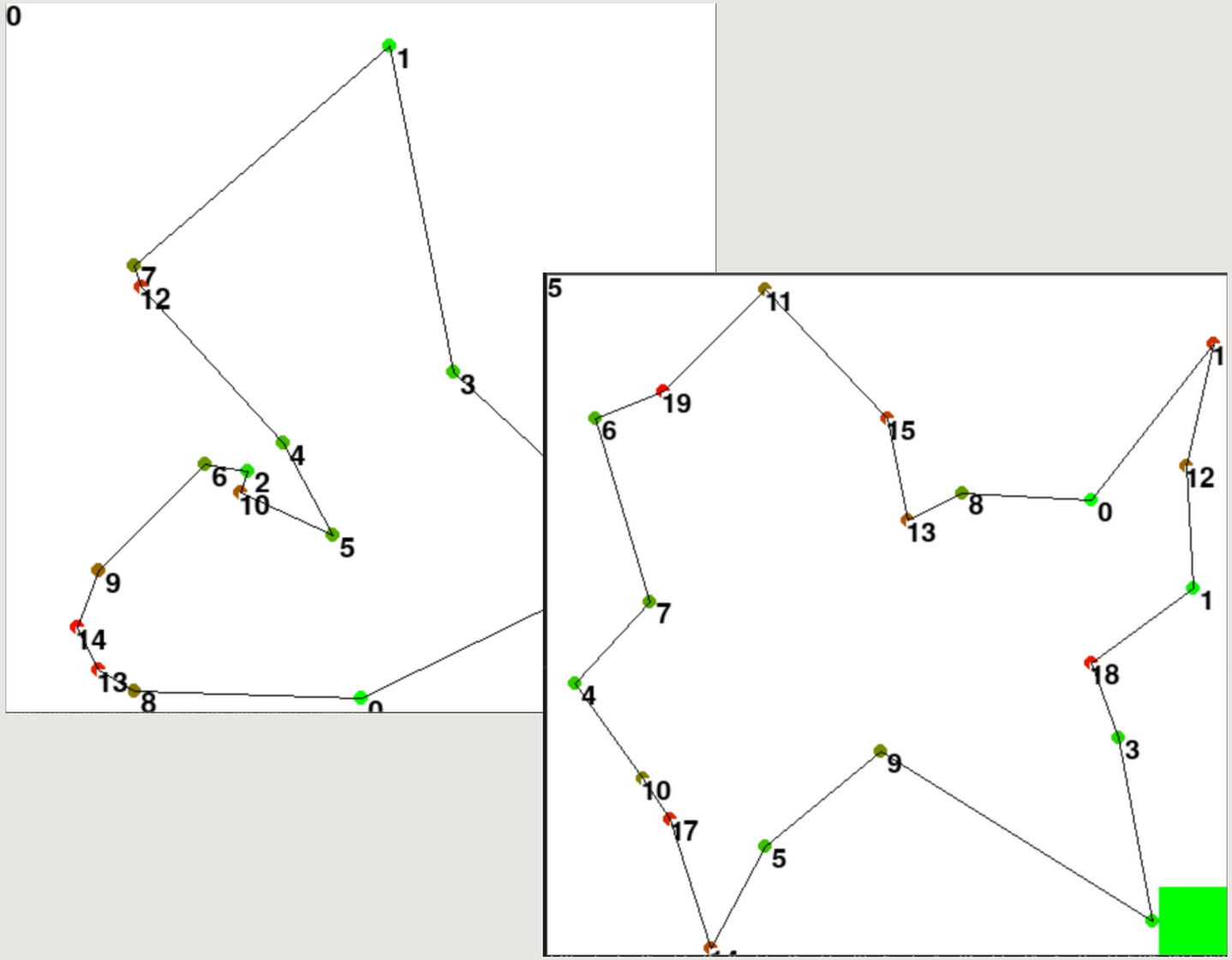
The third rule of the Boids paper is Flock Centering. The analogue for this rule is less direct, since the ACO does not function using Ant groups. The way this rule is implemented is using "Inertia". Inertia is a way to prevent ants from making sharp turns.

