Final Project for the
Swarm Intelligence course

May 9, 2019

1 Modalities

The exam is divided in two parts:

**Project + presentation** You have to select one of the two projects proposed below and provide the required deliverables. Please fill out the following online form to indicate the project of your preference:

[https://forms.gle/mR6LeR6jx58e8jGS6](https://forms.gle/mR6LeR6jx58e8jGS6)

**IMPORTANT**: Please note that what you indicate is just a preference and it does not necessarily mean that the project is assigned to you. Since each project has a limit on the maximum number of students allowed, we will process all registrations on a *first come, first served* basis. In the next few hours after the registration period is closed, you will receive an email confirming the project you have been assigned. If you fail to register on time, please send an email to both aligot@ulb.ac.be and ccamacho@ulb.ac.be to be assigned to a project according to availability.

In addition, you will present your project in a 5-minute talk, followed by 5 minutes of questions. We strongly encourage you to focus **only** on presenting your ideas (how you solved the problem), your results and findings. The project and the presentation will account for up to 10 points of your final grade.

**Oral examination** You will be asked a number of questions concerning the **entire course material**. This will account for up to 10 points of your final grade.

To pass the exam, both parts must be above the threshold.
1.1 Timeline

- You have until **May 10** at 17:00 to register your preference for one of the two projects.
- On **May 10**, after 17:00, you will receive and email specifying the project you have been assigned.
- The project submission deadline is **June 7** at 23.59.
- Delay on the submission will entail a penalty of 1 point every 12 hours delay on the final evaluation of the project. Max delay is **June 10**, at 23:59. After this deadline you go to the second session.

1.2 Project Deliverables

For the project, you will have to provide:

- Your code in digital format, so we can test it (more details on the format are given on the description of each project). It has to be sent by e-mail to the responsible(s) for your project (see contacts at the end of the document).

- A short document (max 6 pages not counting the references\(^1\) single spaced, font 10) written in English that describes your work. You have to explain both the idea and the implementation of your solution.


- On the day of the oral examination, you are expected to show slides of your work. Please bring your own portable computer and make sure it is already switched on and ready when you are called in for your oral exam. The rest of the exam will consist of a question answering session on all the subjects covered by the course. If you happen to not have a laptop, send us a PDF version of your slides **before** the day of the exam.

---

\(^1\) You can add as many pages of bibliographical references as necessary to support your work.

\(^2\) [https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines](https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines)
2 Projects

2.1 General Remarks

Apply what you learned — It is very important that you stick to the principles of swarm intelligence: simple, local interactions, no global communication, no global information.

Avoid complexity — Good solutions are elegantly simple in most cases. So, if your solution is becoming too complex you are very likely in the wrong direction.

Honesty pays off — If you find the solution in a paper or in a website, cite the source and say why and how you used the idea.

Cooperation is forbidden — Always remember that this is an individual work.

The project counts for 50% of your final grade. The basic precondition for you to succeed in the project is that it must work. If it does not, the project will not be considered sufficient. In addition, code must be understandable — comments are mandatory.

The document is very important too. We will evaluate the quality of your text, its clarity, its completeness, and its soundness. References to existing methods are considered a plus — honesty does pay off! More specifically, the document is good if it contains all the information needed to reproduce your work without having seen the code and a good and complete analysis of the results.

The oral presentation is also very important. In contrast to the document, a good talk deals with ideas and leaves the technical details out. Be sure that it fits in the 5-minute slot. Because you only have 5 minutes for your presentation, you should focus on presenting what you have done and the results you obtained, rather than on the presentation of the problem.

2.2 Ant Colony Optimization

2.2.1 Introduction

For this project you are required to design, implement, evaluate and analyze the use of ACO algorithms for the Permutation Flow Shop Problem with Weighted Tardiness (PFSP-WT).

