Lab 1: Trail Laying/Following & Ant Colony Optimization

This laboratory requires the following equipment:

- C/C++ programming tools
- Matlab

The laboratory duration might require an effort of up to four hours although assistance will be provided during a time window of three hours. The laboratory is not graded; however, we encourage you to take notes during the course of this laboratory to help preparing for the exams.

A solution sketch will be posted after the lab session.

Office hours

Additional assistance outside the lab period (office hours) can be requested on appointment using the dis-ta@groupes.epfl.ch mailing list.

Information

In the following text you will find several exercises and questions. The type of question is indicated between parentheses:

- The notation **S** means that the question can be solved using only additional simulation.
- The notation **Q** means that the question can be answered theoretically, without any simulation.
- The notation **I** means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal (the bonus questions can involve a maximal effort of 20 points)

To prepare you for the exams and to permit you better time planning during the exercise session, we also show an indicative number of points for each exercise between the very same parentheses. The combined total number of points for each laboratory exercise is 100, without bonus questions.

1 Self-organization

This lab is intended to be a simple introduction to the topic of self-organized systems. Recall that the key ingredients observed in natural self-organization are:

- Positive feedback (e.g., recruitment mechanisms)
- Negative feedback (e.g., exhaustion, saturation, competition)
- Randomness (e.g., initial fluctuations)
- Multiple interactions (e.g., interactions with the environment and other individuals)

In this lab, you will conduct a set of simulations to investigate several models of foraging strategies and an ACO algorithm, the AS discussed in class. These simulations are meant to illustrate the use of swarms in creating practical solutions to difficult problems. While we will not examine particularly

complex systems, the generalization to these problems is straightforward, and there will be opportunities for inquisitive students to investigate further.

2 Working Environment:

In this laboratory session, you will need to have MATLAB and a C compiler installed on your computer. We recommend that you use Ubuntu Linux as operating system if you have access to it on your machine. All the commands and instructions given in this lab are meant for Ubuntu. However, it is possible to use a different operating system if you wish so, since both MATLAB and gcc work on the most popular operating systems (e.g., Windows, Mac) as well. In this case, please make sure you run the commands adapted to your operating system instead of the ones mentioned in the assignment.

If you do not have MATLAB installed, or if you have a version older than R2018a, please follow the instructions that you can find on the following link to download and install it. https://www.epfl.ch/campus/services/en/it-services/it-support/pro-softwares/

Moreover, if you need help installing gcc on your Windows, Mac, or Ubuntu machines, you can use the series of instructions that we have prepared and uploaded on Moodle.

3 Lab Part 1: Antsim

During this lab you will become a virtual ant expert. You will be given the following four species of ants to examine, and you are to explain their performance in different environments.

1) **Justant**: The basic ant. It lays pheromone wherever it travels until its supply runs out. The pheromone evaporates rather quickly. Justant prefers to move toward regions of higher pheromone concentration, but it also has a tendency to wander randomly.
2) **Sneezy**: Similar to the Justant, except it suffers from an inability to perceive any pheromone trails.
3) **Greedy**: Similar to the Justant, except it must always move toward an area of higher pheromone concentration.
4) **Diffrant**: Most different from Justant. It lays pheromone much more slowly, and its pheromone evaporates more slowly. Also, it has a lower tendency to wander and a smaller pheromone supply.

Antsim simulates the behavior of each species on different environment maps. In each map there is a nest, from which the ants emerge slowly throughout the simulation. The goal of each ant is to find a source of food and return a bit of it to the nest. Antsim will return the average amount of food returned over multiple simulation runs, for a given environment-species pair.

3.1 Getting started with Antsim

First, download and extract the file `lab01.zip` provided on Moodle. Use the simulator by changing into the part1 directory and entering:

```
cd lab01/part1
make

./antsim [environment] [species] [runs] > simul.m
```

- *environment* = {1,2,3,4} specifies the map in which the ants will roam.

- *species* = {1,2,3,4} specifies the species to simulate.
- *runs* specifies the number of simulation runs to perform. Use 1 to get a feeling for what is happening qualitatively, and then increase to 10 or 20 to get a better estimate of food return rate.