When the algorithm is initialized, the angles between cities are stored. During Ant decision making, the angles are normalized to [0,1] and multiplied by a constant Epsilon.

This quality causes Ants to prefer cities that are in the same direction as the Ant is travelling, hence preventing sharp turns and backtracking.

Running Simulations

Running Simulations - Research Method



The increase in efficiency of various programs is evaluated by running each program on a predefined dataset of points.

Consistent data is obtained by running the algorithm on a few selected datasets with N points for k=5 iterations.

The photo (left) shows a completed path on dataset s=0 with n=15 points.

The photo (right) shows a completed path on dataset s=5 with n=20 points.

Running Simulations - The Code

tester.py

```python
from versions.v4 import City, Ant, Colony
import threading
import random
import sys

NCITIES = 15
NANTS = 100
SEED = 0
DECAY = 0.5/NCITIES

MINSEED = int(sys.argv[-3])
MAXSEED = int(sys.argv[-2])
ITERS = int(sys.argv[-1])

pherWeight = float(sys.argv[-8])
idistWeight = float(sys.argv[-7])
poplWeight = float(sys.argv[-6])
greedWeight = float(sys.argv[-5])
inertiaWeight = float(sys.argv[-4])

lock = threading.Lock()
data = []
```

This sets a few constants which are kept the same across tests to increase reliability

Running Simulations - The Code

tester.py

```python
from versions.v4 import City, Ant, Colony
import threading
import random
import sys

NCITIES = 15
NANTS = 100
SEED = 0
DECAY = 0.5/NCITIES

MINSEED = int(sys.argv[-3])
MAXSEED = int(sys.argv[-2])
ITERS = int(sys.argv[-1])

pherWeight = float(sys.argv[-8])
idistWeight = float(sys.argv[-7])
poplWeight = float(sys.argv[-6])
greedWeight = float(sys.argv[-5])
inertiaWeight = float(sys.argv[-4])

lock = threading.Lock()
data = []
```

This sets a few constants which are kept the same across tests to increase reliability

These constants define the minimum (start) and maximum (end) seed as well as the number of iterations per seed. These values are obtained from the command line

Running Simulations - The Code

tester.py

```python
from versions.v4 import City, Ant, Colony
import threading
import random
import sys

NCITIES = 15
NANTS = 100
SEED = 0
DECAY = 0.5/NCITIES

MINSEED = int(sys.argv[-3])
MAXSEED = int(sys.argv[-2])
ITERS = int(sys.argv[-1])

pherWeight = float(sys.argv[-8])
idistWeight = float(sys.argv[-7])
poplWeight = float(sys.argv[-6])
greedWeight = float(sys.argv[-5])
inertiaWeight = float(sys.argv[-4])

lock = threading.Lock()
data = []
```

This sets a few constants which are kept the same across tests to increase reliability

These constants define the minimum (start) and maximum (end) seed as well as the number of iterations per seed. These values are obtained from the command line

These weights are used to control the parameters of the ACO. The values are obtained from the command line. The values of *pherWeight* and *idistWeight* are kept at 0.4 and 0.5 for all tests. The other weights can be modified depending on the test.

