Heriot-Watt University

Dissertation

# Ant Colony Optimisation Demonstration Software

*Author:*

Emilie Tavernier

*Supervisor:*

Dr. Michael Lones

*A dissertation submitted in fulfilment of the requirements for the degree of MSc. Artificial Intelligence*

*in the*

School of Mathematical and Computer Sciences

August 2021

## Declaration of Authorship

I, Emilie Tavernier, declare that this dissertation titled, 'Ant Colony Optimisation Demonstration Software' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: [illegible]

Date: 09/08/2021

## Abstract

The first Ant Colony Optimisation (ACO) algorithm was created in 1991 by Dorigo, Maniezzo and Colorni to solve the famous Traveling Salesman Problem (TSP). Since then, many variants were developed to tackle this problem and it quickly appeared that ACO could find other applications too, such as job scheduling or even image processing. Indeed, the strength of ACO lies in a simple and robust metaphor (ant foraging behaviour) which logic can be applied to a wide variety of problem set-ups. The simplicity of the metaphor can be misleading though, as one could easily miss its potential. For this reason, I think spreading awareness about ACO capacities would benefit greatly this field of study. Many small demonstration programs already exist but most of them focus on a single ACO implementation, generally to solve TSP or sometimes another specific problem. Furthermore, the lack of pedagogic explanations reserves these demonstrations to a few knowledgeable people. Following this observation, I decided to develop a more general demonstration software that would feature several ACO variants and problems to solve. This software would be design as a free learning tool to the destination of students, to help them grasp the full potential of ACO and thus encourage the use of ACO in future studies.

## Acknowledgement

I want to thank Michael Lones for supervising during this project, and for the continuous support including weekly meetings and helpful advice. I really enjoyed working with you.

I also thank Michael Lones and Andrew Ireland for their valuable feedback on the research report I wrote in preparation of this project.

Finally, I give my thanks to all the volunteers that took part in the user evaluations and helped me to improve the ACO demonstration software.

# Content

## List of figures

## List of tables

## Abbreviations

| | |
|---|---|
| **ACO** | **A**nt **C**olony **O**ptimisation |
| **AS** | **A**nt **S**ystem |
| **MMAS** | **M**ax-**M**in **A**nt **S**ystem |
| **ACS** | **A**nt **C**olony **S**ystem |
| **TSP** | **T**raveling **S**alesman **P**roblem |
| **JSP** | **J**ob **S**cheduling **P**roblem |
| **VRP** | **V**ehicle **R**outing **P**roblem |
| **SUS** | **S**ystem **U**sability **S**cale |
| **MVP** | **M**inimum **V**iable **P**roduct |
| **GUI** | **G**raphical **U**ser **I**nterface |

# Chapter 1 Introduction

Biologically inspired computing is a field of study which draws inspiration from biological models to solve computer problems. Ant Colony Optimisation (ACO) is a successful algorithm issued from this kind of research. It was initially used to approximate a solution to the famous problem of the “travelling salesman" and was later applied to various other problems. The clarity of the metaphor (ant behaviour) and its popularity among the researcher’s community with roughly 10,000 citations [1] make it an adequate example to introduce students to the bio-inspired computing field. For educational purpose, several demonstration software programs were made to follow ACO problem solving process. However, most of these only tackles one kind of optimisation problem and one version of the algorithm. This project's aim was to create a more general demonstration program that can truly show to a student the variety of ACO usages as well as the main variants of the algorithm.

This report will start with a literature review about the ACO algorithms and the problems we want to include in the software as well as a review of existing demonstration tools. The following section will detail the requirements analysis that was made in preparation of this project. After this, the methodology and technology adopted during the development phase and the resulting product will be discussed. The next section will present the design and results for the user evaluation that was performed to assess the viability of the application. Finally, I shall conclude and propose plans for future developments and accessibility methods.

# Chapter 2 Literature Review

## 2.1. Biologically inspired computation and ant behaviour

### 2.1.1 A quick overview on biologically inspired computation

Biologically inspired computation takes inspiration from multiple biological systems. The most famous example may be artificial neurone networks, which represent neurones has mathematical functions linked together in a similar fashion has in our brain. Genetic algorithms aim to create problem solutions by reproducing evolution mechanisms such as mutations and survival of the fittest. Another big source of inspiration lies in animal social conduct such as swarm or group behaviours. The idea is to study animals' methods to solve problems (food search, security, orientation…) and see if this can be applied to computer problems. Ant colony behaviour is an example that proved useful in that matter.

### 2.1.2 Ant behaviour and ACO

Ant Colony Optimisation (ACO) algorithms draw inspiration from **ants' foraging behaviour**. The process is as follows: an ant scouts the environment around the colony looking for food; If it found some, it will go back to the colony while dropping **pheromones** on its way; These pheromones attract others ants, leading them to the food; On their way back, they will themselves reinforce the track.

As a natural consequence, shorter paths will be favoured as more ants will complete it in a shorter amount of time, thus reinforcing them with pheromones quicker. Longer paths will be forgotten as pheromones gradually fade away (Figure 2-1).

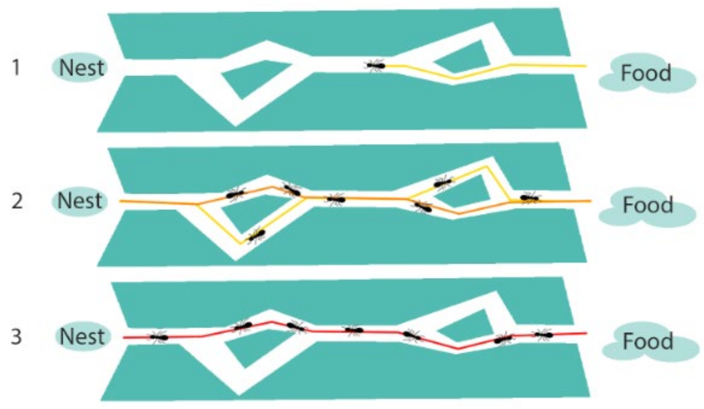


*Figure 2-1: Ant Foraging Behaviour*

This mechanism allowing individual agents to cooperate by modifying the environment is called **stigmergy**. Using this mechanism, a problem can be solved bit by bit. In this case, the ants leave markers which are not a solution by themselves but which will influence the group so that it ultimately finds an optimal answer.

Many variants of ACO have been applied to solve various problems. In this literature review, I shall present three of the most famous variants (Ant System, Min-Max Ant System and Ant Colony System) applied on the travelling salesman problem. Then, I shall discuss other possible usages for ACO: job scheduling, edge detection and vehicle routing problems.

## 2.2. Traveling Salesman Problem (TSP)

This section presents the traveling salesman problem and three variants of ACO (AS, MMAS and ACS) that were designed to solve it. Please note that all mathematical formulae in this section are from paper [2].

### 2.2.1 Problem description

The traveling salesman problem (TSP) is a famous optimisation problem defined as follows: A salesman needs to visit a set of cities. What is the shortest tour passing through all the cities once?

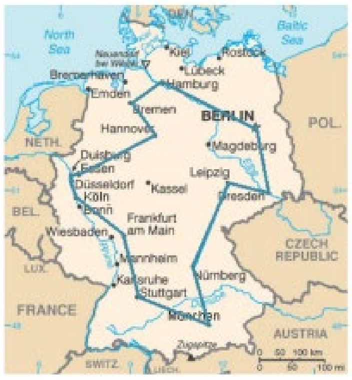


*Figure 2-2: TSP solution example for 12 German cities*

### 2.2.2 TSP Graph Representation

Ant Colony Optimisation (ACO) algorithms are generally used to solve discrete problems represented as graphs.

TSP can be represented as an **undirected weighted graph** where cities are vertices, and paths connecting them are edges. The weight of each edge corresponds to the length of the path. Ants can only travel and lay pheromones on these edges.

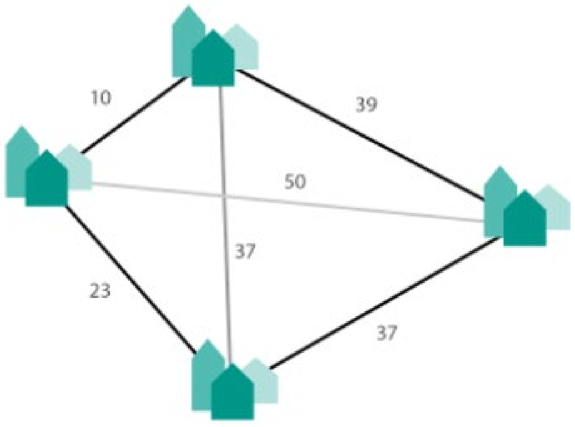


*Figure 2-3: Graph representation of TSP*

In figure 2-3, edges' **opacity is proportional to the pheromone concentration**. The darker is an edge, the more chance ants will take it.

### 2.2.3 Ant System

Ant System (AS) was the first ant colony optimisation algorithm proposed. It was developed by Dorigo, Maniezzo and Colorni in 1991 [3] to solve the **Traveling Salesman Problem (TSP)**. Paper [2] describes the general ant meta-heuristic as follows:

```
Algorithm 1: Ant meta-heuristic
Result: Optimisation problem solution approximation
Set parameters, initialize pheromone trail;
while stopping condition not met do
    ConstructAntSolutions ;
    UpdatePheromones ;
end
```

The stopping condition generally refers to a **maximum number of iteration** (running time) or a **stagnation state** (no improvements during n iterations). Each ant is assigned to a **random departure node**.

During an iteration, each ant constructs a solution step by step until the tour is completed. The probability for an ant k on node i to visit node j next is computed with the following formula:

$$p_{ij}^{k} = \begin{cases} \dfrac{\overbrace{\tau_{ij}^{\alpha}}^{\tau_{ij}\text{ pheromone concentration on edge ij}} \cdot {\overbrace{\frac{1}{d_{ij}}}^{d_{ij}\text{ distance between nodes i and i}}}^{\beta}}{\underbrace{\sum_{il,\ \text{l unvisited}} \tau_{il}^{\alpha} \cdot {\frac{1}{d_{il}}}^{\beta}}_{\text{Sum of } \tau_{il}\cdot\frac{1}{d_{il}} \text{ for all unvisited edge connected to node i}}} & \text{if j is unvisited yet,} \\ 0 \text{ otherwise.} & \end{cases}$$

**Alpha** and **beta** are parameters to control the weight of pheromone and distance between cities when deciding next destination. During the destination selection, a greater **alpha** means that edges with **lot of pheromones** will be favoured, while a greater **beta** will favour **closer cities**.

The pheromone concentration for edge ij is updated via the following formula:

$$\tau_{ij} \leftarrow (1-\underbrace{\rho}_{\rho\text{ the evaporation rate}}).\overbrace{\tau_{ij}}^{\tau_{ij}\text{ pheromone concentration at previous iteration}} + \overbrace{\textstyle\sum_k \Delta\tau_{ij}^k}^{\text{Sum of pheromones laid on the edge ij by each ant}}$$

The **evaporation rate**, [0: 1), allows paths to be forgotten over time. If it is low, ants will be slow, and may fail, to converge to a common path. If it is high, they will be quick to converge but may settle too rapidly to a suboptimal solution.

At each iteration, each ant adds new pheromones on the edges they used in their solution. This is described by the following formula:

$$\Delta\tau_{ij}^k = \begin{cases} \overbrace{Q}^{Q\text{ a constant}}/\overbrace{L_k}^{L_k\text{ the length of ant k solution}} \text{ if ant k used edge(i,j) in its solution,} \\ 0 \text{ otherwise.} \end{cases}$$

Rising **Q** will result in a bigger pheromone addition which will reinforce an edge more each time an ant traverses it. However, this may lead to a snowball effect where the favourite path will quickly outshine all other solutions, and result in premature convergence.

### 2.2.4 Max-Min Ant System

Max-Min Ant System (MMAS) is a variant of ACO algorithms proposed by Thomas Stützle and Holger H. Hoos. It aims to enhance ACO performances thanks to a stronger exploitation of the best known solution. The algorithm structure is the same as **AS**. However, MMAS differs in three points:

- Only the **best ant** will perform the pheromone update. The best ant can be defined as the iteration best or the global best (best from the beginning).
- Pheromone concentration is **bounded** ([$\tau_{min}$; $\tau_{max}$]) to limit stagnation.
- The initial value of pheromones is set to $\boldsymbol{\tau_{max}}$**,** to encourage exploration at the beginning of the algorithm.

This translates in the pheromone update equation as:

$$\tau_{ij}^{best} \leftarrow [(1-\underbrace{\rho}_{\rho \text{ the evaporation rate}}).\overbrace{\tau_{ij}}^{\tau_{ij} \text{ pheromone concentration at previous iteration}} + \overbrace{\Delta\tau_{ij}^{best}}^{\text{Pheromones laid on the edge ij by best ant.}}]_{\tau_{min}}^{\tau_{max}}$$

With the bounding operator defined as:

$$[x]_{min}^{max} = \begin{cases} min \text{ if } x < min, \\ max \text{ if } x > max, \\ x \text{ otherwise} \end{cases}$$

And the quantity of added pheromone computed as:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/\overbrace{L_{best}}^{L_{best} \text{ the shortest solution length}} \text{ if edge(i,j) belong to best tour,} \\ 0 \text{ otherwise.} \end{cases}$$

### 2.2.5 Ant Colony System

Ant Colony System (ACS) is a variant of ACO proposed by Dorigo and Gambardella in 1996. The main difference with AS and MMAS is that it relies on **two kinds of pheromone updates**:

- A **local update** performed by each ant on the last edge it visited, at each construction step. This one involves a decay coefficient.
- An **offline update** at the end of an iteration. Similarly to MMAS, this one is performed only by the best ant (best-so-far or iteration-best).

The formula for the local update is as follow:

$$\tau_{ij} = (1 - \overbrace{\phi}^{\phi \text{ the decay coefficient}}).\tau_{ij} + \underbrace{\phi.\tau_0}_{\tau_0 \text{ pheromone initial concentration}}$$

And the formula for the offline update is:

$$\tau_{ij} = \begin{cases} (1 - \overbrace{\rho}^{\rho \text{ the evaporation rate}}).\tau_{ij} + \rho.\Delta\tau_{ij} \text{ if edge(i,j) belong to best tour,} \\ \underbrace{\tau_{ij}}_{\tau_{ij} \text{ pheromone concentration on edge ij}} \text{ otherwise.} \end{cases}$$

With:

$$\Delta\tau_{ij} = 1/\overbrace{L_{best}}^{L_{best} \text{ the shortest solution length}}$$

Another notable difference between ACS and the other ACO variants concerns the selection of the next destination during the construction of a solution. In ACS, it involves a pseudorandom proportional rule where the probability depends on a random variable **q uniformly distributed on [0,1]**. A parameter **$q_0$** serves as a **threshold** to choose the next destination.

If **q** $\leq$ **q₀**, the next destination j is the city for which edge ij has the greatest product between its pheromone concentration and its heuristic information weighted with the exponent beta.

$$j = \arg\max_{\text{il, l unvisited}}\{\tau_{il}\eta_{il}^{\beta}\} \qquad \text{with} \qquad \overbrace{\eta_{il}}^{\eta_{il}\text{ heuristic information on edge il}} = \frac{1}{\underbrace{d_{il}}_{d_{il}\text{ distance between node i and l}}}$$

A beta equal to zero means we will choose the edge with the greatest pheromone concentration. Rising beta will add more weight to the heuristic information to favour cities that are close to each other.

If **q > q₀**, the same method as in Ant System (AS) is used to choose the next destination. The probability for an ant k on node i to visit node j next is computed with the following formula:

$$p_{ij}^{k} = \begin{cases} \dfrac{\overbrace{\tau_{ij}^{\alpha}}^{\tau_{ij}\text{ pheromone concentration on edge ij}} . \overbrace{\eta_{ij}^{\beta}}^{\eta_{il}^{\beta}\text{ heuristic information associated to edge ij}}}{\underbrace{\sum_{il,\ \text{l unvisited}} \tau_{il}^{\alpha}.\eta_{il}^{\beta}}_{\text{Sum of }\tau_{il}^{\alpha}.\eta_{il}^{\beta}\text{ for all unvisited edge connected to node i}}} & \text{if j is unvisited yet,} \\ 0 \text{ otherwise.} \end{cases}$$

Again, **alpha** and **beta** are parameters to control the weight of the pheromone concentration and the distance between cities when deciding the next destination. If **alpha equals 0**, we will use the edge with the **greatest amount of pheromone**, if **beta equals 0**, we will choose the closest city.