The flow shop problem has been widely studied in optimization and models a variety of scheduling problems that are used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. More formally, in this problem we are given a set of
\( N = \{1, \ldots, n\} \) independent jobs that has to be executed, in order, on a set of \( M = \{1, \ldots, m\} \) machines. Every job \( i \in N \) requires a certain fixed, non-negative processing time \( p_{ij} \) when executed on machine \( j, j \in M \). The completion time \( C_{i,j} \), that is the time needed to complete the processing of job \( i \) on machine \( j \), is defined according to the next equation:

\[
C_{i,j} = \max(C_{i-1,j}, C_{i,j-1}) + p_{ij}
\]

(1)

where \( C_{i-1,j} \) is the completion time of the job processed before \( i \) on machine \( j \) and \( C_{i,j-1} \) is the completion time of job \( i \) on the machine located before \( j \). Moreover, the completion time of the jobs on the first machine is equal to \( C_{i,1} = C_{i-1,1} + p_{i1} \), while the one of the first job of the schedule is equal to its processing time on the first machine.

The PFSP requires to find a certain sequence (schedule) for the \( N \) jobs such that some criterion is optimized; in the PFSP-WT the criterion is the **minimization of the total weighted tardiness**. Thus, in addition to the processing time, each job \( i \) has a due date \( d_i \), which denotes the desired completion time of the job on the last machine, and a priority weight \( w_i \), which denotes its importance. The tardiness of a job \( i \) is defined as \( T_i = \max\{C_i - d_i, 0\} \), where \( C_i \) is the completion time of job \( i \) in the last machine, and the total weight tardiness is given by \( \sum_{i=1}^{n} w_i \cdot T_i \). Note that the lateness of a job, defined as \( L_i = C_i - d_i \), can be positive or negative and, hence, the tardiness is \( \max\{C_i - d_i, 0\} \).

Jobs are taken in the order given by the schedule, and run on the \( M \) machines in order from 1 to \( m \). The sequence is the same for all the machines. When the first job of the sequence terminates its execution on machine 1, it goes to machine 2, and the second job of the sequence can start its execution on machine 1. This process continues until all \( n \) jobs have been executed on all \( m \) machines. In Figure 1 you can see two solutions for a flow shop scheduling of 10 jobs on 3 machines; the first solution is the most immediate one, that considers the jobs in the same order as they are given, while the second one is the scheduling given by the best permutation. Completion of a job after its due date is allowed, but a penalty is then incurred. For this project, you are given \( N, M \), the matrix \( p_{ij}, d_i \) and \( w_i \).

Additionally, the next assumptions and conditions must be satisfied for this problem:

- the processing time \( p_{ij} \) may be 0 if job \( j \) does not need to be executed on machine \( i \);
- all jobs are independent and ready to be executed at time 0;
- all the machines in \( M \) are always available;
- each job \( j \) can be processed on only one machine \( i \) at a time;
- no preemption is allowed, that is, a job cannot be interrupted;
- there is a positive tardiness if the job \( j \) finishes after its due date; otherwise tardiness for the job will be 0;
any machine \( i \in M \) has a queue of unlimited size. If a given operation needs an unavailable machine \( i \), the job enters the queue.

### 2.2.2 Goal

You have to implement two ACO algorithm variants (different than AS), test your implementations and write a report about your work analyzing the results. You can re-use the ACO implementations developed in class. However, note that a number of adaptations are needed to solve the PFSP-WT. You are free to define the components of the algorithms, as long as they follow the general definition of each algorithm. The implementation of ideas proposed in the literature (citation should be included) will be considered a plus. Once the algorithms are implemented, experiments should be carried out in order to compare and analyze their performance. Finally, the addition of local search to the ACO algorithms should be studied. You are free to propose any kind of local search for the problem.

Starting points for PFSP in the literature are [5], [7], [4], [2] and [6]. You are free to use ideas from the literature as long as the source is properly cited in your report.
2.2.3 Instances

Benchmark instances for the PFSP can be found at [http://iridia.ulb.ac.be/~ccamacho/INFO-H-414/project/PFSP_instances.zip](http://iridia.ulb.ac.be/~ccamacho/INFO-H-414/project/PFSP_instances.zip). We provide 20 instances of different size and type, taken from [9], publicly available in [8]. The format of each instance is the following:

- $N$ $M$
- processing times matrix $p_{ij}$. (*Note that in each row the processing time $p_{ij}$ is preceded by the number of the machine $j$.*)
- text string “Reldue”
- -1 due date -1 priority weight. (*The values -1, in the first and third column, are placeholders irrelevant for the problem at hand.*)

An example of instance file is the following:

```
data.txt
20 10
0 25 1 77 2 2 3 2 4 86 5 68 6 47 7 2 8 91 9 64
0 14 1 67 2 27 3 3 4 45 5 82 6 44 7 51 8 66 9 49
0 99 1 89 2 92 3 28 4 80 5 57 6 33 7 72 8 32 9 4
0 5 1 40 2 7 3 1 4 21 5 7 6 15 7 39 8 65 9 45
0 88 1 62 2 64 3 17 4 79 5 47 6 31 7 67 8 75 9 21
0 98 1 5 2 44 3 76 4 59 5 59 6 53 7 57 8 89 9 56
0 72 1 85 2 84 3 43 4 67 5 96 6 44 7 19 8 82 9 19
0 39 1 62 2 76 3 79 4 27 5 94 6 8 7 11 8 11 9 75
0 79 1 19 2 43 3 88 4 29 5 65 6 22 7 18 8 82 9 3
0 13 1 73 2 18 3 13 4 5 5 19 6 88 7 75 8 33 9 96
0 24 1 55 2 27 3 3 4 13 5 82 6 30 7 24 8 64 9 97
0 14 1 24 2 91 3 2 4 8 5 44 6 1 7 96 8 82 9 17
0 82 1 18 2 23 3 39 4 84 5 33 6 24 7 61 8 53 9 53
0 90 1 64 2 17 3 39 4 42 5 13 6 81 7 68 8 67 9 38
0 28 1 41 2 84 3 16 4 15 5 88 6 83 7 1 8 21 9 25
0 50 1 55 2 90 3 95 4 18 5 93 6 96 7 35 8 24 9 7
0 75 1 37 2 98 3 40 4 13 5 78 6 10 7 21 8 44 9 25
0 86 1 44 2 93 3 13 4 18 5 4 6 86 7 9 8 59 9 24
0 57 1 45 2 96 3 27 4 75 5 57 6 15 7 27 8 39 9 82
0 4 1 5 2 56 3 73 4 23 5 44 6 98 7 63 8 84 9 53
Reldue
-1 1039 -1 6
-1 3499 -1 2
-1 3638 -1 7
-1 1667 -1 4
-1 3240 -1 5
-1 4027 -1 2
-1 3512 -1 1
-1 3892 -1 4
-1 1054 -1 8
-1 1906 -1 5
```
2.2.4 Deliverables

The final deliverable will be a zip file containing the next three items (please create a folder for each one):

- Report (pdf format)
- Source code
- Results

Please follow these guidelines for each deliverable:

Report

1. Implementation of **two ACO algorithm variants** to solve the PFSP

   (a) Describe the implemented algorithms. Please explain only the relevant design decisions (representation of solutions, pheromone and heuristic information definition, update mechanism, transition rule, etc.).

   (b) Set the parameters of the ACO algorithms using the provided training set and report the best parameter settings for each algorithm. Describe the process you used to obtain the parameter settings. If you use automatic tuning, please report: parameters tuned, set of values (domain) for each parameter and number of evaluations allowed for the tuner (some freely available tuners are [3] and [1]).

   (c) For each algorithm, perform 10 runs on each instance of the instance set using a budget of 30-second wall clock execution time. 

   **Note 1:** you must use the same budget for both algorithms. 

   **Note 2:** you must use paired seeds as examples in class, that is, the same seed has to be used to try different algorithms (or parameter settings) in one run.

   (d) Include in the report a table showing, for each instance, the best (B), worst (W), mean (M) and standard deviation (SD) obtained by each algorithm, where each row of the table is an instance.

   (e) Compare the performance of the algorithms in the report:

   - Use plots and/or tables to show the results.
   - Perform a pairwise comparison of the algorithms using the Wilcoxon rank-sum test.
   - Based on the results, which of the implemented algorithms would you recommend for solving this problem? Comment.
• Compare the convergence of the algorithms.
  (Note: The goal is to compare the computational effort (e.g. number of evaluations of the objective function) vs. solution quality).