Running Simulations - The Code

tester.py

```
def onThread(seed, id):
    rand = random.Random(seed)
    cities = [City(5*rand.randint(1, 99), 5*rand.randint(1, 99), 1) for _ in range(NCITIES)]
    ants = [Ant(random.Random(), cities) for _ in range(NANTS)]
    colony = Colony(random.Random(), cities, ants, DECAY, 5000,
                    pherWeight, idistWeight, poplWeight, greedWeight, inertiaWeight)
    for ant in ants: ant.setColony(colony)
    pathComplete = False
    while not pathComplete:
        pathComplete = colony.tick()
    with lock:
        data.append((seed, colony.bestTick, colony.bestPLen, colony.bestPath))
        print(f"Chunk {seed},{id} complete")

threads = []
for seed in range(MINSEED, MAXSEED+1):
    for i in range(ITERS):
        thread = threading.Thread(target=onThread, args=(seed,i))
        thread.setDaemon(True)
        threads.append(thread)
        thread.start()

for thread in threads:
    thread.join()

with open(f"data {sys.argv[-9]}.txt", mode='w') as fl:
    for item in data:
        fl.write(f"[{str(item)}],")

```

This function generates the cities, ants, and the colony. It then runs the algorithm until the path is complete.

Once the path is complete, it logs the length of the path and the amount of ticks required to achieve it

Running Simulations - The Code

tester.py

tester.py > onThread

```python
def onThread(seed, id):
    rand = random.Random(seed)
    cities = [City(5*rand.randint(1, 99), 5*rand.randint(1, 99), 1) for _ in range(NCITIES)]
    ants = [Ant(random.Random(), cities) for _ in range(NANTS)]
    colony = Colony(random.Random(), cities, ants, DECAY, 5000,
                    pherWeight, idistWeight, poplWeight, greedWeight, inertiaWeight)
    for ant in ants: ant.setColony(colony)
    pathComplete = False
    while not pathComplete:
        pathComplete = colony.tick()
    with lock:
        data.append((seed, colony.bestTick, colony.bestPLen, colony.bestPath))
        print(f"Chunk {seed},{id} complete")

threads = []
for seed in range(MINSEED, MAXSEED+1):
    for i in range(ITERS):
        thread = threading.Thread(target=onThread, args=(seed,i))
        thread.setDaemon(True)
        threads.append(thread)
        thread.start()

for thread in threads:
    thread.join()

with open(f"data {sys.argv[-9]}.txt", mode='w') as fl:
    for item in data:
        fl.write(f"[{str(item)}],")