### 2.2.6 Summary

Table 2-1 sum up the main similarities and differences between the three ACO variants we described.

*Table 2-1: ACO variants particularities*

| | **Pheromones updates** | **Next destination choice** |
|---|---|---|
| **AS** | When? At the end of an iteration<br>Whom? All ants<br>Bounded? No | Probabilistically chosen:<br>- 0 if already visited<br>- Proportional to the pheromone concentration<br>- Inversely proportional to the distance |
| **MMAS** | When? At the end of an iteration<br>Whom? Best ant<br>Bounded? Yes | Similar to AS |
| **ACS** | When?<br>- At each construction step (local)<br>- At the end of an iteration (offline)<br>Whom?<br>- All ants (local)<br>- Best ant (offline<br>Bounded? No | Dependant on a pseudorandom proportional rule based on a random variable uniformly distributed on [0, 1] |

## 2.3. Job Scheduling Problem (JSP)

Several studies such as [4, 5, 6] show it is possible to solve job scheduling problems using ACO algorithms. This section explains the principle of JSP and proposes a solving method using ACS inspired by [6, 5].

### 2.3.1 Problem description

The job scheduling problem (JSP) is a common problem in computer science, though it could also find uses in other fields such as in project management or in manufacturing (assembly lines) for instance. Several variants exist with more or less constraints but the main idea is as follows:

A set of jobs needs to be executed. Each job is composed of a set of tasks of varying duration which needs to be performed in a specific order (**precedence constraint**). Tasks from the same job can only be executed one at a time on a specific machine from a given set (resources). The goal is to **minimize the makespan**, that is the time to complete the whole process from start to end.

[7] illustrates this problem with a set of three jobs composed of two to three tasks. A task is described by the machine it needs to be executed on, and its duration, as follows: (machine, duration).

Given the following jobs:

- job 0 = [(0,3), (1,2), (2, 2)]
- job 1 = [(0,2), (2,1), (1, 4)]
- job 2 = [(1,4), (2,3)]

Figure 2-4 shows two feasible schedules for machines M0, M1 and M2 but the second one having a shorter makespan is the best option.

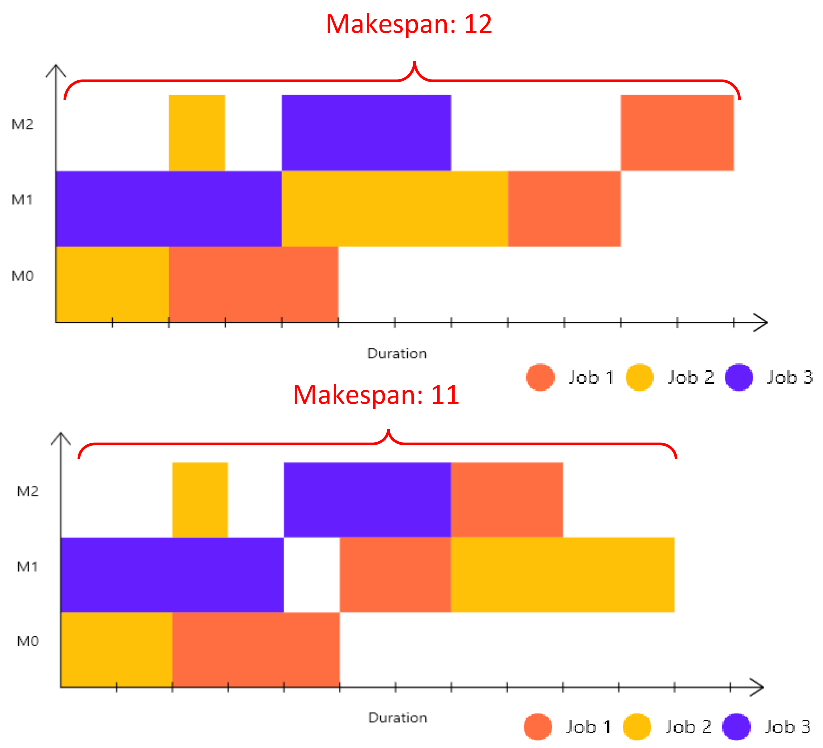


*Figure 2-4: Two feasible schedules*

### 2.3.2 JSP Graph Representation

Let’s consider the set of jobs described in section 2.3.1:

- job 0 = [(0,3), (1,2), (2, 2)]
- job 1 = [(0,2), (2,1), (1, 4)]
- job 2 = [(1,4), (2,3)]

Study [6] proposes to represent this problem as a graph suitable to ACO (Figure 2-5). Each job is represented as a set of nodes corresponding to its composing tasks. Tasks from the same job must be executed in a given order (**precedence constraint**) so we connect them unidirectionally. Tasks from different jobs are all connected bidirectionally. A starting node is added to serve as departure point for all ants.

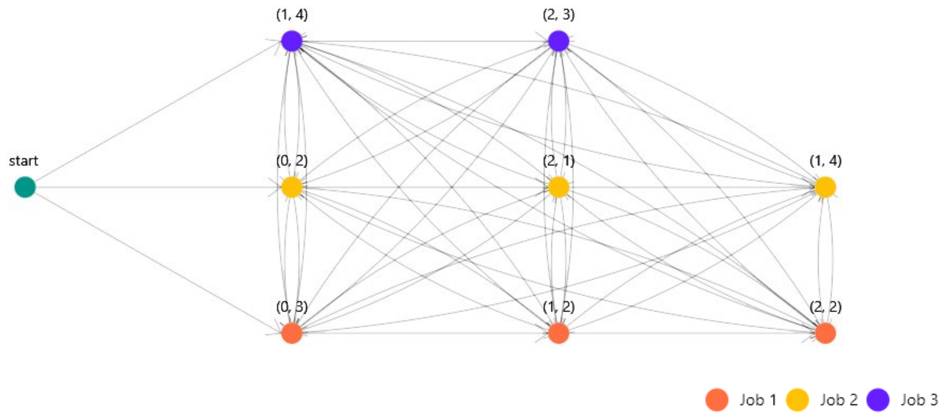


*Figure 2-5: Graph representation for JSP*

An ant constructs a solution by visiting all nodes exactly once. To respect the precedence constraint, an ant cannot visit a node if it has not visited same job predecessors yet. For instance, the ant cannot go to job 3 task (2,3) before job 3 task (1,4).

After completing a solution, we can translate it to an actual schedule by respecting the order of the execution of tasks (order in which they were visited) on their respective machine.

Here is a complete solution:

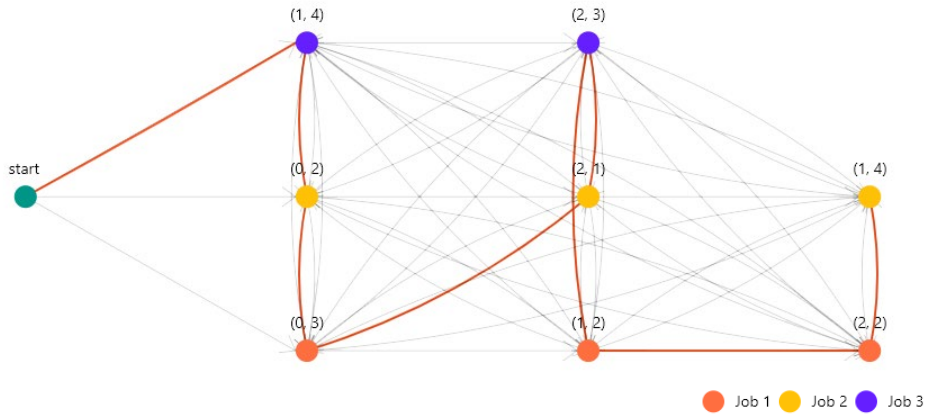


*Figure 2-6: An ant solution for JSP*

And the corresponding schedule:

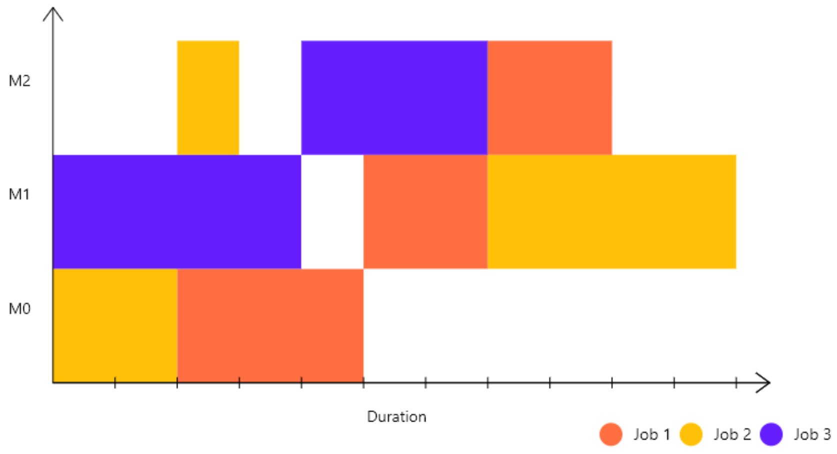


*Figure 2-7: Ant solution corresponding schedule*

A solution is evaluated based on the **makespan** of the corresponding schedule (the shorter, the better).

Ant Colony System (ACS) was initially proposed to solve TSP but can be adapt really easily to job scheduling, with only a few tweaks.

The first difference to take into account is that JSP requires to respect the **precedence constraint**. This play during the construction of a solution when selecting the next destination:

$$p_{ij}^{k} = \begin{cases} \dfrac{\overbrace{\tau_{ij}^{\alpha}}^{\tau_{ij}\text{ pheromone concentration on edge ij}} \cdot \overbrace{\eta_{ij}^{\beta}}^{\eta_{il}^{\beta}\text{ heuristic information associated to edge ij}}}{\underbrace{\sum_{il,\ l \text{ unvisited}} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}_{\text{Sum of } \tau_{il}^{\alpha}.\eta_{il}^{\beta} \text{ for all unvisited edge connected to node i}}} & \text{if j is unvisited yet AND its same jobs predecessors were already visited,} \\ 0 \text{ otherwise.} & \end{cases}$$

The second difference concerns the **heuristic information** associated to the edges ij. Here, we do not have a physical distance between nodes as in TSP. Instead, we define a **machine delay** as the time to wait to execute task j on its associated machine, as suggested by [6]. Note that we ignore any additional potential delay caused by the precedence constraint. The machine delay can be translated as the following formula:

$$machine\_delay_{ij} = \begin{cases} task\_duration_i \text{ if tasks i and j share the same machine,} \\ 0 \text{ otherwise.} \end{cases}$$

The heuristic information on edge ij is then defined as **the sum of the machine delay and the duration of task j** as follow:

$$\eta_{ij} = machine\_delay_{ij} + task\_duration_j$$

Note that this definition is not the only valid option. Article [5] proposes to define the heuristic information as **task j duration** to favour the selection of the **longest tasks first** or as its **inverse** to favour the **shortest tasks first**.

## 2.4. Edge detection

A creative way to use ACO concerns image processing. For instance, several studies successfully applied AS or ACS in edge detection [8, 9, 10]. This section will briefly describe the edge detection problem and explain the ACS adaptation proposed by article [8].

### 2.4.1 Problem description

In image processing, edge detection consists of identifying the points of an image where an **abrupt variation in light intensity** occurs.

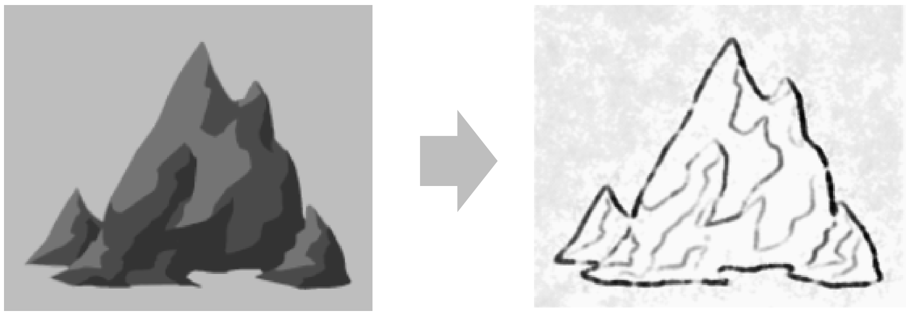


*Figure 2-8: An image (left) and the edges detected by ACS (right)*

### 2.4.2 Image graph representation

A grayscale image can be represented as a **M x N matrix of pixels**. To adapt the problem to ACO, [8] proposes to represent an image as a graph where pixels are nodes connected to adjacent pixels (Figure 2-9).

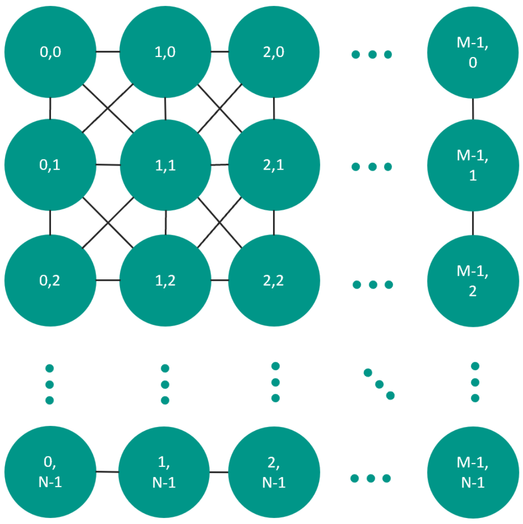


*Figure 2-9: Graph representation for edge detection*

This configuration is quite similar to TSP. However, the TSP graph was fully connected while our image is represented as a **partially connected graph**: an ant can only travel from one pixel to a direct neighbour.

In TSP, a solution was evaluated according to its length (the shorter, the better). Here, we substitute the length by heuristic information derived from the **intensity variation** (the higher the better) along a given path.

### 2.4.3 ACS for edge detection

Ant Colony System (ACS) was initially proposed to solve TSP but can be adapted to edge detection with a few tweaks.

As for TSP initialisation, each ant is positioned on a **random pixel** at the beginning of the first iteration. However, a first difference is that the graph representing an image is far bigger than the one for TSP. Hence, we stop an iteration after a **pre-defined number of construction steps** rather than after the ants visited all nodes.

For edge detection, evaluating a path by its length does not make much sense. Instead of the distance between nodes, we will thus rely on statistics about the **intensity variation** around a pixel to choose the next destination.

To this purpose, [8] defines the heuristic information at pixel (i,j) as:

$$\eta_{i.j} = \frac{V_c(I_{i,j})}{V_{max}}$$

With $I_{i,j}$ the intensity of pixel (i,j). Vc is a function computing the intensity variation based on pixel (i,j)'s neighbours:

$$V_c(I_{i,j}) = \left|I_{i-1,j-1} - I_{i+1,j+1}\right| + \left|I_{i-1,j} - I_{i+1,j}\right| + \\ \left|I_{i-1,j+1} - I_{i+1,j-1}\right| + \left|I_{i,j-1} - I_{i,j+1}\right|$$

And the normalization factor $V_{max}$ which designates the maximum intensity variation for the image.

The heuristic information appears in the formula to choose the next destination for construction step n:

$$p^{(n)}_{(i_0,j_0),(i,j)} = \frac{\overbrace{\left(\tau^{(n-1)}_{i,j}\right)^{\alpha}}^{\tau_{ij}\text{ pheromone concentration on pixel (i, j)}}\overbrace{\left(\eta_{i,j}\right)^{\beta}}^{\eta_{ij}\text{ heuristic information for pixel (i, j)}}}{\sum_{(i,j)\in\underbrace{\Omega_{(i_0,j_0)}}_{\Omega_{(i_o,j_o)}\text{ neighbourhood of pixel }(i_0,\, j_0)}}\left(\tau^{(n-1)}_{i,j}\right)^{\alpha}\left(\eta_{i,j}\right)^{\beta}}$$

Again, **Alpha** and **beta** are parameters to control the weight of pheromone and the heuristic information. During the destination selection, a greater **alpha** means that pixels with **lots of pheromone** will be favoured, while a greater **beta** will privilege pixels with a **high surrounding intensity variation**.