Go through antsim.c to get a feeling about how the simulator and the parameters defining each species work. This will make it easier to answer the questions below.

Various different environments are defined in the .map files. You can display these environments in matlab by typing display_env(<environment file name>). They are text files containing a matrix of numbers which representing different elements (e.g, 2=open space, 3=wall). You can also modify these files to test hypothesis about species behavior.

To visualize the result of the previous simulation you have to start Matlab. Once Matlab is started **execute the script created by antsim** to load the data of the simulation (Do not forget to change your "Working Directory"):

```
>> simul
Environment: 1
Species: 1
Food picked up (avg over 100 runs): 683.3
Food returned to the nest (avg over 100 runs): 292.3
```

It gives you the average performance of the ant species you selected. You can then either display an animation of the simulation to get a feeling on how the ants interact with the environment (for runs = 1):

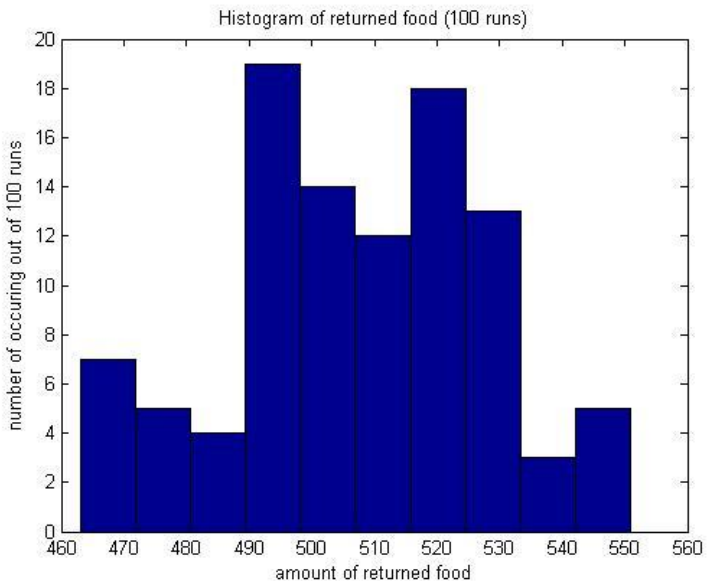
```
>> antsim_play
```

or a step by step animation with (for runs = 1):

```
>> antsim_stepbystep
```

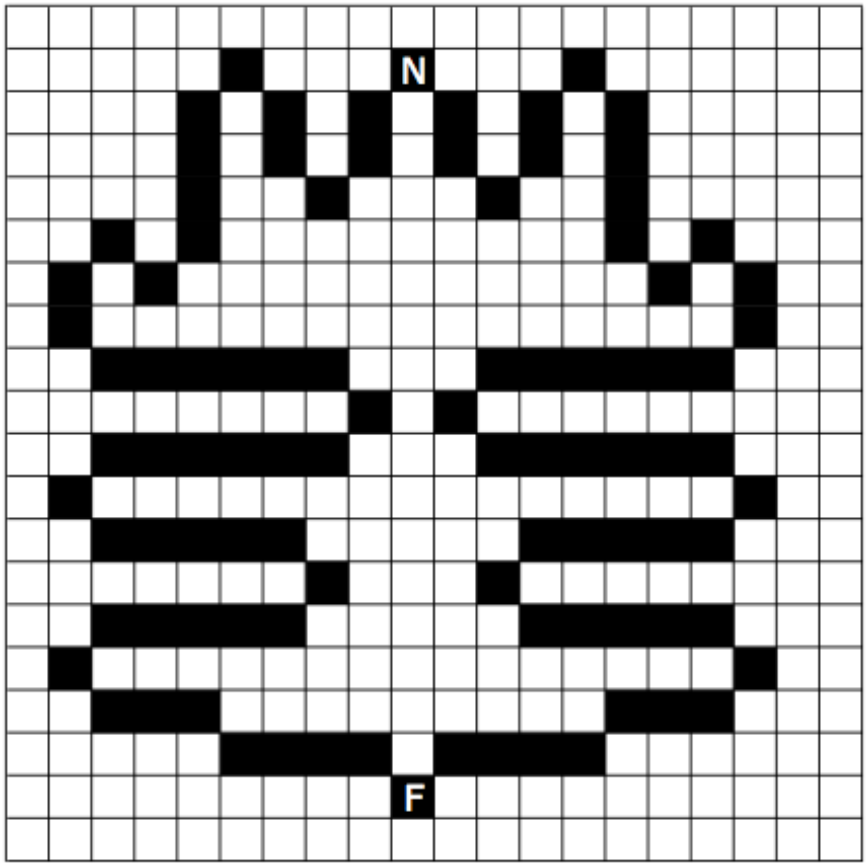
or display a histogram of the results (for runs > 1):

```
>> antsim_histogram
```



