# Ant Colony Optimization Project.

Course – Intro to optimizations by Dr' Ariel Rosenfeld

Noam Koren 308192871
Yuval Khanimov 318970902

## What is Ant colony Optimization?

We'll start by explaining how ants work collaboratively to find food and other resources efficiently in nature.

Leaving the nest, each ant wanders randomly until it finds food.
While going back to the nest the ant will leave a trail of pheromones behind that will evaporate over time.
Then, when other ants leave the nest - instead of wandering randomly they will prefer to follow a path with a high concentration of pheromones with a somewhat high probability, and upon finding food the ants will reinforce the path with more pheromones. Thus, since the pheromones evaporate over time, a shorter path will have higher concertation of pheromones, eventually leading more ants to travel this path instead of a longer one.

The idea of Ant Colony Optimization or for short - ACO, is to mimic this behavior, with artificial ants or "agents" walking on the edges of a graph ( or any other representation of a problem ), while each edge has a cost aka "length".
We can convert this problem to many known problems in the real world like the Traveling salesman problem which we will cover later.
The Algorithm is quite simple and goes as follows:

```
Procedure ACO:
    while not nerminated:
        generateSolutions()
        deamonActions()
        updatePheromones()
     repeat
end
```

Each ant will select an edge with a probability defined as:

$$p_{xy}^{k} = \frac{\left(\tau_{xy}^{\alpha}\right)\left(\eta_{xy}^{\beta}\right)}{\sum_{z \in allowed_z}\left(\tau_{xy}^{\alpha}\right)\left(\eta_{xy}^{\beta}\right)}$$

Where $\tau_{xy}$ is the amount of pheromones that will be released going from $x$ to $y$.

And $\eta_{xy}$ being the desirability to go from x to y.

The parameters $\alpha$, $\beta$ corresponds to the influence of $\tau$ and the desirability of $\eta$ accordingly.

After each iteration (every ant found a solution, (or we timed out) we will update the pheromones (some ACO variations use different update ruls)

For the general ACO the pheromone update is as follows:

$$\tau_{xy} = (1-\rho)\tau_{xy} + \sum_{k}^{m} \Delta\tau_{xy}^{k}$$

With $m$ being the number of ants and $\rho$ the evaporation rate.

In this project, we will compare ACO with some of the optimizations that we learned during the course on the Traveling salesman problem.

The traveling salesman problem is as follows:

Given a set of cities and distance or path cost between each pair of cities, what is the shortest or cheapest path that visits each city once and eventually returns to the origin city?

The traveling salesman problem is a well-known problem in mathematics and computer science and classified as a NP-Hard problem meanings there is no deterministic algorithm that solves the problem in polynomial time.

To fit ACO to this problem we will define $\Delta\tau_{xy}^{k}$ as:

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k, & if\ ant\ k\ uses\ edge\ (x,y)\ on\ its\ solution \\ 0, & otherwise \end{cases}$$

$Q$ is a constant and $L_k$ is the score for the $k$th ant for its solution.

We will try to optimize a solution with ACO and we will be comparing it to Simulated annealing which we learned during the course. first, we will represent 3 variants of ACO which we will be testing to see which performs best against Simulated annealing and in general.

1. **ACS – Ant Colony System:**
   this variant is different from the original ACS (Ant colony system) that is described above in three ways:

1. Ants selecting edges are biased for exploitation, meaning each ant will be favoring shorter paths with a large amount of pheromones.

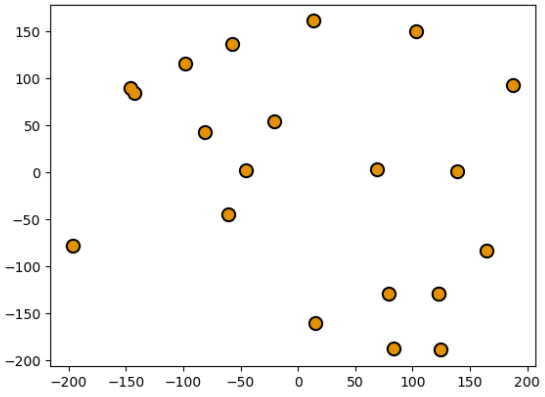
2. Ants update the pheromones locally each iteration