Finally, we remind that ACS uses two kind of pheromone updates. The **local update** involving a **pheromone decay** coefficient is the same than for TSP:

$$\tau_{i,j}^{(n)} = (1 - \overbrace{\varphi}^{\varphi \text{ the decay coefficient}}) \cdot \tau_{i,j}^{(n)} + \varphi \cdot \underbrace{\tau_{\text{init}}}_{\tau_{init} \text{ pheromone initial concentration}}$$

However, the **offline update** involving a **pheromone evaporation** rate is slightly different because there are no “best” ants in edge detection contrary to TSP. Hence, the offline update is here **performed by all ants** like the local update, so the formula become simply:

$$\tau_{i,j}^{(n)} = (1 - \overbrace{\rho}^{\varphi \text{ the evaporation rate}}) \cdot \tau_{i,j}^{(n-1)} + \rho \cdot \sum_{k=1}^{K} \underbrace{\Delta\tau_{i,j}^{(k)}}_{\Delta\tau_{i,j}{}^{(k)} \text{ average heuristic information on the kth ant's tour}}$$

### 2.4.4 Another type of image processing: Edge linking

Edge detection is not the only use there is for ACO in image processing. For instance, study [11] proposes a method for edge linking in an attempt to retrieve missing edge parts after using any edge detection algorithm. The entry points for this method are a sobel edge image and a grayscale visibility matrix which is kind of equivalent to the heuristic information described above for edge detection. Based on these, an initial pheromone matrix will be computed giving higher concentration to current known edges. These edges' endpoints will serve as the departure for ant agents. Construction steps will favour pixels closest to other endpoints so that ants tend to connect them together. Figure 2-10 shows the method process and result obtained in study [11] (from left to right: initial image, sobel edge, edge linking detection, final corrected edges).

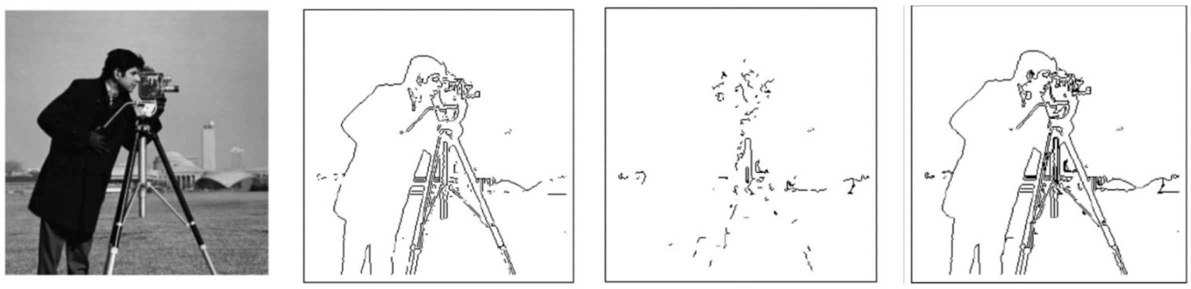


*Figure 2-10: Edge linking process from study [11]*

## 2.5. Vehicle routing problem

### 2.5.1 Problem description

Vehicle Routing Problem (VRP) is a generalisation of TSP which first appeared in G. Dantzig and J. Ramser paper in 1959 [12].

The classical VRP, sometime referred to as Capacited VRP (CVRP), stages a firm that needs to deliver products from its depot to a number of customers. To do so, you dispose of a set of vehicles with equal limited capacity. The objective of the problem is to find the least costly set of routes (e.g. time-wise or money-wise) starting and ending at your depot, to deliver all your customers. A customer can only be visited by one vehicle.

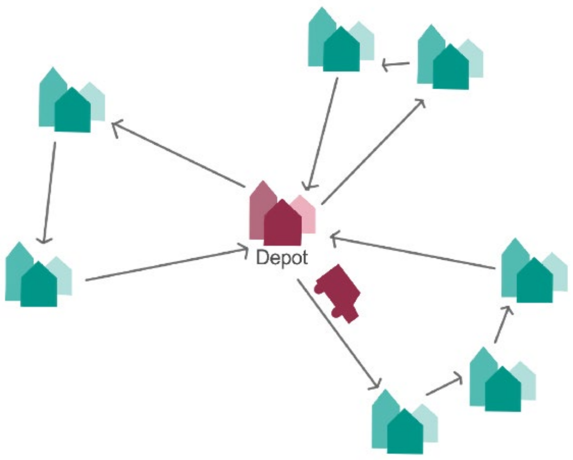


*Figure 2-11: VRP solution example*

There exist many variants of this problem as described in paper [13]. Some of these propose new characteristics from real-life scenario such as multiple vehicle capacities or multiple depots for instance. Other variants also introduce additional constraints such as delivery time windows varying from one customer to another.

The graph representation from this problem is quite similar to TSP except that we need to identify which nodes correspond to depots. Additional information may be attached to the nodes depending on the constraints (e.g, time window, quantity to deliver…).

### 2.5.2 ACO for VRP

Paper [14] proposes the following method to solve this problem using ACO:

An ant, representing a vehicle, travels from one city to another until all cities are visited to construct a solution, quite similarly to TSP solving method. The difference however is that each time the vehicle reaches its capacity constraint, it goes back to a depot before selecting another city to visit. Of course, the trip starts and ends at a depot. At the end of the tour, another ant departs to construct its own solution. Two pheromones update occurs: a local update by each ant after its tour and an offline update by the best ant after m solution were constructed.

### 2.5.3 Improvement strategies

An interesting point stated by paper [14] is that this process can be enhanced using route improvement strategies. Two strategies are retained:

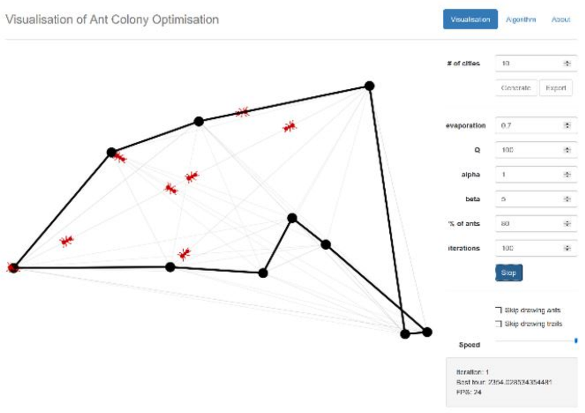
The first strategy consists of comparing obtained routes with all variants attained by inverting cities' location pairwise. For instance, let's say a vehicle visits in order the set of cities {1, 2, 3}. Pairwise exchanges gives three other possibilities {(2, 1, 3), (3, 2, 1), (1, 3, 2)} that are evaluated to keep only the best. This strategy ensures more thorough exploration of the set of possible combination.

The second strategy is to limit the number of next potential cities to visit by allowing only the n closest. This prevent wasting time exploring combinations that are most likely sub-optimal.

## 2.6. Existing demonstration software

There already exist various small demonstration software for ACO. In this section, you will find a selection of interesting ones targeting problems that were previously described.

The first demonstration (Figure 2-10) is about AS solving TSP and is executable directly online. The visual representation featuring ants, pheromone trails and current best route is really clear and easy to understand. ACO parameters can be set to different values as well as the number of cities for TSP. Cities are then placed randomly. The simulation is like "real-time" but with easily controllable speed.

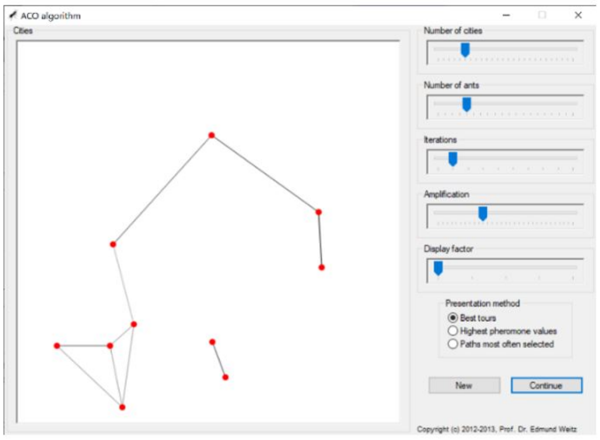


# of cities: 10
evaporation: 0.7
Q: 100
alpha: 1
beta: 5
% of ants: 80
iterations: 100
☐ Skip drawing ants
☐ Skip drawing trails

Iteration: 1
Best tour: 2364.028534354481
FPS: 24

*Figure 2-12: AS solving TSP demonstration software [15]*

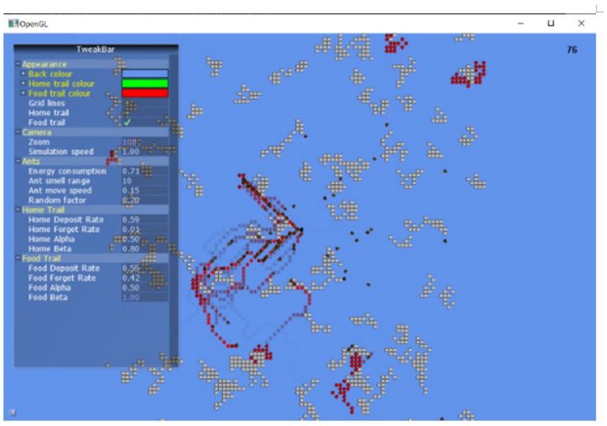
The second demonstration (Figure 2-11) is also targeting TSP. You can easily run it after downloading an executable file. Here, ants are not represented and the choice was made to display best tour or pheromones separately. The overall result is lacking clarity when compared to the previous demonstration but here, the execution is step by step. This feature could be interesting to allow a student to fully control the execution which could be associated with popping information.



Presentation method: [✓ Best tours] Highest pheromone values / Paths most often selected

*Figure 2-13: ACO solving TSP demonstration software [17]*

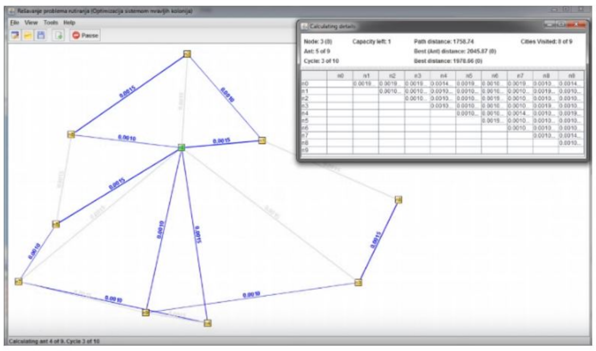
The third demonstration (Figure 2-12) shows ACO in the process of food foraging. This one is also accessible via a downloadable .exe. This is not a common application for ACO and we don't have much control on the execution flow. However, the visual really helps to understand how the metaphor of the ant colony works and how the movements are globally affected by various parameters.



- Appearance
  - Back colour
  - Home trail colour 🟩
  - Food trail colour 🟥
  - Grid lines
  - Home trail
  - Food trail ✓
- Camera
  - Zoom 100
  - Simulation speed 1.00
- Ants
  - Energy consumption 0.71
  - Ant smell range 10
  - Ant move speed 0.15
  - Random factor 0.20
- Home Trail
  - Home Deposit Rate 0.59
  - Home Forget Rate 0.01
  - Home Alpha 0.50
  - Home Beta 0.80
- Food Trail
  - Food Deposit Rate 0.55
  - Food Forget Rate 0.42
  - Food Alpha 0.50
  - Food Beta 1.00

Figure 2-14: Ant foraging demonstration software [16]

The fourth demonstration (Figure 2-13) targets VRP solving. Source code is accessible via a GitHub repository. The simulation features pheromones and current path construction but it may be lacking some visual cues for a novice user. The simulation is like "real-time" with possible pausing. Speed is modifiable via a hidden menu, which is a little less practical than in first simulation.



Node: 3 (0) Capacity left: 1 Path distance: 1758.74 Cities Visited: 8 of 9
Ant: 5 of 9 Best (Ant) distance: 2045.87 (0)
Cycle: 3 of 10 Best distance: 1978.66 (0)

| | n0 | n1 | n2 | n3 | n4 | n5 | n6 | n7 | n8 | n9 |
|---|---|---|---|---|---|---|---|---|---|---|
| n0 | | 0.0019... | 0.0019... | 0.0019... | 0.0014... | 0.0019... | 0.0010... | 0.0019... | 0.0010... | 0.0014... |
| n1 | | | 0.0010... | 0.0010... | 0.0010... | 0.0010... | 0.0010... | 0.0010... | 0.0019... | 0.0010... |
| n2 | | | | 0.0010... | 0.0010... | 0.0010... | 0.0015... | 0.0010... | 0.0010... | 0.0010... |
| n3 | | | | | 0.0010... | 0.0010... | 0.0010... | 0.0010... | 0.0019... | 0.0010... |
| n4 | | | | | | 0.0010... | 0.0010... | 0.0014... | 0.0010... | 0.0019... |
| n5 | | | | | | | 0.0015... | 0.0010... | 0.0010... | 0.0010... |
| n6 | | | | | | | | 0.0010... | 0.0010... | 0.0010... |
| n7 | | | | | | | | | 0.0010... | 0.0014... |
| n8 | | | | | | | | | | 0.0010... |
| n9 | | | | | | | | | | |

Calculating ant 4 of 9, Cycle 3 of 10

Figure 2-15: ACO solving VRP demonstration software [18]

Finally, I did not find accessible code for a demonstration of image processing using ACO. However, we found a nice video demonstrating edge detection (Figure 2-14). Though we don't have any control, the visual showcasing real-time evolution at different levels (original image, pheromone and edges) really helps understanding the process.

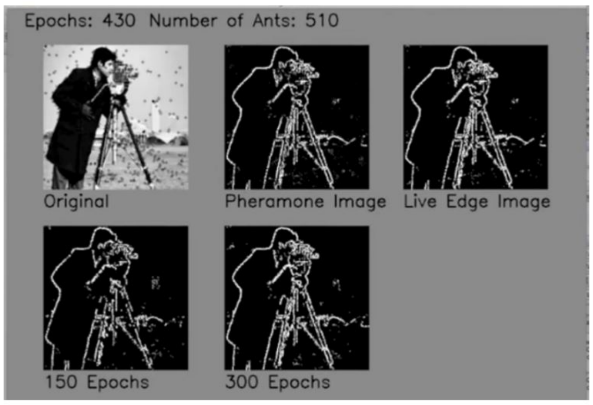


Figure 2-16: ACO for edge detection demonstration software [19]

Table 2-2 sums up the main pros (green) and cons (red) of the demonstrations described previously.

*Table 2-2: Demonstration software pros and cons*

| **Source** | **Targeted problem** | **Accessibility** | **Visual** | **Execution** | **Pedagogy** |
|---|---|---|---|---|---|
|  | TSP | Web[1] | Clear | "Real-time", Easy speed control | Algorithm explained but hard to understand |
|  | TSP | .exe | Lacking | Step by step | No explanation |
|  | Ant foraging | .exe | Clear | No control | No explanation |
|  | VRP | Source code | Lacking | "Real-time", hidden speed control | No explanation |
|  | Edge detection | None | Clear | No control | No explanation |

To conclude, there exists multiple ACO demonstration software. However, they are all focussing on a really specific problem and only implement one version of ACO. The main issue is that none of them offers enough explication about ACO, so a novice would need additional material or a teacher to really understand what is happening on screen. Currently, there is no educational software that covers ACO thoroughly. I believe it would be greatly beneficial for a student wishing to learn about ACO to provide him with a standalone well-explained demonstration tool which allows witnessing the main ACO variants on several representative problems.

[1] Web is the best option in term of accessibility as it allows anybody to use the software, provided they have an internet access. An executable file is a little less practical as it is generally restricted to the OS it was built for.