3.2 Two Equal Paths (Symmetric Bridge)

Environment 1 is as follows. N is the nest and F is the food. The **black** cells are paths open to the ants, and the **white** cells are obstacles. An ant can move to any of the 8 cells surrounding its current location (except for obstacles).
In each step, ants decide to either wander or follow the pheromone. A wandering ant moves randomly to any open cell (including the one it came from). An ant following the pheromone favors moving to cells with a high pheromone concentration and never goes back to the cell it just came from.



1. **(S)** Simulate the different species on this environment.

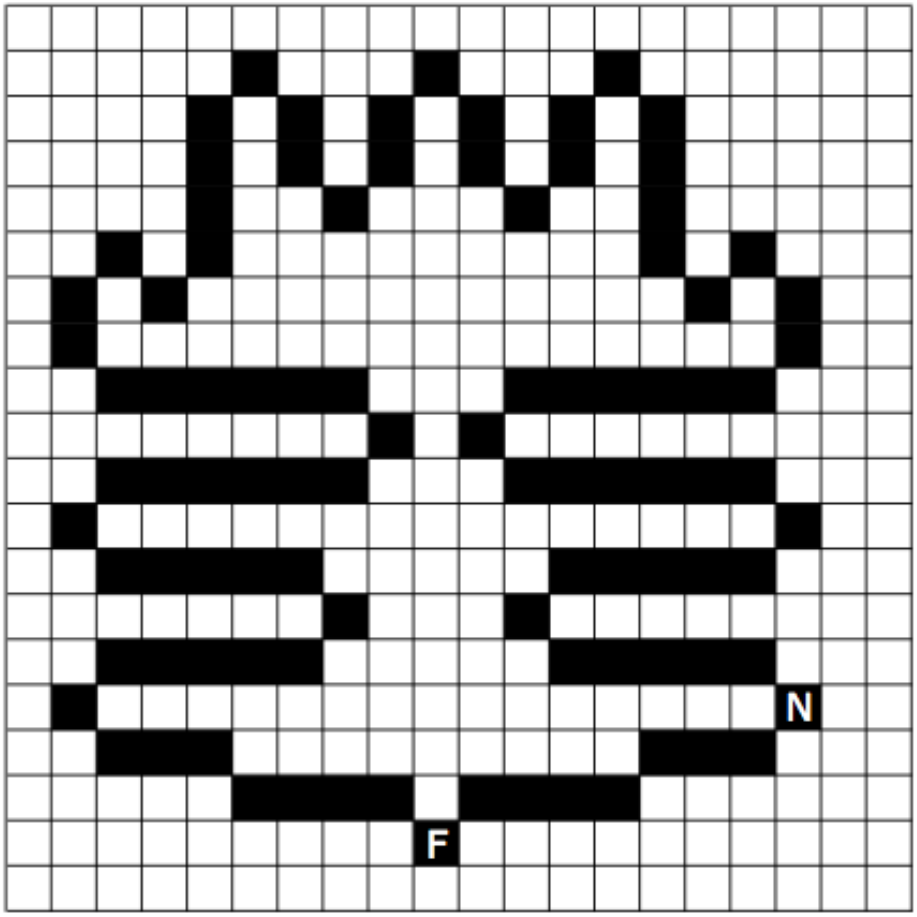
2. **(Q, 2):** Observe how each of the species perform in this environment. Try to take note of a few characteristics for each species behavior and rank the performances. Ties are possible.