```

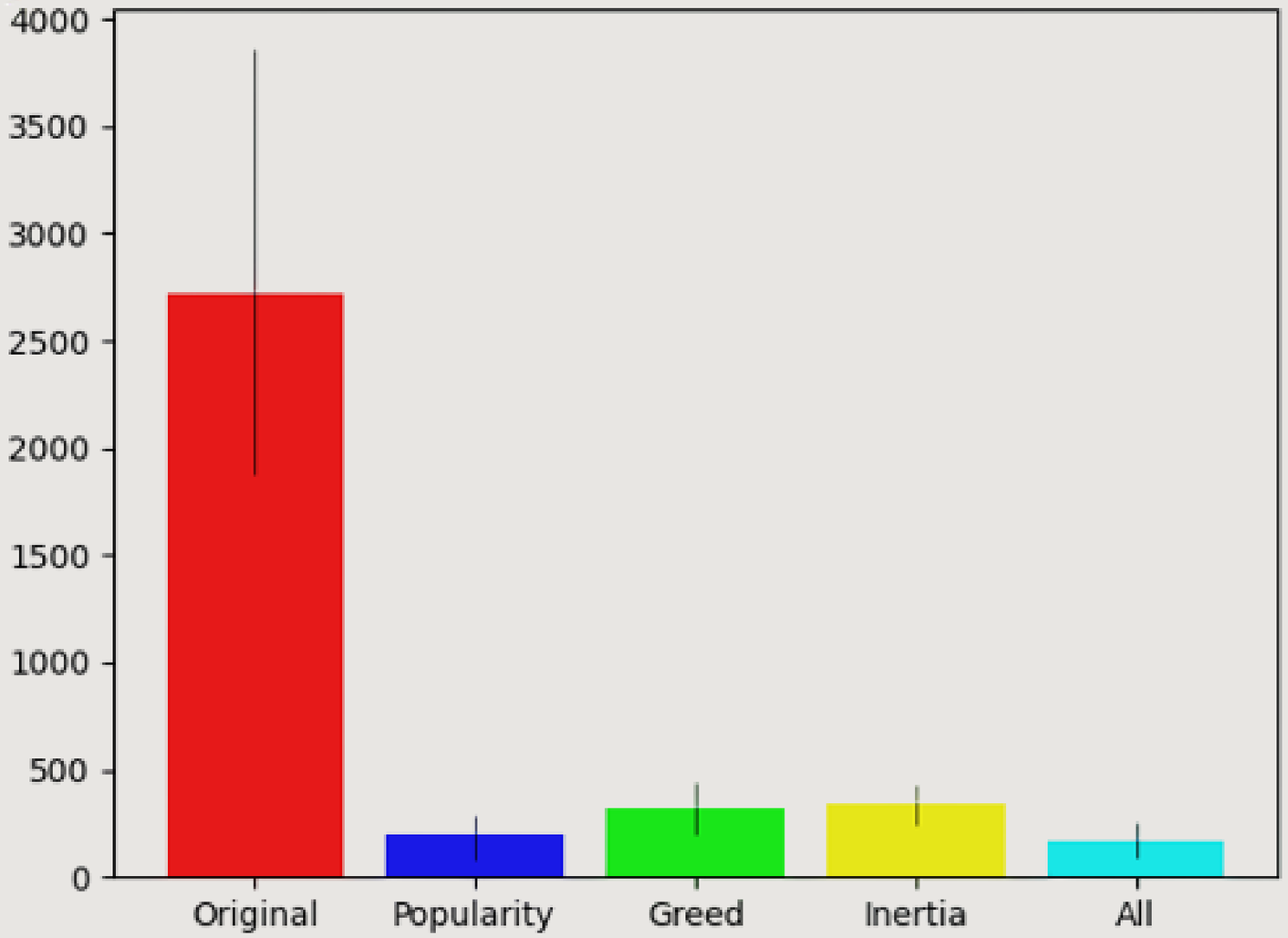
This function generates the cities, ants, and the colony. It then runs the algorithm until the path is complete.

Once the path is complete, it logs the length of the path and the amount of ticks required to achieve it