# Chapter 3 Requirement analysis

This project aims to correct the current lack of a general ACO learning tool. This section will describe in more details the goal, accessibility modalities and stakeholders for this project before identifying the priorities.

## 3.1. Project goal

The goal of this project is to create a standalone learning tool for ACO. Already existing demonstration software do not demonstrate the variety of ACO algorithms possible usages and are lacking in the pedagogic aspect. This project aims to tackle these two issues.

Hence, the software should allow a user to test ACO on various problems such as TSP, job scheduling, vehicle routing or image processing. The user should be allowed to change ACO related parameters and to select between a few variants of the algorithm to witness how this would affect the performance.

Additionally, extra-care should be given to adding clear written explanations in order for the software to be an effective learning tool. This includes detailed descriptions of the problems, algorithms and parameters featured in the application.

## 3.2. Accessibility

Ideally, the application should be usable as a website and as an executable file. Indeed, a website like the one for demonstration app [15] is the most accessible solution for a student but it needs to be hosted on a server. On the other hand, an executable file stays quite easy to run with no installation. It could serve as a safety backup in case the server goes down for instance. The downside however is that several executables would be needed to be compatible with different OS (e.g., Windows, Linux and Mac).

For these reasons, it would be beneficial to have these two options available. Consequently, the project development should follow a cross-platform approach. The programming language Dart, released in 2011 by Google, is optimised for cross-platform applications making it a potentially adequate option.

In the end, this application should be accessible via a public Github repository containing the link to the website, an executable file and the source code of the project.

## 3.3. Stakeholders

Several parties could be affected by the outcome of this project. Table 3-1 identify these people and the associated requirements.

*Table 3-1: Project stakeholders*

| Stakeholders | Requirements |
|---|---|
| Project author | Successful project |
| Project supervisor | Well defined project |
| Heriot Watt University | Presentable work and good dissertation |
| Computer science students | Free and accessible learning tool |
| Teacher in bio-inspired computation | Free and accessible teaching tool |

Additionally, teachers or researchers may want to take over this project and add functionalities for teaching or demonstration purpose in the future. To keep this possibility open, the source code should be clean, well-documented and open-source.

## 3.4. Priorities (MoSCoW method)

It is important to define the priorities and limits of this project. This section describes the features that should be included and the ones that could be interesting given more time. Table 3-2 classes them in term of priority using the MoSCoW method.

**ACO Implementation**

This project aims to demonstrate various implementation of ACO. I want to show the three most famous ACO variants described in Chapter 2: ant system, min-max ant system and ant colony system. To truly demonstrate these algorithms' behaviour, I want the algorithms parameters described in the literature (number of ants, evaporation rate, …) to be configurable to show how they impact the execution. Of course, it would be interesting to implement other variants in the future given more time.

**Problem solving demonstration**

The following choice concern the selection of problems (described in Chapter 2) we want to demonstrate ACO on. The natural first choice is the most iconic problem of the traveling salesman, which was the initial target for ant systems. Vehicle routing and job scheduling are other well-known problems we would like to include. However, their solving method being quite close to TSP, they are not our top priority. We think it would be really interesting to demonstrate a completely different type of problem such as image processing. In this matter, edge detection seems a better choice than edge linking as the output visual is clearer. Given more time, proposing something similar to the demonstration [16] could be interesting to help understanding the ant metaphor.

**Accessibility**

In section 3.2, we talked about having both a web-based app and an executable. For accessibility purpose we may want to prioritize the web support but both are fine as long as we have at least one working.

**Graphical user interface**

The main concern for the GUI is to have clear explanations of features and during demonstration for a student to understand ACO. The "real-time" demonstration should be visual and clear. It would be really interesting to allow experiments with various parameters to run parallelly to compare them but we likely won't have time for this.

**Additional features**

The problem space should be designable to some extent to run various experiments. Usually, the demonstration apps allow parameters configuration (e.g., TSP number of cities) and place them generate the map randomly from there [15, 17, 18]. If time allows, it would be great to push this customisation further (e.g., user defined cities' coordinates). For image detection, we simply need to allow the user loading his own images. Lastly, allowing to save an experiment's parameters or the full execution would be a nice feature but I probably won't have time.

*Table 3-2: MoSCoW classification*

| **Requirements** | **Priorities** |
|---|---|
| **ACO implementation** | |
| Ant System | Must |
| Max-min Ant System | Must |
| Ant Colony System | Must |
| Other ACO variants | Won't |
| Configurable parameters (number of ants, number of iteration, evaporation rate, …) | Must |
| **Problem solving demonstrations** | |
| TSP | Must |
| Job scheduling | Should |
| Vehicle routing | Should |
| Edge detection | Must |
| Other problems (edge linking, ant foraging…) | Won't |
| **Accessibility** | |
| Web-based | Must |
| Executable | Should |
| **Graphical user interface** | |
| Clear Explanations panels | Must |
| "Real-time" visualisation of ACO execution | Must |
| ACO parallel execution with different parameters | Won't |
| **Additional features** | |
| Saving experiment parameters and/or execution | Won't |
| Randomize problem space design | Must |
| "Fully" editable problem space design (e.g., choosing TSP cities' position) | Could |

# Chapter 4 Implementation

## 4.1. Methodology

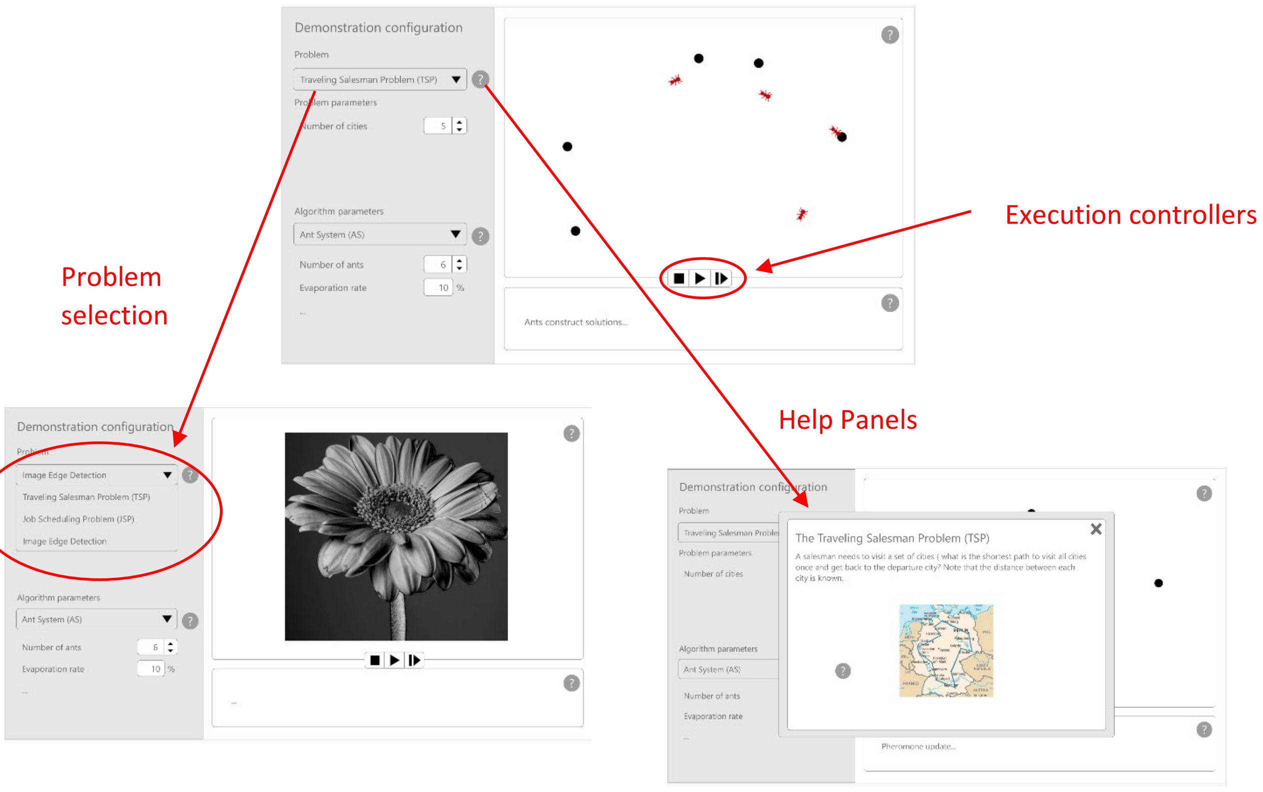
### 4.1.1 UI design

One of the first steps of the project development was to prototype the GUI to validate with the project supervisor if all necessary features were included as we both intended. Additionally, it would help to test the interface usability (intuitivity and user friendliness) and serve as a plan for the actual development of the UI.

I initially intended to use Figma, a vector graphic editor specialized in prototyping web-based user interface. The advantage of using such a tool over simple hand drawing or classical software such as photoshop or inkscape is that it allows to re-create easily dynamic behaviours such as scrollbars or dropdowns to simulate user interactions. However, I then realised my project was not exactly a classical website were most of the interactions consist in navigating between pages. Instead, I wanted to be closer to a desktop application which happens to be usable on a web browser. My experience on Figma told me this was not that adequate to my vision, so I decided to try a new solution of UI/UX design instead: Adobe XD.

It turns out Adobe XD was not that adapted either. In fact, it is a tool with a lot of potential but a really important feature is missing and asked by the community: cross components interactions. A component is a set of elements (e.g., texts, shapes, …) for which one can define a state. Interactions are simulated by changing the component state (hidden/shown elements, colours, position, etc…). The problem currently is that an interaction can only influence the elements inside the component for which it was defined. If one wants to hide a button when selecting a specific item in a dropdown, both needs to be in the same component. This becomes quickly unmanageable as it leads to enormous components where multiple elements must trigger various actions which multiply the number of states with new actions to define, and so on…

Because I could not afford to test multiple software and drop behind my schedule too much, I had to accommodate with it. I was able to prototype the overall design of the interface (Figure 4-1) and the main interactions for my application (problems and algorithms selection, help buttons, …). Despite a few minor bugs, this was enough at this stage to help me plan the UI and share my idea with my supervisor.



Number of cities 5

Number of ants 6

Evaporation rate 10 %

Ants construct solutions...

The Traveling Salesman Problem (TSP)

A salesman needs to visit a set of cities ( what is the shortest path to visit all cities once and get back to the departure city? Note that the distance between each city is known.

Pheromone update...

*Figure 4-1: GUI prototype*

### 4.1.2 MVP and Early prototype evaluations

The development schedule for this project being really tight meant the final user evaluation would occur quite late, not letting much time to make any corrections. To counterbalance this, I used a minimum viable product (MVP) approach for this project (Table 4-1) and launched a small user evaluation as soon as my prototype met the first MVP requirements.

*Table 4-1: Minimum Viable Products*

| Rank | MVPs | Requirements |
|---|---|---|
| 1 | A pedagogic demonstration software for solving TSP using AS. | GUI implementation, TSP problem definition, AS algorithm implementation, clear written explanation. |
| 2[2] | Any new ACO variant implementation. | ACO algorithm implementation, clear written explanation. |
| 3 | Any new problem-solving demonstration. | GUI update, problem definition, ACO algorithm implementation, clear written explanation. |

Four people, with no prior knowledge in ACO, took part in the first evaluation. Their feedbacks highlighted a severe lack of clarity in the explanations concerning AS and the influence of the user defined parameters. Additionally, the evaluation gave me some idea of which features the users wanted to see next such as edge detection demonstration, speed control, showing the evolution of

[2] The order for MVPs 2 and 3 does not really matter. Both can be developed and tested independently.

the shortest path as a graph, pre-filled parameters to launch a demonstration immediately, etc... It also appears they were not that interested in saving demonstrations at this point.

Taking their comments into account, I spent extra time refining the help panel containing the explanations before resuming the implementation of new features. The second released prototype featured corrected help panels, pre-filled parameters to launch demonstration immediately, an additional option to solve TSP with MMAS and a new demonstration of edge detection with ACS. I asked three teachers to evaluate this release to benefit from there experience regarding pedagogic material, but only my supervisor agreed to participate. His feedback confirmed me I was going in the good direction and suggested judicious idea for improvement including adding a lecture key for the graphs, scaling images to fill the demonstration space to be more visible during a projected lecture…

Based on these two evaluations, I revised some of my priorities. To spend more time refining the other demonstrations (TSP, edge detection and JSP), I decided to give up on Vehicle Routing Problem (VRP) demonstration. In fact, VRP being quite similar to TSP, it felt less interesting than the others from a pedagogic standpoint. This additional time allowed me to implement some of the features requested by the users (lecture key, speed control, scaling images, …). It also let me rewrite part of the code to smoothen the edge detection demonstration by reducing greatly computation time during pheromone updates.

## 4.2. Technology: Dart and flutter

### 4.2.1 Choice of technologies

The demonstration software was meant to be usable both as a website and a desktop application. For this reason, I chose to program this application in Dart, a language released in 2011 by Google and optimised for cross-platform development. Additionally, I adopted the development toolkit written in Dart and named Flutter to implement the user interface.

Dart and Flutter were new technologies for me. I spent the first nine days of the development phase to learn using them and to make sure they would be adapted to my project. The main point of worry was that flutter desktop support was still in development and thus not fully stable yet. In fact, Flutter related works seemed mainly targeted towards smartphone and web apps. These kinds of apps do not usually animate ten to hundreds of objects simultaneously as I would needs to visually demonstrate ant colony algorithms. My first test was to animate a bunch of elements randomly on screen as a desktop and web application. Both showed satisfying performances for up to a thousand objects, which reassured me regarding my choice of using Flutter and Dart for my project.

Other technologies such as the framework Electron, which focus on Mac, Linux and Windows cross-platform desktop application development, could have been another choice. However, I chose Dart for its resemblance to Java, a language I am more familiar with than the web technologies (javascript, html, css) on which relies Electron. Considering my previous tests and the tight schedule for my project, relying on these similarities to reduce the learning time I would need seemed an adequate solution.

## 4.3. Language description: Java and Widgets

**Dart**

As already stated, Dart is heavily inspired by Java programming language. However, some differences were included to make coding quicker. For instance, key words like protected, public or private were removed. Instead, you can make a variable private by starting its name with an underscore "_". Dart also introduces the use of facultative function parameters to replace Java usual overloads.

Another interesting point is that Dart support both imperative programming and functional programming. Functional programming enables among other to pass functions as arguments or assign them to variables, create nameless functions to use as constants or rewrite a function with multiple arguments in a series of single argument functions. For instance, you could rewrite a classical imperative "for" loop in the functional way "values.map(function).forEach(DoSomething)". This kind of programming can help with simpler syntaxes and code testability but its stateless logic can be hard to apply in some occasion such as for recursive functions for example.

One last big feature of Dart is the null safety. It allows to distinguish between non-nullable and nullable variables. A variable is non-nullable by default. Adding "?" behind the type during a variable declaration make it nullable. This safety helps to locate null error during edit time rather than at runtime. This helps reducing the need to check whether a value is null or not in code before using it (we already know it is not for non-nullable variables) and saves time on debug.

**Flutter**

Flutter toolkit main feature is probably its set of widgets which help to develop beautiful UI quickly and easily. Widgets are ready-to-be-used configurable components which allow you to define any aspect of your UI. Low level components (dropdowns, buttons or text input fields…) can be contained in higher ones (navigation bar, columns, rows, stack) to be arranged freely in the application window. This results in a tree like structure of parents and children which I find a lot clearer and easier to modify than Java's classical layouts. Furthermore, one of the highest-level widgets (MaterialApp) allows you to select a theme for your overall application to help maintaining a visual consistency. Of course, one remains free to overwrite this theme at lower level if needs be.

For more complex behaviours, Flutter allows to animate widgets with the help of the AnimatedWidget class. The basic process for simple animations relies and the Tween class which interpolate linearly between a start and end values to vary smoothly parameters such as an object position, colour, size… Additionally, the CustomPainter class allows to draw any form that can't be found in the default widgets set. Coupling this with animations gives Flutter extensive possibilities in term of application design. This is what allowed me to implement the visual demonstration for ACO involving hundreds of animated ants over custom graphs.