3. **(Q, 3):** Explain the strategy of the fittest species.

4. **(Q, 5):** Explain the lower performance of each of the other species.

3.3 Two Unequal Paths (Asymmetric Bridge)

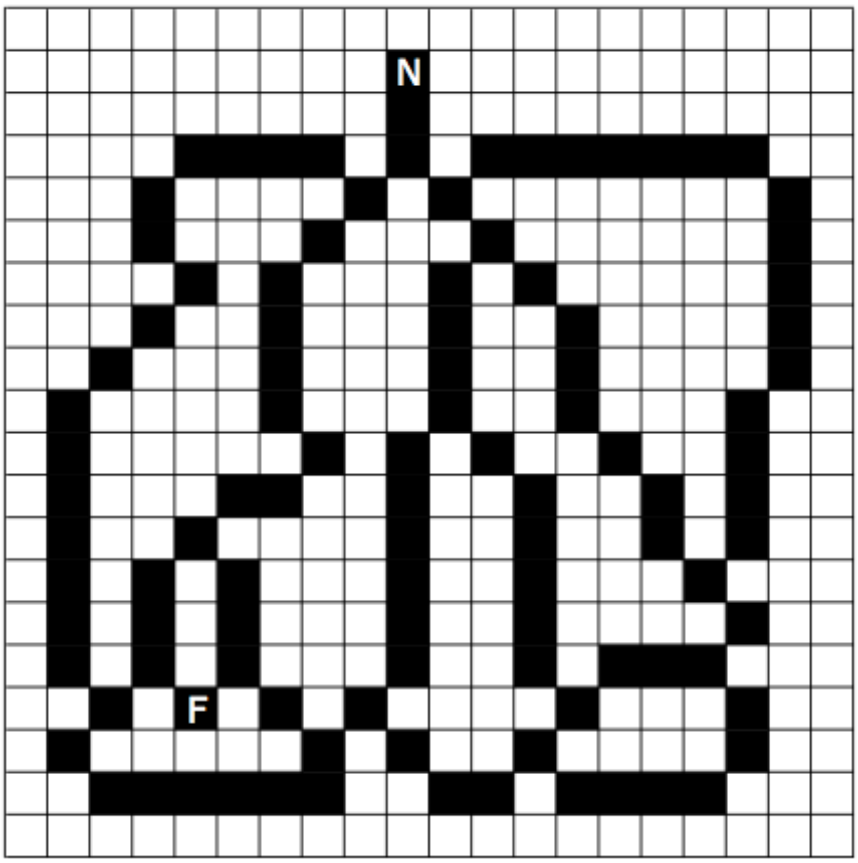
Environment 2 is as follows:



5. **(S) :** Simulate the different species on this environment.
6. **(Q, 2):** Observe how each of the species perform in this environment. Try to take note of a few characteristics for each species behavior and rank the performances. Ties are possible.
7. **(Q, 3):** Explain the strategy of the fittest species with respect to the particularity of this environment.
8. **(Q, 5):** Explain the lower performance of each of the other species.