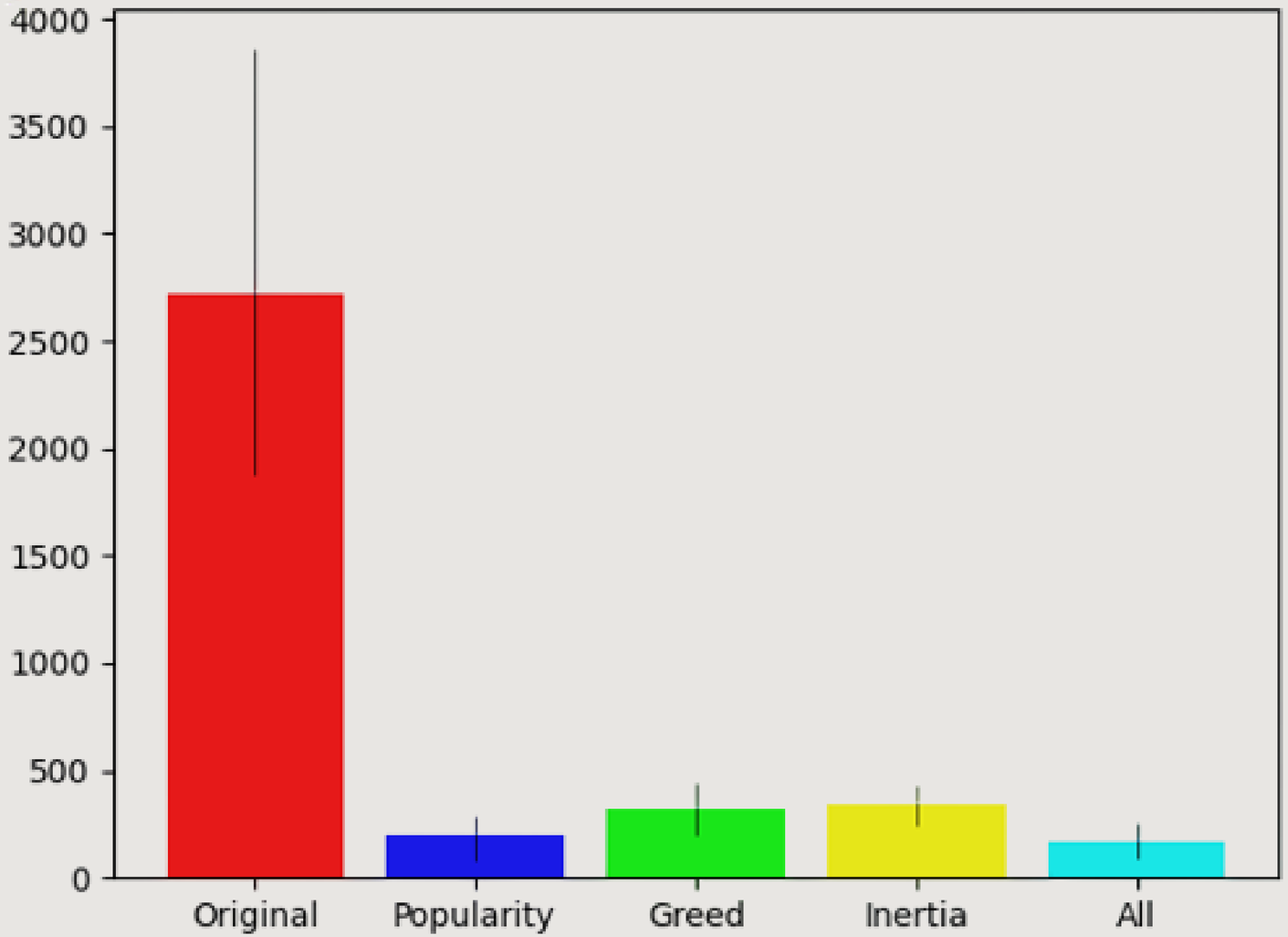
The loop spawns multiple instances of the solver function across multiple seeds and iterations.

It then waits for all the threads to complete before writing the collected data to a file.

Running Simulations - Data Comparison



Running Simulations - Data Comparison



	Min	Avg	Max
Original	1878	2718	3858
Popularity	84	195	291
Greed	204	316	432
Inertia	246	346	432
Combined	93	162	255

Bibliography - References

1. Ant Colony Optimization Algorithms." InTechOpen, 2012.
2. Dorigo, Marco, and Thomas Stützle. "Ant Colony Optimization." MIT Press, 2004.
3. Gendreau, Michel, and Jean-Yves Potvin. "Handbook of Metaheuristics." Springer, 2010.
4. Russell, Stuart J., and Peter Norvig. "Artificial Intelligence: A Modern Approach." Pearson, 2016.
5. Strogatz, Steven H. "Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering." Westview Press, 2014.
6. Gleick, James. "Chaos: Making a New Science." Penguin Books, 2008.
7. Li, Tingwen, and Dingguo Zhang. "Chaos-Based Optimization Methods." Springer, 2016.
8. Craig Reynolds: Flocks, Herds, and Schools: A Distributed Behavioral Model, www.cs.toronto.edu/~dt/siggraph97-course/cwr87/. Accessed 28 June 2023.
9. “Solving the Travelling Salesman Problem Using Ant Colony Optimization.” YouTube, 18 May 2021, www.youtube.com/watch?v=oXb2nC-e_EA&ab_channel=LearnbyExample.
10. Ant Colony Optimization for the Traveling Salesman Problem, 6 Sept. 2022, strikingloo.github.io/ant-colony-optimization-tsp.