## 4.4. Cross platform

Dart and Flutter are optimised for cross-platform development. It supports Android, iOS, and Web platforms as well as desktop for Linux, MacOS and Windows. This project only focused on web and desktop platforms.

The code could be run as a Windows desktop app or on a web browser window (Chrome or Edge) with no modification for most part. Performance was a little weaker on the browser which seemed slower. This was especially noticeable for the edge detection demonstration which is the most expansive both memory and computationally wise. The application was however functional on both platform and the UI design identical. We can note however a small bug on the desktop app which fails to represent HTML lists' bullet points (insert an unwanted box character behind). Such issue I believe should be corrected in the future as Flutter desktop support and additional external libraries[3] are still in development and not fully stable yet.

One feature however is not compatible across platforms: the file picker. Directory management being so different from one operating system to another, there is currently no cross-platform compatible solution to manage it. Hence, two different file picker libraries were used to select images for the edge detection demonstration when running the Web or Windows desktop app. This relies on Dart conditional imports which detect whether the app is running on web or desktop at launch time. However, there is currently no way to know which OS we are running on when using the desktop app and the application could crash because of non-supported imports. As an alternative, an option to select an image from a sample included as part of the application was coded. Using this option instead, the demonstration software should theoretically be compatible with the three OS if the file picker feature is disabled.

## 4.5. Architecture (MVC inspired)

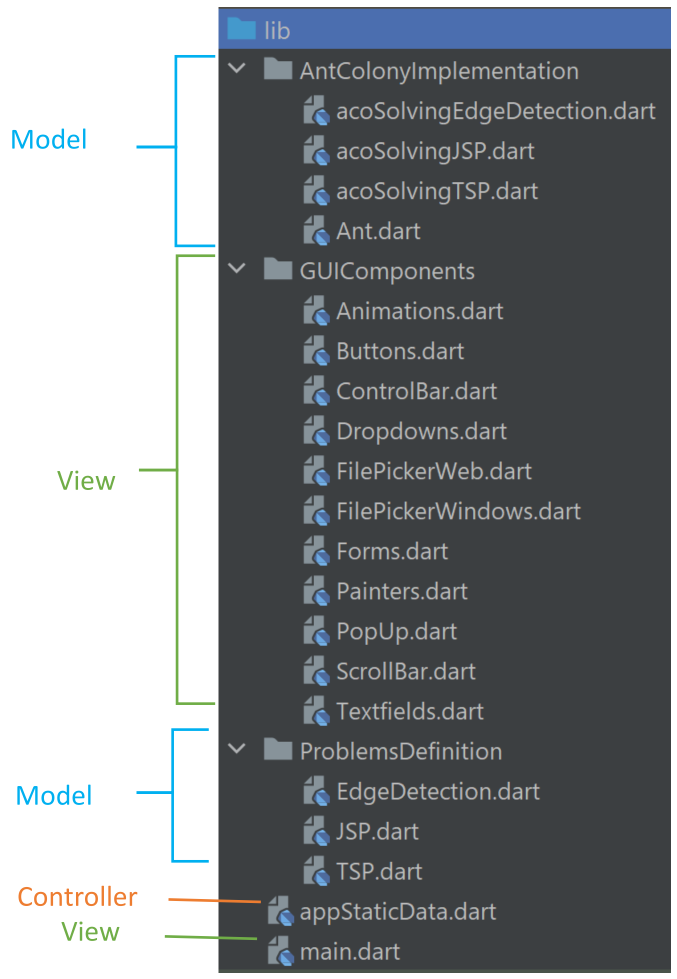
The code structure I decided for this project is inspired by the Model-View-Controller (MVC) software design pattern. Even though I did not use MVC terminology in my folders or files naming, I tried to use the same logic to clearly separate GUI related code from application logic related code.

In MVC, the **model** contains the data and logic of the application. In my code, this corresponds to the classes related to problems definition (TSP, JSP, edge detection) and ACO algorithms. The **view** corresponds to which is visible to the user: the graphical user interface. All widget related code in my program fits this category. I defined the main structure of the application in the "main.dart" file and created lower level graphical components in dedicated files stored in the "GUIComponents" folder. Lastly, the **controller** serves as a bridge between the view and the model. The equivalent for this in my project is contained in the file "appStaticData.dart" which holds global and static variables and methods that needs to be accessible by both the view and the model. The state of the application (selected algorithm and problem, animation status, etc…) is stored in the AppState class.

On user inputs (e.g. button click, form submission), the view side calls methods in the AppState class such as "generateProblem(…)" and "launchDemonstration(…)" which will serve as a router to call the adequate method from the model based on the state of the application (e.g. which problem to

[3] Help panels content was described in HTML files and interpreted thanks to flutter_html library.

generate? The current selected problem is TSP). On the other side, the model can call AppState "update" methods to request a graphic update on the view side. To prevent reloading the whole GUI for any small update, widgets are contained in "ValueListenableBuilders" that listen to a specified notifier variable. If a notifier value changes, all listeners that were watching it rebuild the widget they contain and their descendants. This allows to reload only targeted parts of the application widget tree and reduce computation time.  AppState "update" methods merely consist in changing a notifier value to trigger the rebuild.



```
lib
  AntColonyImplementation
    acoSolvingEdgeDetection.dart
    acoSolvingJSP.dart
    acoSolvingTSP.dart
    Ant.dart
  GUIComponents
    Animations.dart
    Buttons.dart
    ControlBar.dart
    Dropdowns.dart
    FilePickerWeb.dart
    FilePickerWindows.dart
    Forms.dart
    Painters.dart
    PopUp.dart
    ScrollBar.dart
    Textfields.dart
  ProblemsDefinition
    EdgeDetection.dart
    JSP.dart
    TSP.dart
  appStaticData.dart
  main.dart
```

*Figure 4-3: Project file tree*

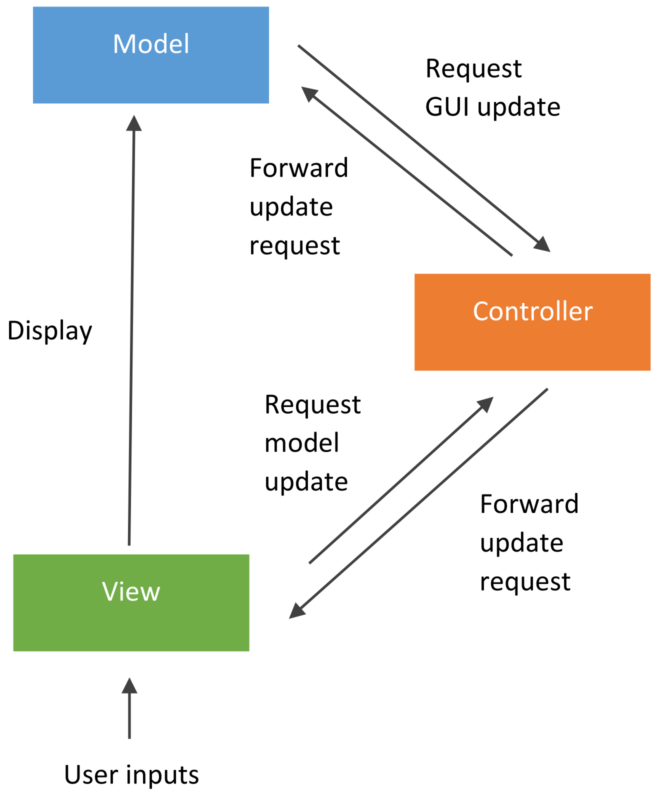


*Figure 4-2: MVC architecture*

ACO solving methods have been coded in their own separated file for each problem. This may lead to some repeated code but allows for a much easier maintenance and clear code separation. Problem solving demonstrations can be modified, suppressed or added independently, without heavy structural constraints, as if they were their own module.

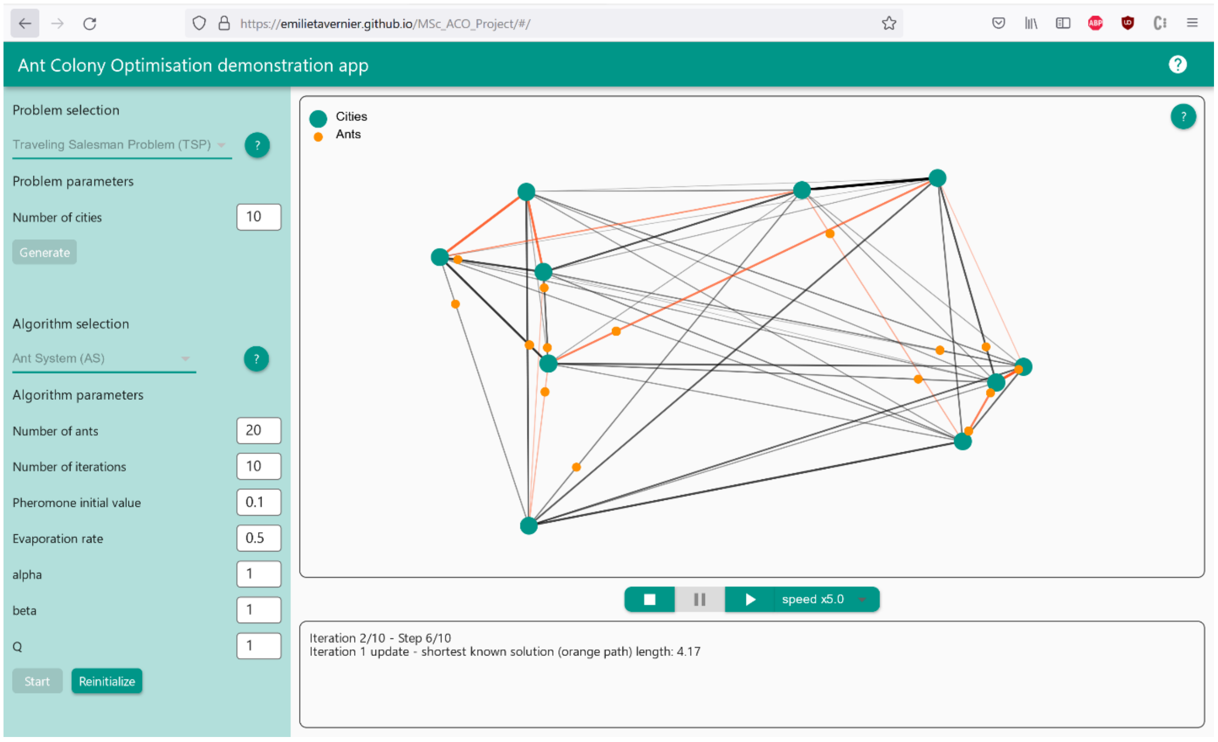
## 4.6. Functionalities

**Main features**

The ACO demonstration software was developed as a pedagogic tool to introduce students to ant colony optimisation and some of its usages. The application proposes a visual demonstration for ACO solving three different problems: TSP, JSP and edge detection. The user can select between three of the main ACO variants (AS, MMAS and ACS) for the TSP demonstration. However, only ACS was implemented for the other two problems because of time restrictions.

The three problem demonstrations show animated ants' movements and the evolution of pheromone concentration over time. This is meant to help the learner to visually understand how the algorithm works and how the ant's foraging metaphor is relevant.

Figure 4-4 shows the TSP visual demonstration. Ants move along the TSP graph edges. The edges' opacity and thickness are proportional to the pheromone concentration for the current iteration and the best path known is represented in orange.

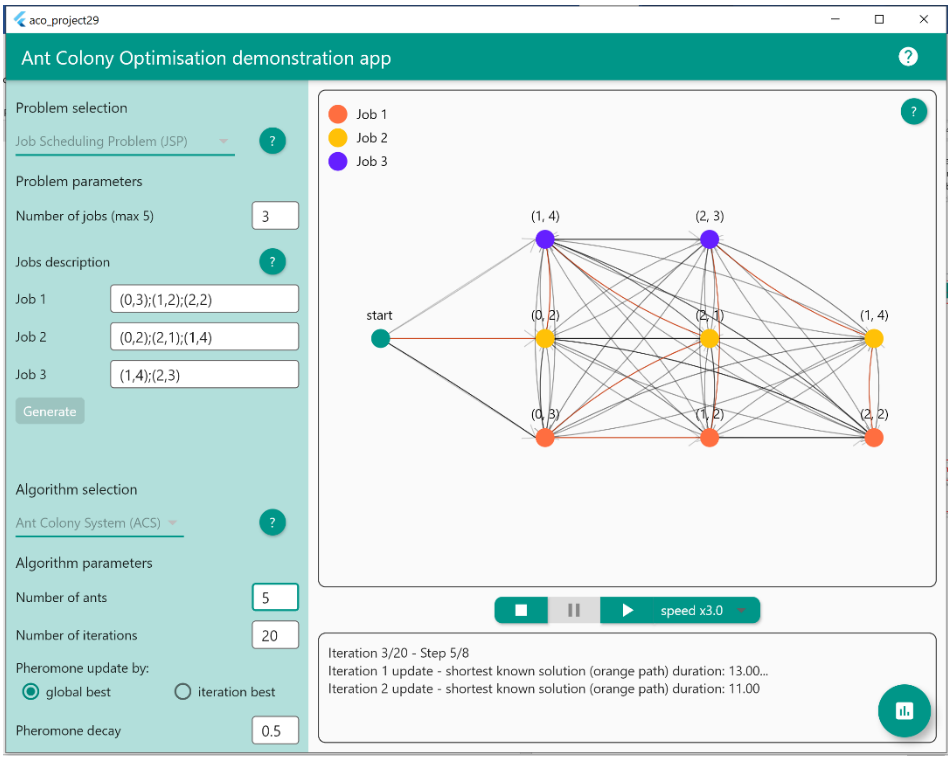

https://emilietavernier.github.io/MSc_ACO_Project/#/
Ant Colony Optimisation demonstration app
Cities
Ants
Generate
Start
Reinitialize
speed x5.0

Problem selection
Traveling Salesman Problem (TSP)
Problem parameters
Number of cities 10
Algorithm selection
Ant System (AS)
Algorithm parameters
Number of ants 20
Number of iterations 10
Pheromone initial value 0.1
Evaporation rate 0.5
alpha 1
beta 1
Q 1

Iteration 2/10 - Step 6/10
Iteration 1 update - shortest known solution (orange path) length: 4.17

*Figure 4-4: TSP demonstration*

JSP was the hardest to represent has several edges were superposed. Curved edges enabled to tackle this issue.


aco_project29

Ant Colony Optimisation demonstration app

Problem selection

Job Scheduling Problem (JSP)

Problem parameters

| | |
|---|---|
| Number of jobs (max 5) | 3 |

Jobs description

| | |
|---|---|
| Job 1 | (0,3);(1,2);(2,2) |
| Job 2 | (0,2);(2,1);(1,4) |
| Job 3 | (1,4);(2,3) |

Generate

Algorithm selection

Ant Colony System (ACS)