3.4 Multiple Paths

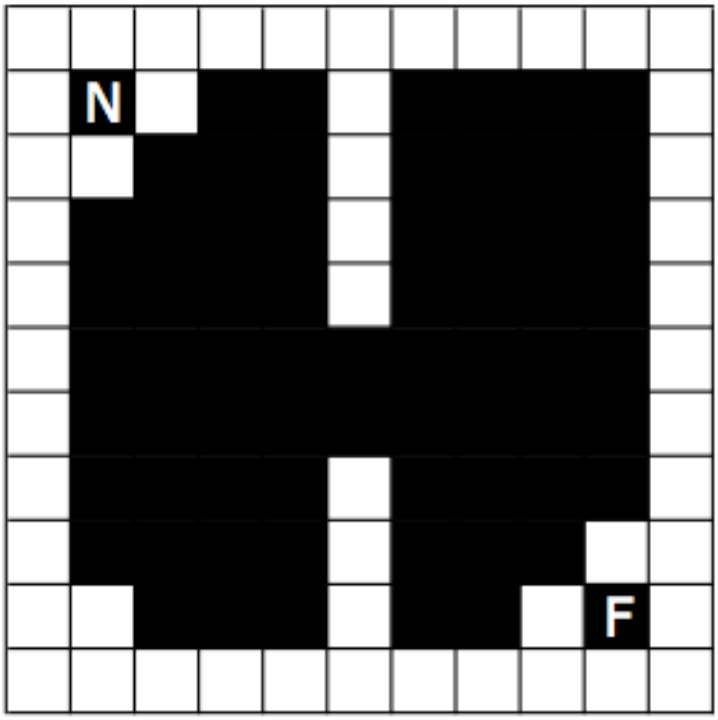
Environment 3 is as follows:



9. **(S)** Simulate the different species on this environment.
10. **(Q, 2):** Observe how each of the species perform in this environment. Try to take note of a few characteristics for each species behavior and rank the performances. Ties are possible.
11. **(Q, 3):** Explain the strategy of the fittest species with respect to the particularity of this environment.
12. **(Q, 5):** Explain the lower performance of each of the other species.

3.5 Open Terrain with Bridge

Environment 4 is as follows:

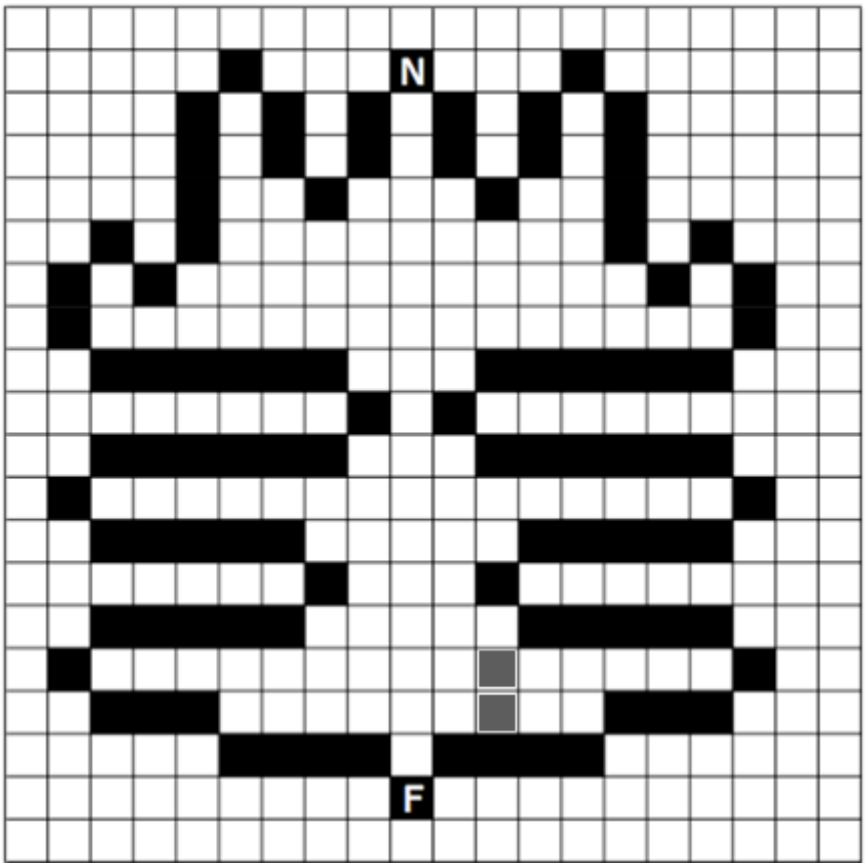


13. **(S)** Simulate the different species on this environment.
14. **(Q, 3)** You probably observed that all species except Sneezy got a very bad performance. Why are these species not successful? What problems are specific to this type of environment?
15. **(Q, 2)** How might these difficulties be avoided?
16. **(Q, 2)** Why does Sneezy still achieve acceptable performance?
17. **(Q, 3)** In many types of complex search systems there is a delicate balance between exploration and exploitation. Briefly explain the contribution of each of the species parameters (as defined in `species.c`) toward these conflicting goals.
18. **(I, 10)** The provided code has facility for the use of more than one type of pheromone (as defined by `PHER_KINDS` in `environment.h`). However, `antsim.c` only makes use of one pheromone. How might you use a second pheromone so that ants other than Sneezy can perform well in this environment? Modify `PHER_KINDS` in `environment.h` (one line), the species parameters in `species.c` (two lines) and the function `ant_strategy` in `antsim.c` (six lines) as necessary. The lines to be modified are marked in the code. Clean and compile the modified code, then run the code and observe the results. Try to understand and explain how the different types of pheromones are deployed and how this improves the performance of ants.

    Note: Please run the command `make clean` *to remove older binary objects before running* `make`.

3.6 Modifying the environment during the experiment

Environment 1 is modified as follows in the file `env1_mod.map`:



The new open cells are shown grey. You can also display the environment in Matlab by typing `display_env('env1_mod.map')`.

Note: Remove the modifications you made in section 3.5 and use the original code.

19. **(I, 7)** Read the file `antsim.c`. For each run there are `ITERS` iterations (`ITERS=1000` currently). See how the environment is loaded. There is a function called `environment_load_modified`, which loads a similar but modified environment (but preserves the existing pheromone levels in the cells). Increase the number of iterations to 2000. You now have to start the experiment with Environment 1, and modify the environment midway (say after half the number of iterations). To do so, you need to
    - Locate the place in code where you must call the function to load the modified environment.
    - Reset the environment to the non-modified version after each *run*.
20. **(Q, 4)** Without performing any experiment, can you predict which ant species is more likely to start using the new shorter path? Explain why.
21. **(I, 6)** Run the simulations with different species of ants, and see if your observations are in agreement with your prediction.

4 Lab Part 2: Ant Colony Optimization and the Traveling Salesman Problem

Ant Colony Optimization (ACO) algorithms are meta-heuristic approaches that allow solving a suite of hard optimization problems by using the ant colony/trail laying metaphor [Dorigo2004]. They are inspired by the optimization capabilities of foraging ants as it can be observed in the bridge experiments of J.L. Deneubourg.

The Traveling Salesman Problem (TSP) is a classical optimization problem. It deals with finding the shortest path that connect a number of cities, and passes every city once and only once. Due to its immediate connection to the shortest path problem that ants face during foraging, ACO approaches have been first tested on a TSP problem.

In this part of the lab you will familiarize yourself with different ACO algorithms to solve the TSP, and observe and evaluate their performance experimentally.

4.1 Basic Ant System

The first ACO algorithm, Ant System (AS), was presented in Marco Dorigo's PhD thesis in 1992. In AS, *m* ants travel from a random starting point from city to city until all *n* cities have been visited. Hereby, paths are chosen randomly with a probability which is a function of the amount of pheromones already deposited and of the distance to the next city. The ants are capable of memorizing the visited nodes.

The probability for an ant *k* at city *i* to go to city *j*, is then given by

$$p_{ij}^k = \frac{Q_{ij}}{\sum_{l \in J_i^k} Q_{il}} \quad \text{with } j \in J_i^k \qquad (1)$$

where J_i^k is the feasible neighborhood for ant *k* at city *i* and Q_{ij} a combined metric of the quality of the route. In AS, the quality of a route *ij* is a function of the pheromones already deposited by other ants given by τ_{ij}, and a heuristic $\eta_{ij} = 1/d_{ij}$, with d_{ij} the distance between city *i* and city *j*. Thus, Q_{ij} is defined as

$$Q_{ij} = [\tau_{ij}]^{\alpha} [\eta_{ij}]^{\beta} \qquad (2)$$

with α and β allowing us to fine tune the impact of pheromones and heuristic information on the metric.