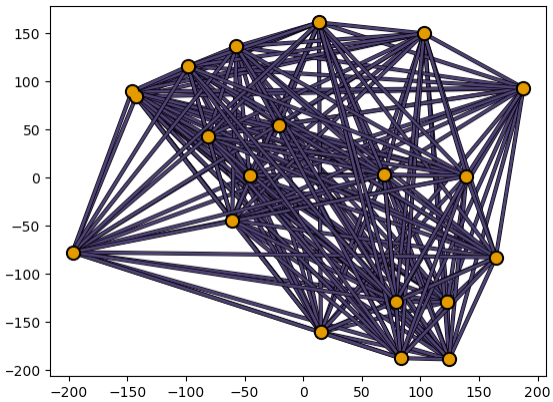
3. The best ant (the ant who found the best path so far) will allow reproducing pheromones so older solutions that are worst will evaporate over time.

2. **Elitist ant system:**
   Here the best solutions will be maintained each iteration (add more pheromones) even if no ant walked on this path. With this ants will tend to use the part of the this path while finding outer solutions.

3. **Minmax ant system:**
   each trail is limited to a minimum and maximum amount of pheromones; each trail starts with the maximum amount of pheromones to allow high exploration. And only the best route is allowed to add more pheromones each iteration, so one might imagine this as we light up all the possible solutions and every solution fades away except the best one or one which is very good.

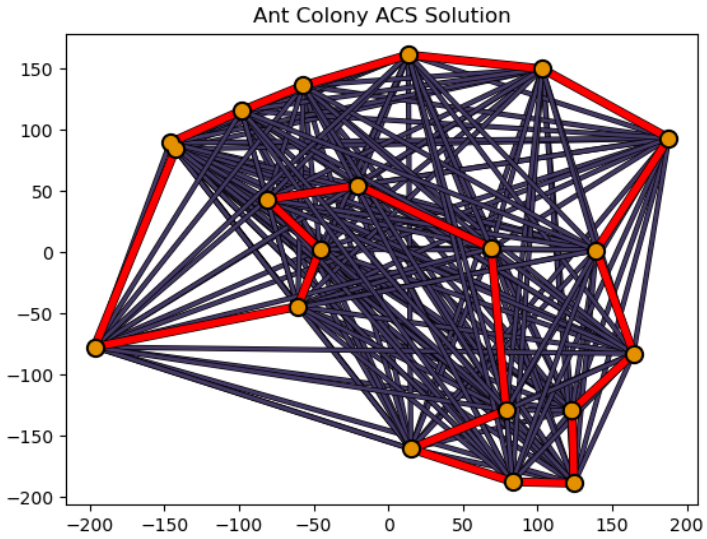
Well start by looking at the following problem:
Given a graph with nodes as random coordinates. Solving the traveling salesmen problem, we will want to find the shortest path (Euclidian distance between two points) that will visit all the nodes (20):



On the right is our coordinates system and on the left is all possible paths (complete graph).
We choose a set of vertices where the coordinates go from -200 to 200 and were generated randomly. For our comparations with Simulated annealing, we will be using the same set for all corporations to avoid biased results towards some graphs.

Well start by simply running the ACS algorithm with 30 ants and 300 iterations to see how it performs and we will get the following solution:



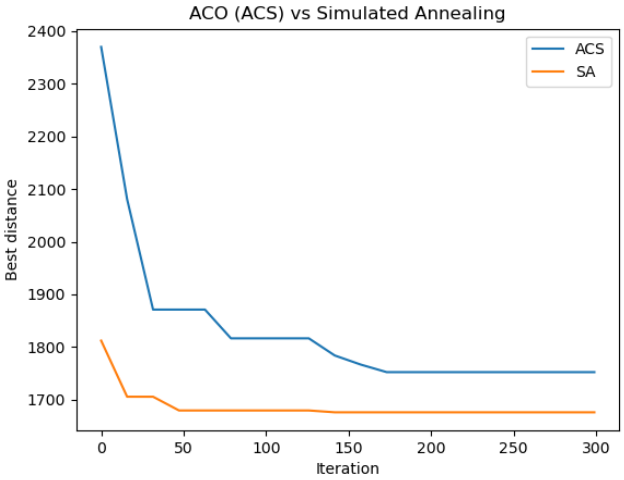

Looking pretty cool! Next, we will compare the Simulated Annealing algorithm which we learned during the course with all 3 Ant system variants, and we want to focus on the following questions:

- Since both algorithms are anytime algorithms, which algorithm is faster to come close to the optimal solution or find it?
- How much adding ants will improve against Simulated annealing?

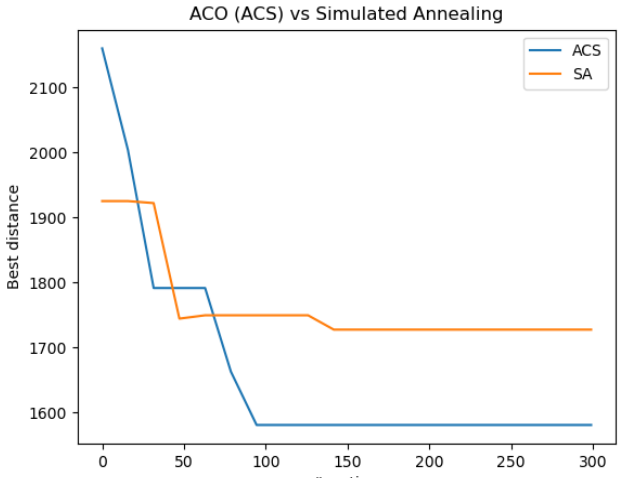
    - We will answer these for each ACO variant described above.

We will try various combinations of ants and iterations and Compare the result for each algorithm running both algorithms and plot their best solution for each iteration to see which is faster to come close to an optimal solution, well start with 1 & 10 ants and 300 iterations.

*Ant count: 1*




*Ant count: 10*



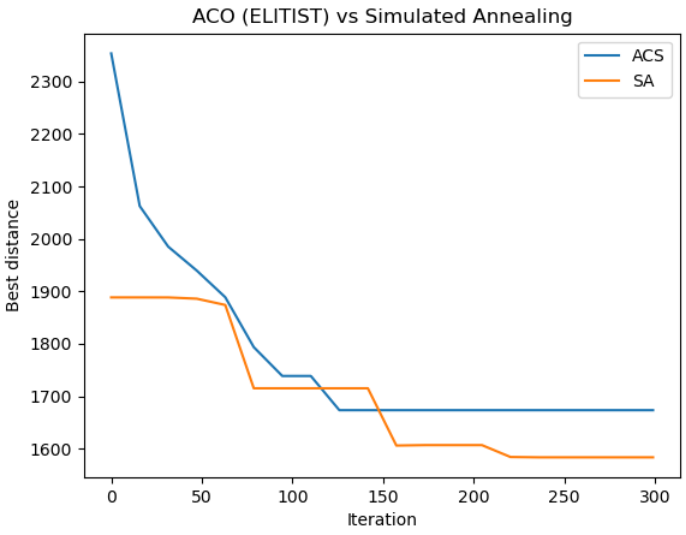

The most noticeable difference is that ACS will find a good solution fast and start with a bad solution since its starts with a random solution which statistically will be bad on a large graph, while SA will start with not such a bad solution and slowly will improve it over time.

As seen in the left chart (with 1 ant) SA found a better solution, but running the test again we might get a much different result with ACS giving the better solution.