Algorithm parameters

| | |
|---|---|
| Number of ants | 5 |
| Number of iterations | 20 |

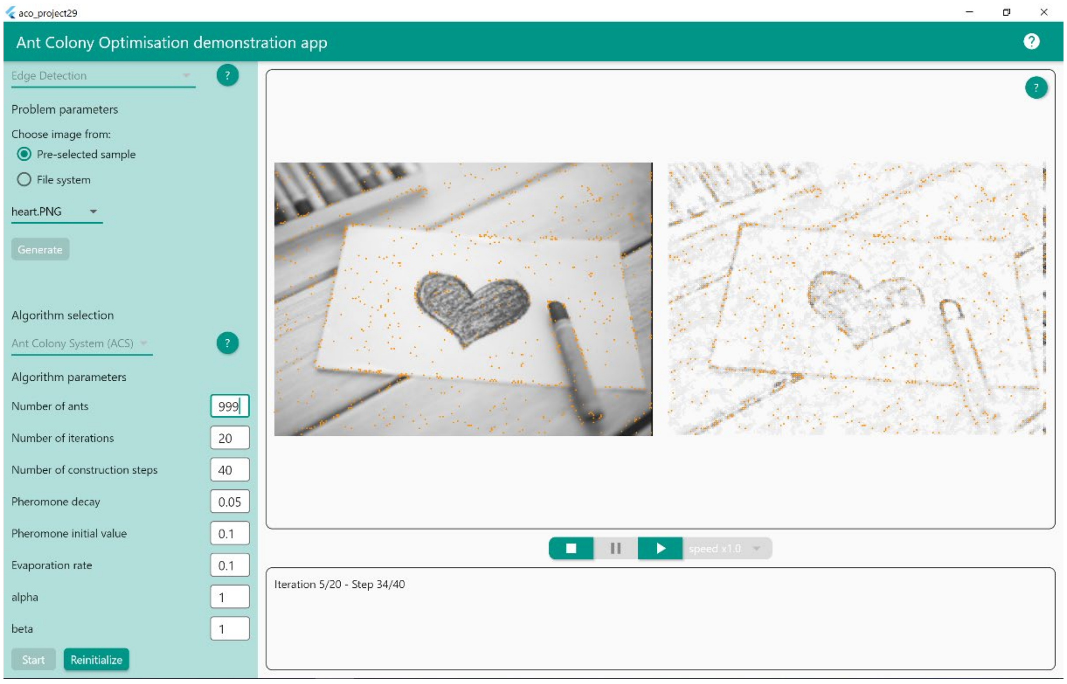
Pheromone update by: [✓ global best] iteration best

| | |
|---|---|
| Pheromone decay | 0.5 |

Iteration 3/20 - Step 5/8
Iteration 1 update - shortest known solution (orange path) duration: 13.00...
Iteration 2 update - shortest known solution (orange path) duration: 11.00

*Figure 4-5: JSP demonstration*

For edge detection, the original image is represented on the left. The image is grayscale and scaled to fit in half the demonstration space. On the right are represented the pheromones laid on the ants' paths. Ants are visible on both sides to understand their movements and the result.


aco_project29

Ant Colony Optimisation demonstration app

Edge Detection

Problem parameters

Choose image from:
[✓ Pre-selected sample]
File system

heart.PNG

Generate

Algorithm selection

Ant Colony System (ACS)

Algorithm parameters

| | |
|---|---|
| Number of ants | 999 |
| Number of iterations | 20 |
| Number of construction steps | 40 |
| Pheromone decay | 0.05 |
| Pheromone initial value | 0.1 |
| Evaporation rate | 0.1 |
| alpha | 1 |
| beta | 1 |

Start Reinitialize

Iteration 5/20 - Step 34/40

*Figure 4-6: Edge detection demonstration*

**Parameters' configuration**

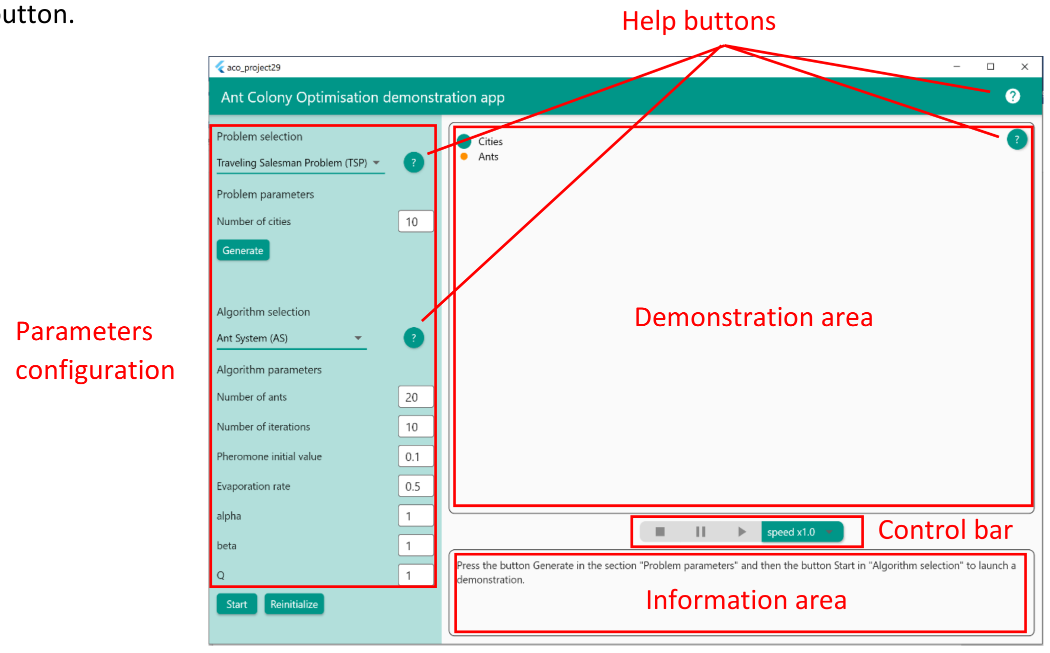
Most of the parameters related to the problem definition are configurable. The user can choose the number of cities for TSP but not their coordinates (randomly positioned[4]). For JSP, he can define the number of jobs and describe their composing tasks (order, duration and machine). Finally, the user can select an image for edge detection from their computer (file picker) or from a sample of images included directly in the application. The sample is composed of small images to maintain reasonable performances (limit application freeze) and guaranties good visual results.

All parameters concerning ACO are configurable (number of ant and iteration, pheromone initial value, evaporation rate…) as described in the literature. For MMAS and ACS, the user can choose between using the global-best or iteration-best ant for pheromone updates (except for edge detection where there is no best ant). All text fields receiving user inputs are secured thanks to regex and validators. Input value upper limits were decided in regards to limitation concerning three factors:

- **Performance**: the application start freezing if too many ants and graph's nodes are allowed,
- **Number representation**: number that are too big are store as "infinity" resulting in unexploitable computation results.
- **Visual representation**: there is a limit to the number of nodes and edges that can be represented clearly on screen.

The limit to the number of iteration (999) is arbitrary but I doubt any user would have the patience or need to wait for that many iterations. Most of the demonstrations would probably have settled to a solution long before that point anyway.

Lastly, parameter inputs are pre-filled so that beginner users can launch a demonstration as soon as they enter the application, even if they don't understand the parameters yet. The default values were chosen to show a demonstration that is both clear visually and can show good results reasonably quick. ACO parameters default values can be restored easily thanks to a "Reinitialized" button.



Problem selection: Traveling Salesman Problem (TSP)

Problem parameters

Number of cities: 10

Algorithm selection: Ant System (AS)

Algorithm parameters

Number of ants: 20

Number of iterations: 10

Pheromone initial value: 0.1

Evaporation rate: 0.5

alpha: 1

beta: 1

Q: 1

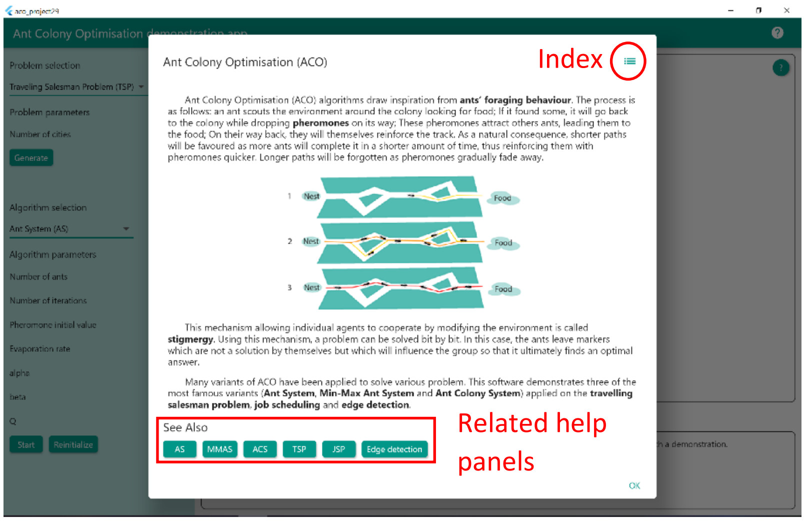
Press the button Generate in the section "Problem parameters" and then the button Start in "Algorithm selection" to launch a demonstration.

*Figure 4-7: GUI composition*

[4] Random positioning allows to launch a demonstration more quickly. This was also easier to program to have the first user evaluation as early as possible. Based on the user evaluations, it turns out participants did not think choosing coordinates was that important so this feature got removed from the priorities.

## Additional features

For the application to be pedagogic, multiple help buttons (Figure 4-7) allow to open explanation panels (Figure 4-9) focusing on every aspect of the demonstration (problems definition, graphs representation and algorithms description). Each panel ends with a list of buttons linking to other related help panels. A button in the top right corner allows to open an index from which all panels are accessible (Figure 4-8). With this, the user can choose whether to read specific parts of the help or to read it all in order to have a complete overview.



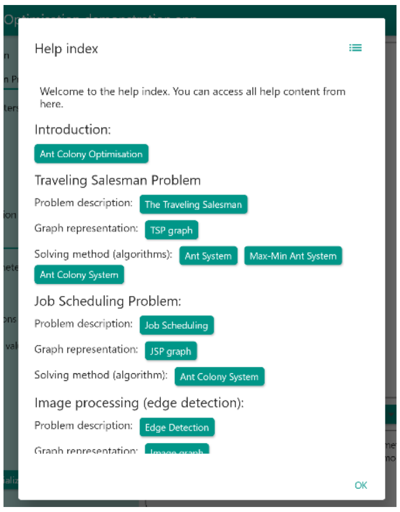
Ant Colony Optimisation (ACO)

Ant Colony Optimisation (ACO) algorithms draw inspiration from **ants' foraging behaviour**. The process is as follows: an ant scouts the environment around the colony looking for food; If it found some, it will go back to the colony while dropping **pheromones** on its way; These pheromones attract others ants, leading them to the food; On their way back, they will themselves reinforce the track. As a natural consequence, shorter paths will be favoured as more ants will complete it in a shorter amount of time, thus reinforcing them with pheromones quicker. Longer paths will be forgotten as pheromones gradually fade away.

This mechanism allowing individual agents to cooperate by modifying the environment is called **stigmergy**. Using this mechanism, a problem can be solved bit by bit. In this case, the ants leave markers which are not a solution by themselves but which will influence the group so that it ultimately finds an optimal answer.

Many variants of ACO have been applied to solve various problem. This software demonstrates three of the most famous variants (**Ant System**, **Min-Max Ant System** and **Ant Colony System**) applied on the **travelling salesman problem**, **job scheduling** and **edge detection**.

Figure 4-9: Help panel



Help index

Welcome to the help index. You can access all help content from here.

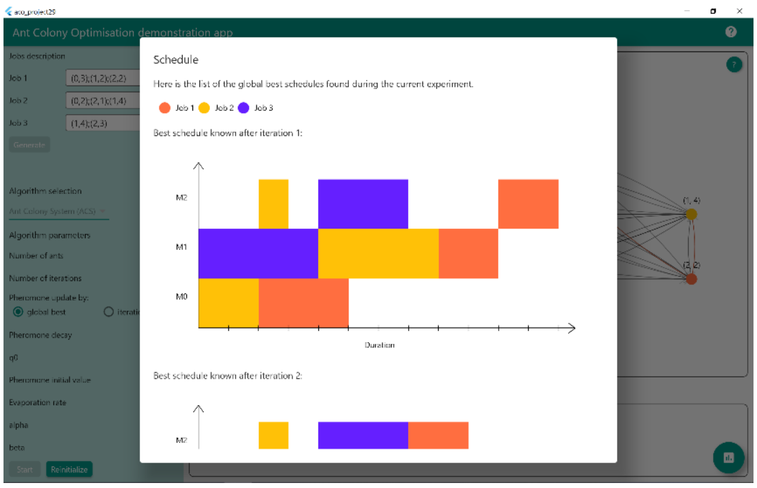
Introduction:
Ant Colony Optimisation
Traveling Salesman Problem
Problem description: The Traveling Salesman
Graph representation: TSP graph
Solving method (algorithms): Ant System
Max-Min Ant System
Ant Colony System
Job Scheduling Problem:
Problem description: Job Scheduling
Graph representation: JSP graph
Solving method (algorithm): Ant Colony System
Image processing (edge detection):
Problem description: Edge Detection
OK

Figure 4-8: Help index

A control bar (Figure 4-7) under the demonstration area allows to stop, pause and resume the demonstration as well as to control its speed. I think this feature would be especially useful for teachers, giving them full control to pace their explanation in front of students.

Under the control bar, there is an area to display various information (Figure 4-7) such as instructions to launch a new demonstration, counters to keep track of the demonstration advancement (iteration and step counters), and updates about the best scores known for TSP and JSP (shortest path or shortest schedule).

For JSP, a button appears at the lower right corner of the app. It enables to display a list of graphs representing the best schedules found during the demonstration (Figure 4-10).



Schedule

Here is the list of the global best schedules found during the current experiment.

● Job 1 ● Job 2 ● Job 3
Best schedule known after iteration 1:
M2
M1
M0
Duration
Best schedule known after iteration 2:
M2

Figure 4-10: JSP best schedules list

**Accessibility**

The application can be accessed directly online or you can download an executable file for desktop (windows only) at the following addresses:

- Online web app: https://emilietavernier.github.io/MSc_ACO_Project/#/
- Desktop version: https://github.com/EmilieTavernier/MSc_ACO_Project

For better performances, the desktop version is advised. The web app takes several seconds to display the interface at first, you may need to reload the page if this takes too long.

## 4.7. Open source and external libraries

The final product for this project is meant to be a free learning tool accessible for anybody that wish to know about ACO. Furthermore, I would be happy if teachers or researchers want to expand this tool for educational or demonstrating purpose by adding new implementations of ACO algorithms or problems to solve. To allow this, the software needs to be open-source. Dart, the programming language we selected for this project, uses a BSD license which imposes minimal restrictions on the use and distribution of the source code.

To facilitate the application maintenance and preserve its open-source status, I tried to limit the use of external libraries. Here is the list of external libraries I could not workaround because of time restrictions or feature complexity:

*Table 4-2: List of the external libraries used for the project*

| **Library** | **Usage** | **License** |
|---|---|---|
| Filepicker_windows | File picker for the windows desktop app | BSD |
| File_picker | File picker for the web app | MIT |
| Arrow_path | Library to draw arrows (for JSP graph representation) | BSD |
| Flutter_html | Library to display html content (help panels) | MIT |
| Image | Library to process images (grayscaling) | Apache 2.0 |

I made sure the covering licenses for all these libraries were open-source compliant and to include their copyright notice in my project folder as required for redistribution. The final source code for this project has been forked on a GitHub public repository so that anybody wishing to participate in eventual future development can do so.

# Chapter 5 Evaluation and results

## 5.1. Consent form and protection of personal data

The participation to the study was on a voluntary basis where participants were free to withdraw at any time with no consequence for them. Additionally, they were and remain free to ask for the suppression of the data that were collected from them or to access this data. I made sure they were aware of their rights via a consent form.

To easily identify a participant data if he wants to access them or ask for their suppression, I asked for a name or a pseudonym at the beginning of the survey. The pseudonym option was proposed in case participants did not wish to share their identity with me.

To be able to ask eventual clarifications to participants about their answers or if they wished to be informed about the final software release, I asked for a way to contact them. This field was completely optional. Of course, the contact information they eventually gave won't be use for anything they did not consent to.

The data included in the publication of results are completely anonyme to protect the participants privacy. You can find a copy of the consent form as Appendix A.

## 5.2. Evaluation design

The main objective of the evaluation was to see whether a user knowledge in ACO had increased after using the software. Additionally, another aim was to evaluate our interface usability and the relevance of the implemented features.

To proceed, an announce was sent to all MSC students at Heriot Watt University (HWU) and to the students of EFREI Paris, a French engineering school specialised in computer science. I also contacted some software engineers and some HWU teachers. This evaluation mainly, but not exclusively, targeted people in higher education, with some knowledge in computer science or algorithmic as they were potentially the ones most interested in learning ACO.