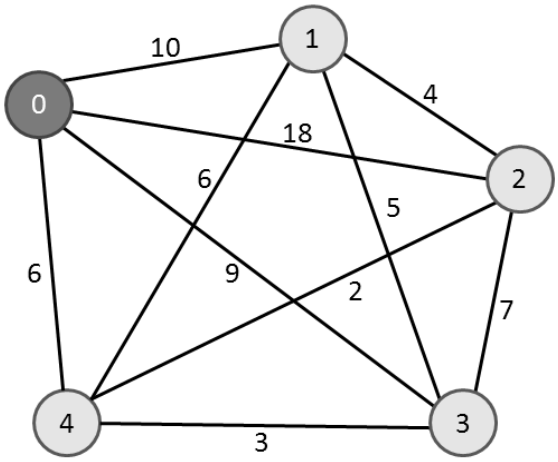


Figure 1: A simple instance of a TSP. Edges are marked with pheromones deposited by ants in previous iterations of the AS algorithm.

After completion of a tour, i.e. arriving at the starting city, ants assess the tour, and deposit an amount of pheromones that is inversely proportional to the tour length on every link *ij* they visited, i.e. $\Delta\tau_{ij}{}^{k} = Q/L^{k}$ with L^{k} the total length of the tour, and Q a parameter adjusted by heuristic. Here we simply set Q=1.

22. **(Q, 3)** Consider the TSP in Figure 1. Edges are marked with pheromones (the numbers next to the edges) that have been deposited during previous tours of an ant, which is now sitting at the dark grey vertex. What will be the route that is most likely taken by the ant (assume heuristic information not to be available/constant)? Your solution should be a sequence of integers.

23. **(Q, 3)** Again consider the TSP in Figure 1. Ignore now the pheromone values, and let the ant decide solely based on heuristic information. What will be the route that is most likely taken by the ant in this case? Keep in mind that the heuristic information associated with an edge is inversely proportional to the Euclidean distance d_{ij} between two vertices *i* and *j*, given by the following matrix $D = (d_{ij})_{i=0,\ldots,4;\ j=0,\ldots,4}$:

$$D = \begin{pmatrix} 0 & 4 & 7 & 10 & 5 \\ 4 & 0 & 3 & 7 & 8 \\ 7 & 3 & 0 & 5 & 9 \\ 10 & 7 & 5 & 0 & 6 \\ 5 & 8 & 9 & 6 & 0 \end{pmatrix}$$

Your solution should be a sequence of integers.

4.2 Elitist Ant System (EAS)

EAS is an updated version of AS that give the best route extra pheromone reinforcement.

24. **(S)** Change the working directory in Matlab to part 2 and type in the command window:

```
cities=randommap(6)
tsp('random',cities)
```

Which creates a matrix with 6 cities at random locations (2D coordinates), and solves the TSP by creating a random tour. You can also type `load eil51`, which will load a `cities` structure with 51 cities whose tour length minimal value is known to be 426 [TSPLIB].

25. **(Q, 7)** Look in the function `solvetsp_guess`, which is found in `tsp.m`. Explain which ant strategy lies in this code. Test your solution by running

```
tsp('guess',cities)
```

for different scenarios created using `randommap()` for a small number of cities, and qualitatively verify your idea using the `eil51` map.

26. **(I, 10)** The function `solvetsp_ant` provides a skeleton of an Elitist Ant System (EAS). In EAS the best ant of a tour is allowed to deploy extra pheromone. Implement the missing two lines that allow an ant to calculate the probability for choosing the next city according to the Equation 1.Test your solution by running

```
tsp('ant',cities)
```

27. **(S)** You can now compare the different approaches (random, guess, and EAS) using the command `testall(cities)`.

28. **(Q, 3)** Observe the number of tours that EAS needs for finding the best solution (you can increase the number of tours using the parameter `ntours`). What happens? Which feature of the algorithm would improve its global performance?

29. **(Q, 3)** The TSPs considered so far were fully connected, i.e. every city could be reached from every other city, leading to a quadratic distance matrix. How would you need to modify this matrix in order for taking into account cities that are not directly connected?