2. Choose an algorithm and add local search to it:
   (a) Describe the local search procedure implemented.
   (b) Perform 10 runs per instance for the new algorithm (use the same budget of 30-second wall clock execution time) and compare it with its version without local search. Report as above.
   (c) Are there any improvements? Why? Comment.

3. Tune the parameters of the algorithm with local search. Analyze your results, is the algorithm more sensitive to some parameters?

Code
• The code should run in the same way as the code developed in class, that is, using the same command line arguments. Feel free to add new arguments in case you need them.
• The code should be properly commented and indented.
• The code should include a README file with the main instructions and specifications of your algorithms and parameters.
• You must implement your own code. Plagiarism will be strongly penalized.
• Make sure your implementation works on Linux.

Results
• For each instance/algorithm create a csv file (algorithm_name-results.csv) with the data required in Section 2.2.4 point 1d.
• Include a copy of all graphs (box plots, convergence, etc.) and statistical tests carried out to analyze the performance of your implementations (Section 2.2.4 point 1e). Name the files using the structure algorithm_name-instance-{bxp|cvg|wt|*} according to the case.

2.3 Swarm Robotics: Foraging without Ambient Cues
The activity of food search and retrieval is commonly referred to as foraging. In swarm robotics, foraging is a commonly used task to compare different algorithms for exploration (what is the best way to discover interesting places in the environment?), division of labor (who should explore? for how long?),
etc. In the most general setting, food items are scattered in an environment at locations unknown to the robots and the robots need to explore the environment, find the food, and take it to the nest. The foraging environments often contain cues, such as light sources, that help the robots in navigating through the environment.

In this project, we consider a foraging environment without any cue to be used with navigation purposes. The swarm is distributed in the arena and the location of the food source is originally unknown to the robots. The swarm must act cooperatively to discover and keep track of the location of the food source for an efficient foraging. The students are asked to develop and provide control software for the robot swarm, and to experiment and evaluate its performance.

2.3.1 Problem definition

The robot swarm operates in a diamond-shaped arena that includes a food source and a nest. The food source is represented by a black circle and the nest is represented by a white area in the opposite side of the diamond. The robots can perceive the ground color through the ground sensor.

Goal The goal of the robot swarm is to retrieve and transport items from the food source to the nest. The overall performance of the swarm is measured by the number of items it is able to collect during a fixed experiment time. Each experiment is automatically terminated after 1000 seconds (10000 time-steps). More precisely, the performance of the swarm is described by

\[ \max N_d \]

with \( N_d \) being the number of items collected from source and successfully delivered to the nest.

Swarm composition The swarm comprises 20 homogeneous robots. The robots are equipped with the following sensors and actuators:

- colored_blob.omnidirectional_camera to detect the walls LEDs;
- range_and_bearing to communicate between robots;
- light to perceive the light above the nest;
- motor_ground to sense the color of the ground;
- proximity to avoid obstacles;
- wheels to explore the environment;
- leds to display information.
Figure 2: Top view of the arena. The white area indicates the position of the nest. The black area indicates the position of the food source. The dashed area indicates the region on which the center of the food source is randomly located.

Additional Remarks

- The maximal wheel velocity should not exceed 15 cm/s.
- ARGoS has been configured so that the robots collect items on the food source and drop them at the nest automatically. A robot can only carry one item at a time.
- The position of the nest is fixed, but the position of the food source can change from one experiment to another. See Figure 2.

2.3.2 Requirements

The goal of this project is to design, implement and evaluate control software that aims to maximize the number of items delivered at the nest. The control software has to demonstrate a cooperative behavior: one that takes advantage of the swarm’s principles.