The instructions for the evaluation proposed a five steps process:

1. Complete the consent form;
2. Take a test on ACO to evaluate your current knowledge (15 questions);
3. Download and test the ACO demonstration program;
4. Retake the same test and see if there was any progress (15 questions);
5. Complete the user evaluation (20 mandatory questions – 7 optional).

Because the evaluation was quite long (about one hour), I specified that after completing steps 1 to 4, a participant had already provided me with valuable data to analyse and could choose to stop here. Participants who were already expert in ACO (e.g., teachers in biologically inspired computation) could skip steps 2 and 4.

A link for each step sent to a separate Microsoft form (see appendices p50-p58) so that participants could take pauses. No time limit was imposed for any step nor for the whole process.

The knowledge test was design as a multi-choice questionnaire. Asking the participants to take the same test twice would allow to evaluate precisely how much they improved on the quiz thanks to our application, even if they already add some basic knowledge on the matter. The user evaluation was composed of a System Usability Scale (SUS) survey and additional questions concerning the features relevance and quality. Figure 5-1 represents this whole evaluation process.

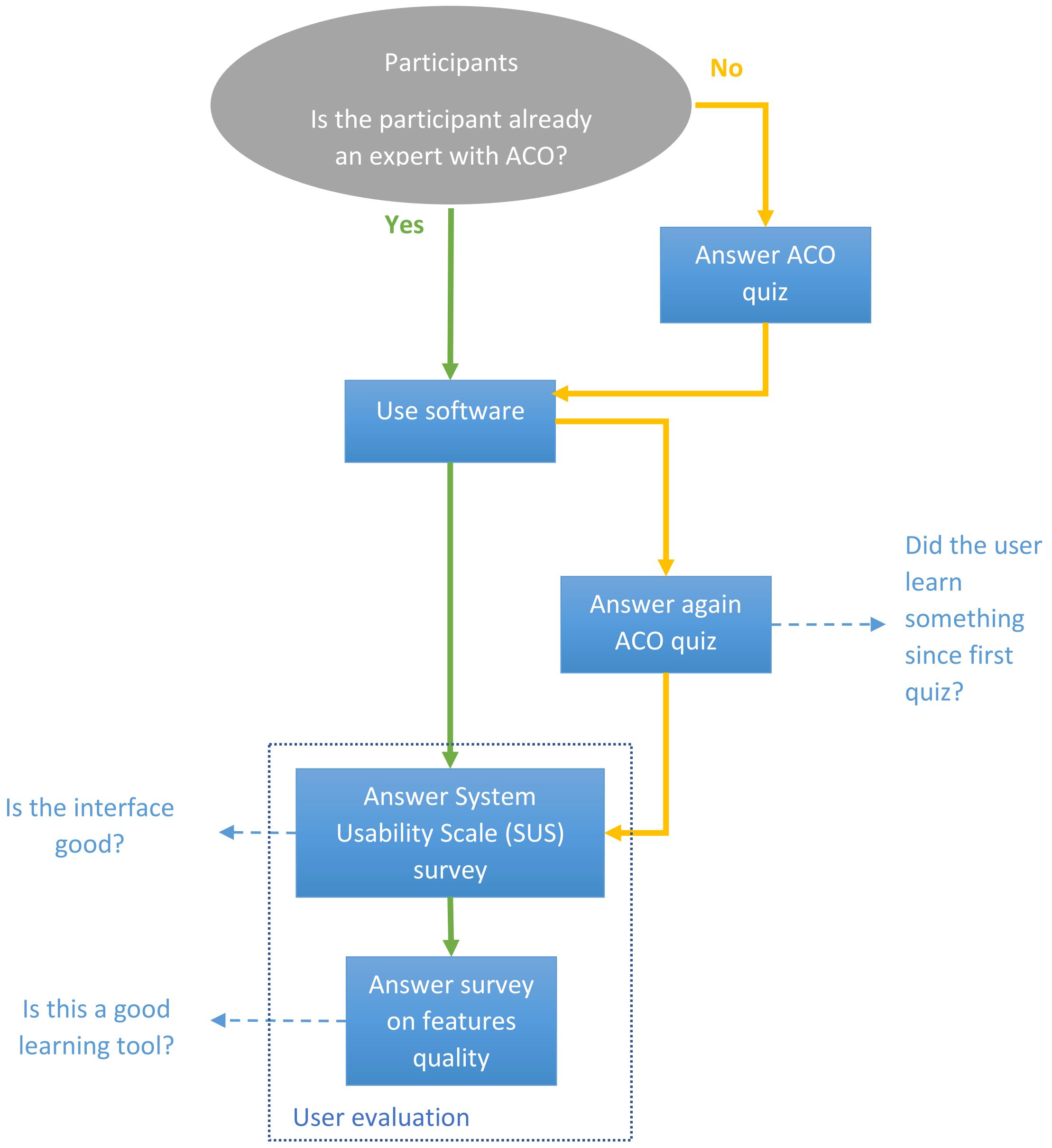

Participants
Is the participant already an expert with ACO?
No
Yes
Answer ACO quiz
Use software
Answer again ACO quiz
Did the user learn something since first quiz?
Is the interface good?
Answer System Usability Scale (SUS) survey
Is this a good learning tool?
Answer survey on features quality
User evaluation

*Figure 5-1: Evaluation plan*

The evaluation was launch on the 4th of august, letting only ten days for participants to volunteer and complete it. This was shorter than I would have wished but cutting more on the development schedule would not have allowed me to propose a satisfying version of JSP demonstration so I had to settle with this compromise.

## 5.3. Results

### 5.3.1 Knowledge tests

Seven volunteers took part in the knowledge tests (Appendix B). Participants were asked to fill the first test before downloading and trying the ACO demonstration software. They were authorized to use the software before and during the second attempt to look for answers. Looking for answers over the internet was not allowed for both tests. There was no time limit but the overall user evaluation was designed to last around one hour. There were 15 questions, each worth 1 point.

The results were pretty positive. As shown on Figure 5-2, all of the participants but one managed to increase their score on the second attempt and none scored worse. On average, they improved by 6.9 points on their second try.

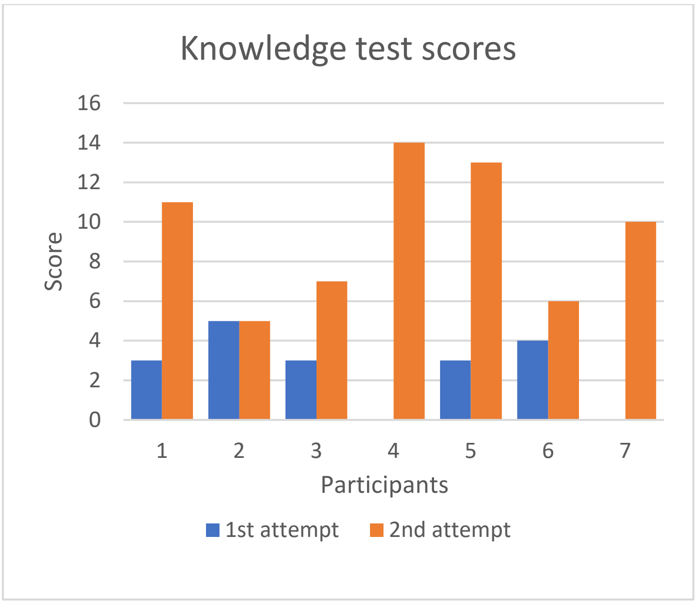


*Figure 5-2: Knowledge test scores*

### 5.3.2 User evaluation

Eight volunteers took part in the user evaluation (Appendix C) which was composed of a System Usability Scale (SUS) survey and custom questions about the application content (features quality and relevance regarding ACO). This section describes and analyses the results obtain on each part.

**System usability scale (SUS)**

Figure 5-3 shows the results on the SUS survey. Given the nature of the application, the first question "I think that I would like to use this system frequently" was only really relevant for teachers in algorithmy or biology inspired computation. This was confirmed by some commentaries from the participants and explains the mixed result on this question.

Two participants also commented on this section that with no background in mathematic nor computer science, the system could be a little overwhelming and the documentation challenging. This remarks are fully justified but may be more targeted toward the application content (ACO related), which I tried to evaluate in the second section,  rather than the purely objective system usability.

Overall, the results of the SUS survey were sill quite positive. This validates the user interface design and implementation.

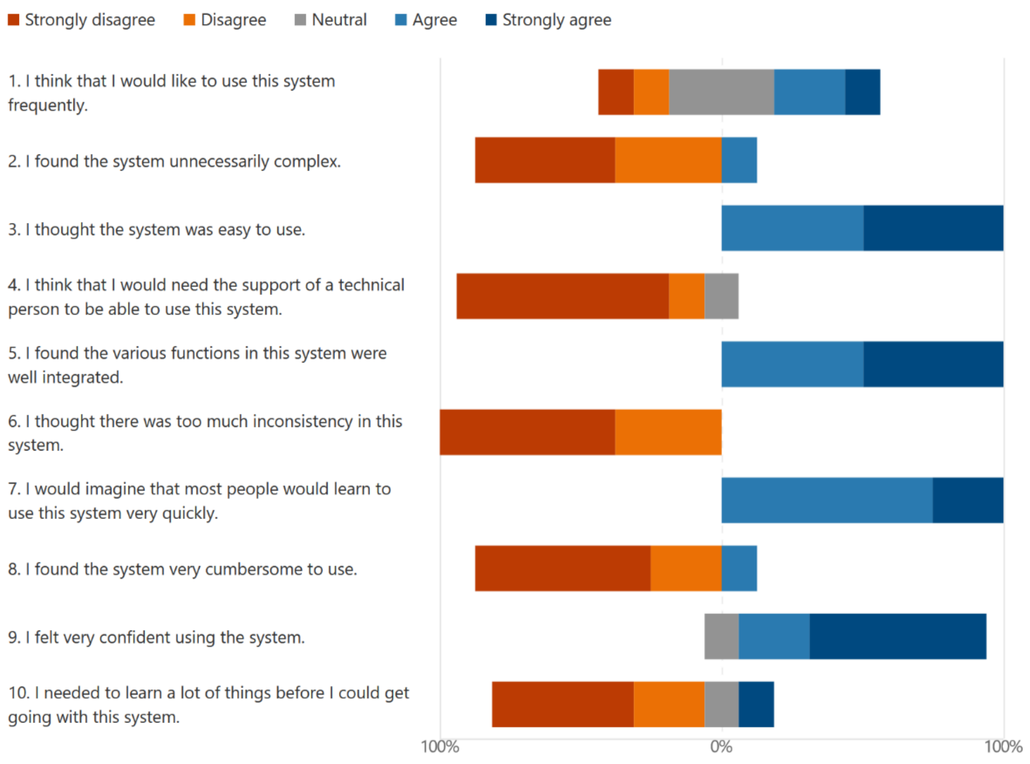


*Figure 5-3: System Usability Scale results*

**Content quality**

Figure 5-4 shows results for the second part of the user evaluation. Overall, the participants thought that the different algorithms and problems were well explained. They thought that the software would be a good learning tool with and without a teacher so our main goal was reached according to their opinion.

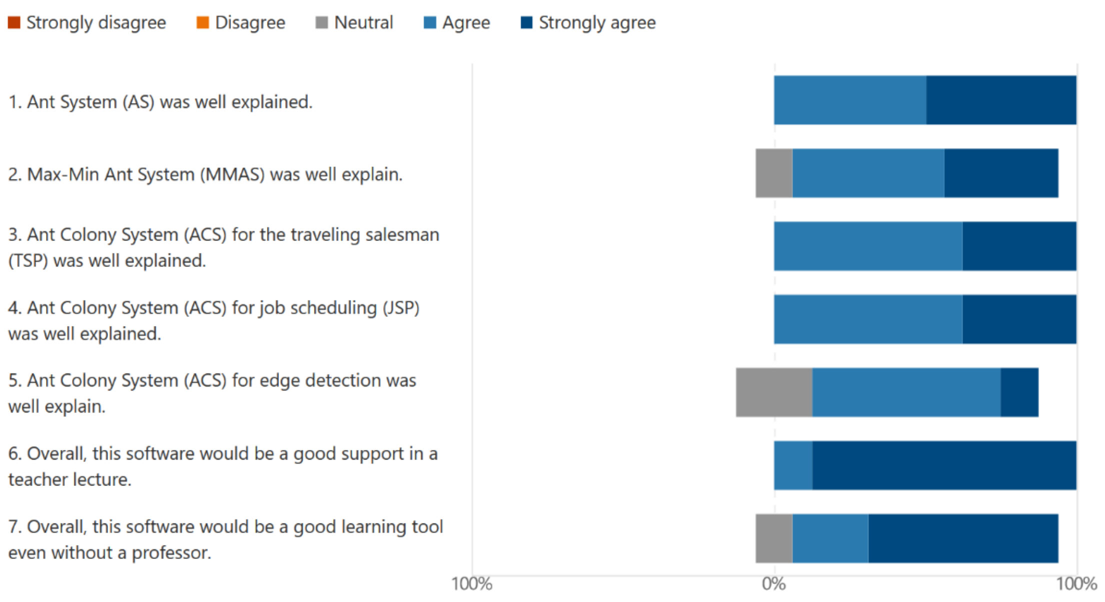


*Figure 5-4: Content evaluation result*

**Additional feedback**

The participants were asked to rank potential future features in term of priority (Table 5-1, see Appendix D for detailed ranking). In general, enhancing already implemented demonstration appeared more important to them than adding a new one. The top three features are the easiest to implement and would add the most value in the eye of the user. This gives some clear objectives for the near future.

*Table 5-1: Potential future features ranking*

| Rank | Feature |
|---|---|
| 1 | Allow tracking an ant movement thanks to a different color |
| 2 | Hide/show best path |
| 3 | Allow disabling ants' animation for longer demonstration (show only evolution of pheromone concentration) |
| 4 | Allow forward execution step by step |
| 5 | Generate graph to display shortest solution length evolution for TSP |
| 6 | Allow backward execution step by step |
| 7 | Display pheromone concentration value of an edge when hovering it |
| 8 | Allowing the user to freely position TSP cities on the map (as an alternative to the current random positioning) |
| 9 | Saving a demonstration to replay later |
| 10 | Adding a demonstration for Vehicle Routing problem |

Additionally, four participants commented about the difficulty to see the best path and to distinguish slight variation in pheromone concentration. As a solution, they proposed to play on edges thickness property in addition of the opacity. Two other comments suggested adding more speed control options (up to x10 multiplier). These remarks have been taken into account and implemented in the most recent version of the project. Currently however, the best path remains purposely hard to see if it is low in pheromone concentration. A button to hide/show the best path with constant thickness and opacity could allow to tackle this issue.

# Chapter 6 Conclusion and future work

The goal of this project was to develop a pedagogical demonstration software showcasing ACO main variants and various possible usages. The results of the knowledge test and of the user evaluation both show the pedagogical objective was reached.

Regarding the MoSCoW requirements that were defined at the beginning of the project, all "must" have been successfully implemented.

To polish other features, the vehicle routing demonstration and the "fully" editable problem space requirements were given up. Considering the user ranking of potential future features (Table 5-1), this appeared to be an adequate decision. Furthermore, it allowed user requested features including speed control, pre-filled reinitialisable parameter fields and a help index, to be implemented instead.

Table 6-1 recaps MoSCoW requirements from section 3.4 and their achievement status.