30. **(B, 10):** Look at the graph in Figure 2, and the accompanying distance matrix. One of your collaborators suggests an algorithm simpler than ACO for the shortest path problem represented in Figure 2: the ants lay continuously a constant amount of pheromone per distance unit (or per time unit since the average speed of the robots is assumed constant) while they travel (no backward ants). They prefer to choose paths characterized by higher virtual pheromone concentration (per unit length), but they do so probabilistically and have therefore a tendency to wander randomly. Will this approach work although there is no modulation of the dropped pheromone quantity as a function of the solution quality? If yes, what is the route that is most likely selected by the swarm? What are the underlying mechanisms that enable such a collective decision?

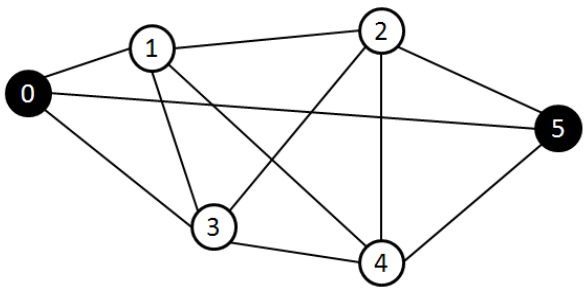


Figure 2: This graph represents an environment which contains one depot (0) and one goal (5). Nodes 1-4 are intersections of possible routes (time for traversing an intersection is negligible and inter-node distance include also node dimensions).

The Euclidean distances d_{ij} between two intersections i and j, are given by the following matrix $D = (d_{ij})_{i=0,\ldots,5;\, j=0,\ldots,5}$:

$$D = \begin{pmatrix} 0 & 2 & -1 & 5 & -1 & 11 \\ 2 & 0 & 5 & 3 & 6 & -1 \\ -1 & 5 & 0 & 5 & 4 & 4 \\ 5 & 3 & 5 & 0 & 2 & -1 \\ -1 & 6 & 4 & 2 & 0 & 4 \\ 11 & -1 & 4 & -1 & 4 & 0 \end{pmatrix}$$

4.3 Local Search

Whereas constructive algorithms like the EAS and its derivatives are constructing solutions in an incremental way using some heuristic rules, Local Search algorithms (LS) optimize a given solution by iteratively exploring neighborhoods of the solutions that are generated by local changes.

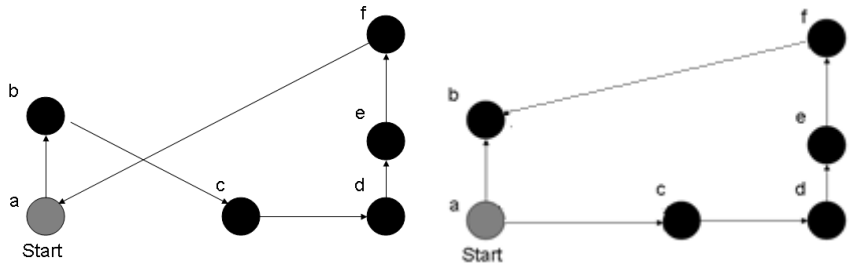


Figure 3: A simple TSP problem solved by a greedy algorithm (left) and optimized by a local search (right).

A neighborhood of a solution consists hereby of all possible solutions that can be achieved by k permutations, for instance by exchanging two edges as depicted in Figure 3.

31. **(Q, 4)** Write both tours (starting point is 'a' for both and the initial direction the same) of Figure 3 as a string/vector of characters and compare them. What happens to the order of a substring when you swap two edges?

32. **(B, 10)** A 2-opt local search systematically evaluates all permutations of a solution that can be achieved by exchanging two edges. Write the missing code to implement such permutation of edges in the function `local_search_2opt` in the file `tsp.m`. Skeleton of the function is already provided.

    Use the command `testall(cities,1)`. To test all of the algorithms in the file `tsp.m` with local search. The second parameter enables local search in all of the algorithms.

5 References

[Dorigo2004] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004.
[TSPLIB] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.