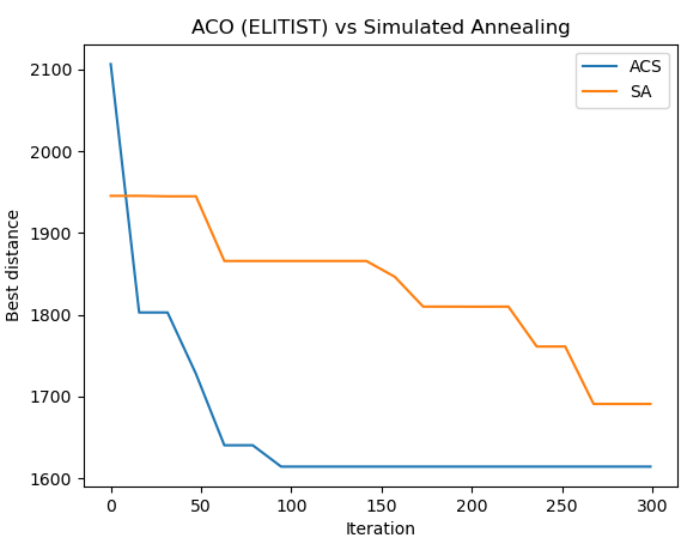
On the other hand, in the right chart (with 10 ants) we see some significant improvement over SA which seems to get a somewhat worst result, ACS got a much better result and much quicker with already around 100th iteration found the global optimum solution**, this shows how much the "colony" part in this optimization is important and playing a key rule here.**

Next, we will try the Elitist ACO variant and see how it performs compared to Simulated Annealing:

*Ant count: 1*

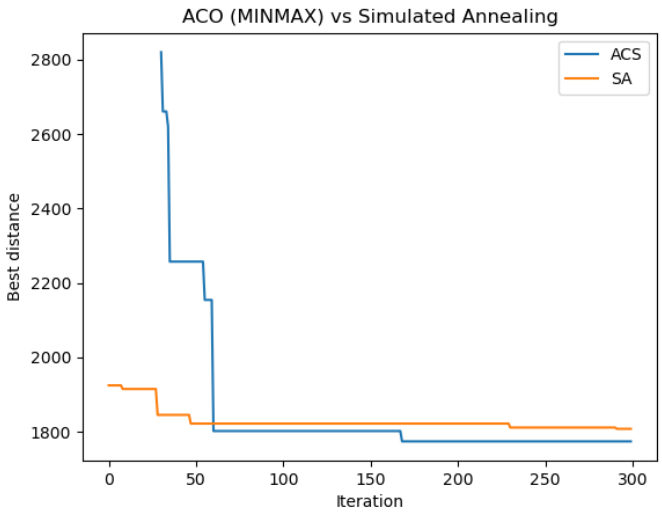



*Ant count: 10*

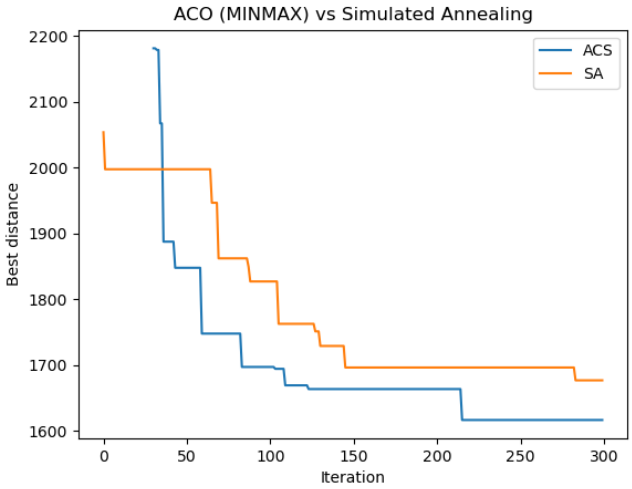


As expected with 1 ant Elitist will not perform well against SA but will still end up with a somewhat good approximation, with more iterations will probably do better.
Running again with 10 ants and 300 iterations (right chart): We can see that adding more ants significantly increases performance. Elitist seems to perform somewhat similar to ACS, perhaps with different parameters will see some significant changes, well check that as well later on.
Lastly, we will compare min-max with Simulated Annealing. Once again we will run with 1 ant and with 10 ants with 300 iterations:

*Ant count: 1*

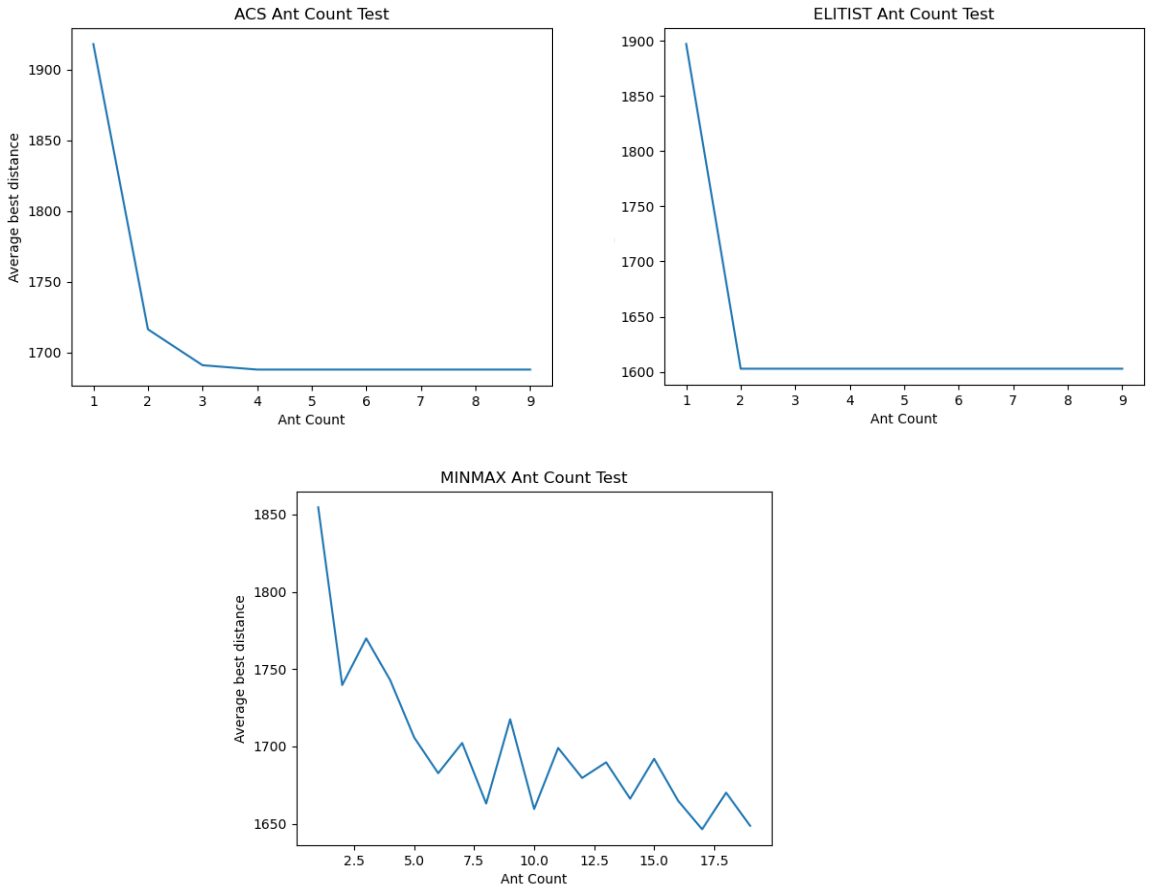



*Ant count: 10*




For our implementation of min-max, we took a small percentage of the first iterations to set the max pheromone amount, so we won't be returning any solution at those iterations, but as soon as we start exploring after the max pheromone was initially set we can see an immediate drop towards better solutions compared to a more gradual decrease in ACS and ELITIST since the ant will, with high probability choose paths of the best solution path to explore and thus improving the solution very quickly. In both cases (1 ant & 10 ants), we can observe the min-max Ant system usually gives a better solution for 300 iterations over SA. With a small downside of not being able to return solutions on the first iterations that are not that bad since those solutions are already random and insignificant.