*Table 6-1: Requirements achievement assessment*

| **Requirements** | **Old MoSCoW** | **Status** |
|---|---|---|
| **ACO implementation** | | |
| Ant System | Must | Done |
| Max-min Ant System | Must | Done |
| Ant Colony System | Must | Done |
| ~~Other ACO variants~~ | ~~Won't~~ | |
| Configurable parameters (number of ants, iterations…) | Must | Done |
| **Problem solving demonstrations** | | |
| TSP | Must | Done |
| Job scheduling | Should | Done |
| Vehicle routing | Should | Not done |
| Edge detection | Must | Done |
| ~~Other problems (edge linking, ant foraging…)~~ | ~~Won't~~ | |
| **Accessibility** | | |
| Web-based | Must | Done |
| Executable | Should | Done |
| **Graphical user interface** | | |
| Clear Explanations panel | Must | Done |
| "Real-time" visualisation of ACO execution | Must | Done |
| ~~ACO parallel execution with different parameters~~ | ~~Won't~~ | |
| **Additional features** | | |
| ~~Saving experiment parameters and/or execution~~ | ~~Won't~~ | |
| Randomize problem space design | Must | Done |
| "Fully" editable problem space design (e.g., choosing TSP cities' position) | Could | Not done |
| **Not planned** | | |
| Control bar including pause, stop, play, and speed control | | Added |
| Pre-filled reinitializable parameters | | Added |
| Fully navigable help content (index and "see also") | | Added |

To conclude, all major requirements have been fulfilled and validated by the user evaluation and knowledge tests results. The main objective to create an educational ACO demonstration software was successfully reach. The resulting product offers the pedagogical content, accessibility (web and desktop), and a larger coverage of ACO that were lacking in pre-existing demonstration tools discussed in section 2.6.

There is still some room for improvements however. The first objective in the near future will be to implement the top three ranking features chosen by the user evaluation participants (Table 5-1) and maybe features ranked 5th and 8th. In fact, these features would result in easily added value without involving too much work.

After this, it would be interesting to study optimisation possibilities to increase performances on the web, mostly for the edge detection demonstration which is quite slow. In the meantime, I would like to build desktop version for Mac and Linux to complete the Windows one and serve as an alternative for better performances than the web app.

Given the user evaluation results and my overall satisfaction with the current state of the application, I do not think a VRP demonstration will be implemented. Step by step forward and backward execution would probably be more interesting. In addition to be useful in a teacher lecture, they would require the implementation of a buffer to store ACO computations, which would allow to save demonstrations easily and may help solving the web performance issues.

Ultimately, the code for this application is fully open-source and can be accessed on my GitHub page. Anyone is free to copy it and develop new versions. To this purpose, the code has been thoroughly documented to offer others the keys to expand on it.

# Appendix A: Consent Form

This study aims to evaluate the Ant Colonisation Optimisation (ACO) demonstration software prototype I developed for my MSc project at Heriot Watt University (MSc Artificial Intelligence). The evaluation is designed in 4 steps:

1. take an ACO knowledge test;
2. Download and test the ACO demonstration software;
3. Re-take the ACO knowledge test;
4. Answer the user evaluation survey.

Participation in the study is completely voluntary and you can withdraw at any time.

You can also ask me to send you back or suppress the data I collected about you (surveys' answers) at any time. The data collected in this study will only be used for research purposes for a MSc dissertation project at Heriot-Watt University.

I will only use your name or pseudonym to:

- identify the data I need to suppress or to send you in case you ask for it at emt2000@hw.ac.uk.
- contact you for eventual clarification on your answers if you consent to it (optional).

You should use a pseudonym that do not relate to you in anyway if you don't wish to share your identity with me.

Results use in the study will be completely anonymised.

If you agree to participate, you will be ask to test our software. To use it, you will need to download and run an executable file (.exe) on your own device. Please note that neither the HWU nor the investigator accept any responsibility for any loss/damage incurred by the use of the software on your device.

By completing the fields bellow, you are giving voluntary consent that you have read all the information above, and fully agree to take part in this study:

Name or pseudonym

2. Date of consent

3.Do you agree to be eventually contacted for clarification concerning your answers?
Of course, you remain free to change your answer anytime in the future.

○ Yes ○ No

4.If you answered yes to question 3, where should I contact you? (optional)
*(e.g., email address, phone number, ...)*
*I will not use your contact for anything you did not consent to.*

## Appendix B: Knowledge test

Introduction

If you are here, I assume you first completed the consent form I sent you. If not, please turn back.

This test aims to evaluate your knowledge concerning Ant Colony Optimisation prior to using the demonstration software. You will be asked to retake the same test after using the software. There is absolutely no shame in making mistakes now nor during the second test. I am evaluating the software's potential as a learning tool, not you.

There is no time limit.

Please, for the validity of this study:

- DO NOT look up for answers on the internet.
- Complete this test BEFORE using the demonstration software.

1.Name or pseudonym

Please use the same as in the consent form

2.Ant System pheromone update happens...

Select best answer.

○ At each construction step

○ At the end of an iteration

○ Both

○ No idea

3.When using ACO to solve TSP, the probability for an ant to select a city as next destination is:

select ALL true statements

☐ Proportional to the distance

☐ Proportional to the pheromones concentration

☐ Null if the city was already visited by this ant

☐ No idea

4.Compared to AS, MMAS pheromone update...

Select ALL true statements

- ☐ is performed by less ants
- ☐ happens at a different time
- ☐ No idea

5.ACS is different of AS...

Select ALL true statements

- ☐ On next destination choice policy
- ☐ On pheromone update policy
- ☐ No idea

6.AS pheromone updates are performed...

Select best answer

- ○ By all ants
- ○ Only by best ants
- ○ No idea

7.ACS pheromone updates are performed...

Select best answer

- ○ At each construction step
- ○ At the end of an iteration
- ○ Both
- ○ No idea

8.MMAS pheromone updates are performed...

Select best answer

- ○ At each construction step
- ○ At the end of an iteration
- ○ Both
- [✓ No idea]

9.The graph representation for TSP is:

Select best answer

- A fully connected graph
- A partially connected graph
- No idea

10.In JSP, an iteration is over after...

Select best answer

- A pre-defined number of nodes was visited
- All nodes were visited
- No idea

11.In JSP graph representation, a node corresponds to...

Select best answer

- A task
- A machine
- A job
- No idea

12.When solving JSP using ACO...

Select best answer

- It is possible that different paths leads to the same schedule
- Each path lead to a unique schedule
- No idea

13.In JSP, the makespan is...

Select best answer

- The sum of all jobs' duration
- The duration to complete all jobs
- The minimum duration of the longest job
- No idea

14.For edge detection, an iteration is finished after...

Select best answer

- ○ A pre-defined number of nodes was visited
- ○ All node were visited
- ○ No idea

15.For edge detection, the graph representation of  an image is

Select best answer

- ○ A fully connected graph
- ○ A partially connected graph
- ○ No idea

16.When using ACO, ants are starting on...

Select ALL true statements

- ☐ Random nodes for TSP
- ☐ A common departure node for TSP
- ☐ Random nodes for JSP
- ☐ A common departure node for JSP
- ☐ Random nodes for edge detection
- ☐ A common departure node for edge detection
- ☐ No idea

# Appendix C: User evaluation

Introduction

If you are here, I assume you already completed the consent form I sent you AND tested my software. If not, please turn back.

This survey aims to evaluate the Ant Colonisation Optimisation (ACO) demonstration software prototype. The evaluation will focus on two aspects:

1. the system usability (11 questions);
2. the features relevance and quality (16 questions).

1.Name or pseudonym

Please use the same as in the consent form

## System Usability Scale

This part focus solely on the software usability. It aims to evaluate the software on three aspects:

1. effectiveness - can the user achieve is goal?
2. efficiency - can the goal be achieved easily?
3. satisfaction - was the experience pleasant?

2.Do you agree or disagree with these affirmations?

| | Strongly disagree | Disagree | Neutral | Agree | Strongly agree |
|---|---|---|---|---|---|
| 1. I think that I would like to use this system frequently. | ○ | ○ | ○ | ○ | ○ |
| 2. I found the system unnecessarily complex. | ○ | ○ | ○ | ○ | ○ |
| 3. I thought the system was easy to use. | ○ | ○ | ○ | ○ | ○ |
| 4. I think that I would need the support of a technical person to be able to use this system. | ○ | ○ | ○ | ○ | ○ |
| 5. I found the various functions in this system were well integrated. | ○ | ○ | ○ | ○ | ○ |
| 6. I thought there was too much inconsistency in this system. | ○ | ○ | ○ | ○ | ○ |
| 7. I would imagine that most people would learn to use this system very quickly. | ○ | ○ | ○ | ○ | ○ |
| 8. I found the system very cumbersome to use. | ○ | ○ | ○ | ○ | ○ |

| | Strongly disagree | Disagree | Neutral | Agree | Strongly agree |
|---|---|---|---|---|---|
| 9. I felt very confident using the system. | ○ | ○ | ○ | ○ | ○ |
| 10. I needed to learn a lot of things before I could get going with this system. | ○ | ○ | ○ | ○ | ○ |

3.Any comments concerning these grades?

## Features evaluation

This section concerns existing and future features relevance.

4.Before using this software, were you familiar with Ant Colony Optimisation (ACO)?

○ Yes

○ No

5.We will keep in mind your previous answer to analyze the following results. Please grade the following affirmations:

| | Strongly disagree | Disagree | Neutral | Agree | Strongly agree |
|---|---|---|---|---|---|
| 1. Ant System (AS) was well explained. | ○ | ○ | ○ | ○ | ○ |
| 2. Max-Min Ant System (MMAS) was well explain. | ○ | ○ | ○ | ○ | ○ |
| 3. Ant Colony System (ACS) for the traveling salesman (TSP) was well explained. | ○ | ○ | ○ | ○ | ○ |
| 4. Ant Colony System (ACS) for job scheduling (JSP) was well explained. | ○ | ○ | ○ | ○ | ○ |
| 5. Ant Colony System (ACS) for edge detection was well explain. | ○ | ○ | ○ | ○ | ○ |
| 6. Overall, this software would be a good support in a teacher lecture. | ○ | ○ | ○ | ○ | ○ |
| 7. Overall, this software would be a good learning tool even without a professor. | ○ | ○ | ○ | ○ | ○ |

6.Any additional comments concerning question 6?

7.Any comments concerning the Traveling Salesman demonstration?

8.Any comments concerning the Job Scheduling demonstration?

9.Any comments concerning the Edge Detection demonstration?

10.The current software release contains all the major features I wanted to implement. If time allows however, I may consider implementing additional features from the following list. Please class these features in term of priority.

11.Any additional features you would like to see?

12.Is there big flaws you would like to report?

Report a bug, missing important features...

13.Anything else I forgot to cover in this survey?

You did it!

Thank you very much for your time. If you have any question or want to discuss about this project, feel free to email me at emt2000@hw.ac.uk.

What comes next?

The deadline for this MSc project will be mid august. Until then, I will mainly focus on writing my MSc dissertation and correct eventual issues highlighted by this user evaluation.

After that, I will upload an executable file for the desktop application and all source code on a public GitHub repository as I wish this project to be fully open-source. The desktop support for the development kit I used (flutter) is not stable yet so I may also consider making the app available online depending on hosting conditions.

14.Do you wish to be contacted when the final release is published and receive a link to access it? Of course, you remain free to change your answer anytime in the future.

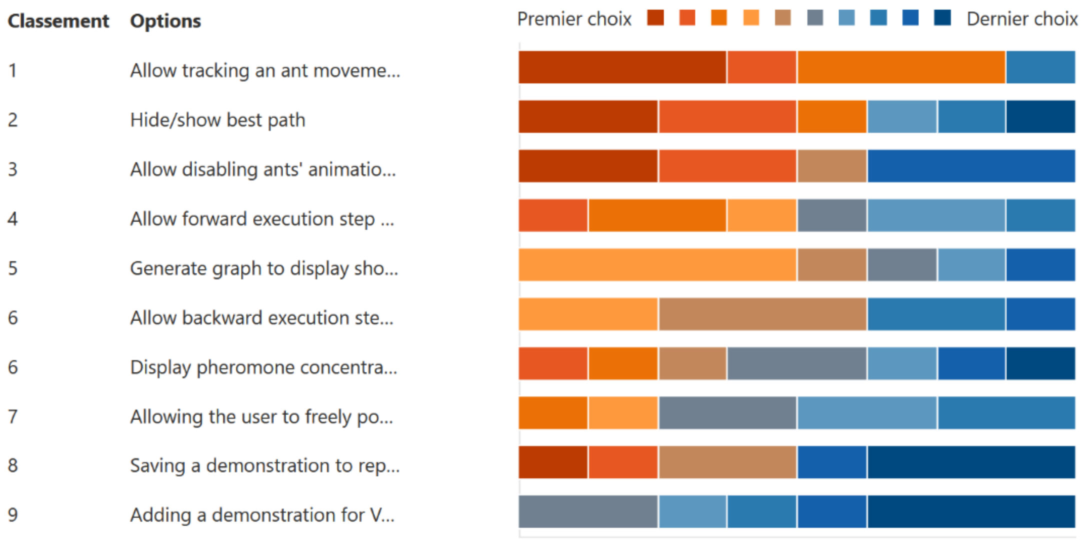
○ No

○ Yes

15.If you answered yes to question 15, where should I contact you? (optional)

(e.g. email address, phone number, ...)
I will not use your contact for anything you did not consent to.

## Appendix D: Potential future features detailed ranking



| Classement | Options |
|---|---|
| 1 | Allow tracking an ant moveme... |
| 2 | Hide/show best path |
| 3 | Allow disabling ants' animatio... |
| 4 | Allow forward execution step ... |
| 5 | Generate graph to display sho... |
| 6 | Allow backward execution ste... |
| 6 | Display pheromone concentra... |
| 7 | Allowing the user to freely po... |
| 8 | Saving a demonstration to rep... |
| 9 | Adding a demonstration for V... |

# Bibliography

[1] M. A. Lones, “Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms,” *SN Computer Science,* Mars 2020.

[2] T. Stutzle, M. Dorigo and M. Birattari, “Ant colony optimization artificial ants as a computational intelligence technique,” *IEEE Computational Intelligence Magazine,* November 2006.

[3] A. Colorni, M. Dorigo and V. Maniezzo, “The ant system: Optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* February 1996.

[4] A. Puris, R. Bello, Y. Trujillo, A. Nowe and Y. Martínez, “Two-Stage ACO to Solve the Job Shop”.

[5] E. Florez, L. Bautista, W. Ariel and G. Bueno, “An ant colony optimization algorithm for job shop scheduling problem,” September 2013.

[6] S. v. d. Zwaan and C. Marques, “Ant Colony Optimisation for Job Shop Scheduling,” 1998.

[7] Google, “An online course about job-scheduling problem,” [Online]. Available: https://acrogenesis.com/or-tools/documentation/user_manual/manual/ls/jobshop_def_data.html.

[8] C. Oppus and A. V. Baterina, “Image edge detection using ant colony optimization,” *WSEAS transaction on Signal Processing,* April 2010.

[9] A. Rezaee, “Extracting edge of images with ant colony,” *Journal of Electrical Engineering,* 2008.

[10] W. Y. J. Tian and S. Xie, “An ant colony optimization algorithm for image edge detection,” *IEEE Congress on Evolutionary Computation,* 2008.

[11] D. Andina, A. Jevtić and I. Melgar, “Ant based edge linking algorithm,” *Industrial Electronics, 35th Annual Conference of IEEE,* December 2009.

[12] G. B. Dantzig and J. H. Ramse, “Tye truck dispatching problem,” *Management Science, Vol 6, No. 1,* pp. 80-91, October 1959.

[13] I. V. Nieuwenhuyse, K. Braekers and K. Ramaekersn, “The vehicle routing problem: State of the art classification and review,” *Computers Industrial Engineering, vol. 99,* pp. 300-313, December 2015.

[14] P. R. McMullen and J. E. Bell, “Ant colony optimization techniques for the vehicle routing problem,” *Advanced Engineering Informatics, vol. 18,* pp. 41-48, November 2004.

[15] T. Pool, “Visualisation of ant colony optimisation,” 2015. [Online]. Available: https://courses.cs.ut.ee/demos/visual-aco/#/visualisation.

[16] A. Belezjaks, “Ant colony simulation,” Mars 2014. [Online]. Available: https://code.google.com/archive/p/ant-colony-simulation/.

[17] E. Weitz, “Ant colony optimization,” [Online]. Available: http://weitz.de/aco/.

[18] I. Panasiuk, “Ant colony optimization on vehicle routing problem demonstrations,” February 2014. [Online]. Available: https://www.youtube.com/watch?v=21g0RxiKtAA.

[19] TSealsRobotics, “Image edge detection inspired by ant colony optimisation systems,” May 2013. [Online]. Available: https://www.youtube.com/watch?v=qaiyydNp4TU.