ARGoS is configured to automatically dump data on a file whose name can be changed in the .argos experiment configuration (see Section 2.4). This file contains a table with two columns:

- CLOCK: Column indicating the current step
- ITEMS: Column indicating the number of items collected so far
A complete analysis must be performed to evaluate the quality of the controller implemented. In particular, to present statistically meaningful results, we suggest you to execute and collect the results over at least 30 runs of the control software. The 30 runs should differ from each other for the random seed specified in the .argos experiment. In your report, you will include a table which will contain the following columns:

- Random seed
- Final number of items collected from source

Beside these two tables, quantitative numerical measures of the performance of the two solutions must be produced. Specifically, you should produce i) a plot that shows the trend over time of the items collected by the swarm; ii) a plot that shows the variance of the number of items collected by the swarm in different runs. On the basis of these measures, you should discuss the results and draw the appropriate conclusions.

Be aware that, for the project evaluation, the analysis is as important as the implementation. Make sure that the information provided in the report is meaningful, clearly written and complete. Any meaningful additional content will be rewarded (e.g., comparison of different control software).

### Setting up the code


- Unpack the archive into your $HOME directory and compile the code

  ```
  $ tar -xzf SR_Project_H414.tar.gz  # Unpacking
  $ cd SR_Project_H414               # Enter the directory
  $ mkdir build                     # Creating build dir
  $ cd build                        # Entering build dir
  $ cmake ../src                    # Configuring the build dir
  $ make                            # Compiling the code
  ```

- Set the environment variable ARGOS_PLUGIN_PATH to the full path in which the build/ directory is located:

  ```
  $ export ARGOS_PLUGIN_PATH=$HOME/SR_Project_H414/build/
  ```

  You can also put this line into your $HOME/.bashrc file, so it will be automatically executed every time you open a terminal.
• Run the experiment to check that everything is OK:

```
$ cd $HOME/SR_Project_H414  # Enter the directory
$ argos3 -c foraging.argos      # Run the experiment
```

If the usual ARGoS GUI appears, you’re ready to go.

### 2.4 Setting up the experiment

**Switching the visualization on and off.** The experiment configuration file allows you to launch ARGoS both with and without visualization. When you launch ARGoS with the visualization, you can program the robots interactively exactly like you did during the course. Launching ARGoS without the visualization allows you to run multiple repetitions of an experiment automatically, e.g., through a script. By default, the script launches ARGoS in interactive mode. To switch the visualization off, just substitute the visualization section with: `<visualization />`, or, equivalently, comment out the entire qt-opengl section.

**Loading a script at init time.** When you launch ARGoS without visualization, you cannot use the GUI to set the running script. However, you can modify the XML configuration file to load automatically a script for you. At line 50 of foraging.argos you’ll see that the Lua controller has an empty section `<params />`. An example of how to set the script is at line 53 of the same file. Just comment line 50, uncomment line 53 and set the script attribute to the file name of your script.

**Changing the random seed.** When you want to run multiple repetitions of an experiment, it is necessary to change the random seed every time. To change the random seed, set the value at line 11 of foraging.argos, attribute random_seed.

**Changing the output file name.** As explained above, ARGoS automatically dumps data to a file as the experiment goes. To set the name of this file, set a new value for the attribute output at line 17 of foraging.argos.

**Making ARGoS run faster.** Sometimes ARGoS is a little slow, especially when many robots and many sensors are being simulated. You can make ARGoS go faster by setting the attribute threads at line 9 of foraging.argos. Experiment with the values, because the best setting depends on your computer.

### 2.4.1 Deliverables

The final deliverables must include source code and documentation:

**Code:** The Lua scripts that you developed, well-commented and well-structured.
Documentation: A report of up to 6 pages structured as a scientific article and containing:

- Main idea and swarm robotics principles in your approach.
- Structure of your solution (the state machine).
- Analysis and discussion of the results.

See Section 1.2 for more details about the deliverables.

3 Contacts

Marco Dorigo mdorigo@ulb.ac.be for general questions
Mauro Birattari mbiro@ulb.ac.be for general questions
Christian Camacho Villalón ccamacho@ulb.ac.be for ACO
Federico Pagnozzi federico.pagnozzi@ulb.ac.be for ACO
Antoine Ligot aligot@ulb.ac.be for swarm robotics
David Garzon Ramos dgarzonr@ulb.ac.be for swarm robotics

4 Bibliography

References