Lastly, we will want to see how the number of ants affects each variant's performance compared to SA. For this, we will run each algorithm 10 times for each number of ants and get the average of all the best results.




for ACS and Elitist, it is clear that there is a threshold for how many ants are needed to get the optimal solution (for running with 50 iterations), but it does not necessarily mean that we shouldn't add more, since we can run this Optimization with concurrency and find an optimal solution fast with a lot of ants. Looking at the min-max chart we needed to run it max ant amount of 20! And we still didn't hit a threshold which the algorithm consistently returns the optimal solution. Running the algorithm with 20 ants took a lot of time. We will discuss how to improve the algorithm in the summary. Minmax seems to wander up and down regarding the number of ants it uses but it does seem to decrease with the more ants it uses, so we can comfortably say that in general for all 3 variants adding more ants will improve the performance of the algorithm but will require higher computation power.

## Let's sum things up

It is important to mention that since both Ant system and Simulated Annealing are using probability choices, we can get a lot of different results to our tests and we cannot conclude hard facts on what we saw here, but after running all the tests several times we got pretty much the same result every time.

So, we might ask, which is better Ant Systems or Simulated annealing? Of course, we cannot answer this question since we only tested it on the Travelling salesman problem although it is an NP-Hard problem we still might want to use either for a different problem to get better results from the other. For the simple traveling salesman problem, we can conclude that Ant system with

ants count > 5, we will get better results than simulated annealing for the long run, although Simulated annealing seems to give much better results at the first iterations compared to all 3 Ant system variants' which is somewhat important since we are comparing any-time algorithms.

So, in general for problems with a low variance of solutions aka (not a lot of local maximum-minimum that are sparse), we might consider using Simulated annealing. But we can argue about how we can assume something like that in the first place. For applications that are more likely to have a somewhat unknown 'environment,' we night consider using Ant Systems since we don't assume running for more time will less likely find as the global best solution but rather sending a lot of agents, we simply taking the best one or following the best one for their solution and trying to improve it even more.

For other applications like APSP (All Pairs Shortest Paths) although it is not an NP-Hard Problem, the ant colony system will probably still be a good choice for **very large** Graphs or somewhat more complex systems since we are introducing the randomness in this algorithm which might do better than traditional polynomial Algorithms like Bellman-Ford or Floyd—Warshall with some probability. Another set of problems that the Ant colony system will be a good fit for solving is Problems with multiple Objectives. Thus, since Any colony systems use 'agents' we could solve multiple Objectives in one iteration over iterating and solving one objective at a time since we can allocate *x* number of Ants aka 'agents' for each objective and run them as a colony.

## How can we improve?

Well, we might want to use different variants of Ant systems (yes there is more!). Another thing we could do; is if the pheromone update rules are global meaning we only update after a whole iteration, in this case, each ant is independent so we could use concurrency; each ant will run in her thread that way each iteration will end much faster and we will hopefully converge faster.
We could still use concurrency if the update rule is local but that will create a lot of critical sections in the code that we will need to handle. Also using concurrency will allow us to add more ants without reducing runtime that much since each Ant won't need to wait for other ants to finish their iteration.

➔ Code instructions are below:

## Running the project

Get the code from the git repo: https://github.com/Noamko/AntColonyOptimization

Prerequisites:

1. install python > 3.7
2. install NumPy, PyPlot matplotlib

```
pip3 install numpy
pip3 install matplotlib
```

3. Download the zip from git
4. Extract files

Running the code:

Running test 1:

In test 1 we compare ACO variants with Simulated annealing

Example: run test 1 with ACS variant

```
Python3 main.py test1 acs
```

Run test 1 with Elitist

```
Python3 main.py test1 elitist
```

Run test 1 with minmax

```
Python3 main.py test1 minmax
```

**After running Test 1 we would see some verbose in the terminal**

**And when finished 2 files will be added to the current directory.**

1. **Graph image with the found solution**
2. **The plot of ACO variant vs Simulated Annealing**

Running test 2:

In test 2 we compare how adding ants affect the overall result

Again, we can provide any function we want:

```
Python3 main.py test2 acs

Python3 main.py test2 elitist

Python3 main.py test2 minmax
```

**After running test 2 an image of the resulting plot will be added to the current directory.**

**Note**

- It is possible to play with the code; change some hyperparameters to see more results by adding more ants, iterations, etc…