Final Project for the Swarm Intelligence course

May 3, 2018

1 Modalities

The exam is divided in two parts:

Project You have to pick a project among those proposed below and provide the required deliverables. In addition, you will present your project in a 7-minute talk, followed by 5 minutes of questions. This will account for up to 10 points of your final grade.

Questions 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

- 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.
- By **May 10**, at maximum, you must have emailed your choice (ACO or Swarm Robotics) to both responsibles of the topic of the project you choose (see contacts at the end of the document).

1.2 Project Deliverables

For the project, you will have to provide:

- Your code in digital format (i.e., text files), so we can test it. Send it by e-mail to the people responsible for your project (see contacts at the end of the document);

- A short document (max 10 pages, single spaced, font 12) written in English that describes your work. You have to explain both the idea and the implementation of your solution. The document must be in the format of a scientific article, including abstract, introduction, short description of the problem, description of the solution, results, conclusions and references (see the examples available at `http://iridia.ulb.ac.be/~ccamacho/INFO-H-414/project/examples/`).

- On the day of the oral presentation, you are expected to show slides and come with your own laptop. If you happen to not have a laptop, send us a PDF version of your slides **before** the day of the exam.

2 Projects

2.1 General Remarks

Apply what you learned It is mostly 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 it and say why you used that idea and how.

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 7-minute slot.

2.2 Ant Colony Optimization

2.2.1 Introduction

For this project your are required to design, implement, evaluate and analyze the use of ACO algorithms for the *Quadratic Assignment Problem* (QAP).

The Quadratic Assignment Problem is one of the hardest problems in the NP-hard class. It has several practical applications as, for example, assigning a set of facilities to a set of locations in such a way as to minimize the total assignment cost; or placing interconnected electronic components onto a printed circuit board so as to minimize the wiring. The QAP is an important problem also because a number of optimization problems can be formulated as a QAP, such as, for example, the traveling salesman problem and many scheduling problems.

The QAP can be stated as the problem of assigning a set of facilities to a set of locations with given distances between the locations and given flows between the facilities. The goal is to place the facilities on locations in such a way that the sum of the product between flows and distances is minimal.

Formally, given n facilities and n locations, two $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{rs}]$, where a_{ij} is the distance between locations i and j and b_{rs} is the flow between facilities r and s, we seek to minimize:

$$\min_{\psi \in S(n)} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\psi_i \psi_j} \tag{1}$$

where $S(n)$ is the set of all permutations (assignments) of the set of integers $\{1, \ldots, n\}$, and ψ_i gives the location of facility i in the current solution $\psi \in S(n)$. Here $b_{ij} a \psi_i \psi_j$ describes the cost contribution of simultaneously assigning facility i to location ψ_i and facility j to location ψ_j.

As an example, consider the next two matrices. The distances between locations are given in matrix A and the flows between facilities in matrix B

$$A = [a_{ij}] = \begin{bmatrix} 0 & 3 & 6 & 4 & 2 \\ 3 & 0 & 2 & 3 & 3 \\ 6 & 2 & 0 & 3 & 4 \\ 4 & 3 & 3 & 0 & 1 \\ 2 & 3 & 4 & 1 & 0 \end{bmatrix} \qquad B = [b_{rs}] = \begin{bmatrix} 0 & 10 & 15 & 0 & 7 \\ 10 & 0 & 5 & 6 & 0 \\ 15 & 5 & 0 & 4 & 2 \\ 0 & 6 & 4 & 0 & 5 \\ 7 & 0 & 2 & 5 & 0 \end{bmatrix} \tag{2}$$

Using Equation 1, a permutation [1 2 3 4 5], where facility b_1 corresponds to location a_1, b_2 to a_2, b_3 to a_3, b_4 to a_4 and b_5 to a_5, has a cost of 374. In this example, the optimal solution has a cost of 258 and is obtained with permutation [2 4 5 3 1], where facility b_1 corresponds to a_5, b_2 to a_1, b_3 to a_4, b_4 to a_2 and b_5 to a_3, this is:

$$B_{[2\ 4\ 5\ 3\ 1]} = \begin{bmatrix} 0 & 6 & 0 & 5 & 10 \\ 6 & 0 & 5 & 4 & 0 \\ 0 & 5 & 0 & 2 & 7 \\ 5 & 4 & 2 & 0 & 15 \\ 10 & 0 & 7 & 15 & 0 \end{bmatrix} \tag{3}$$

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 adaptations are needed to solve the QAP problem. 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 QAP in the literature are [1], [5] 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 QAP can be found at `http://iridia.ulb.ac.be/~ccamacho/INFO-H-414/project/QAP_instances.zip`. We provide 10 instances of different size and type, taken from [2]. The format of each instance line is the following:

1. size of the instance (n)
2. matrix A (size $n \times n$)
3. matrix B (size $n \times n$)

Note that matrix A and B are either flow or distance. An example of instance file is the following:

`data.txt`

```
15

 0 12 86 68  0  0  0  0  0  0  0  0  0  0  0
12  0  0  0 69  0  0  0  0  0  0  0  0  0  0
86  0  0  0  0 34 35  0  0  0  0  0  0  0  0
68  0  0  0  0  0  0 19  0  0  0  0  0  0  0
```

```
 0 69  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0 34  0  0  0  0  0  0  0  0  0  0  0  0
 0  0 35  0  0  0  0  0  2  0  0  0  0  0  0
 0  0  0 19  0  0  0  0  0 21 88  0  0  0  0
 0  0  0  0  0  0  2  0  0  0  0 79  0  0  0
 0  0  0  0  0  0  0 21  0  0  0  0  3  0  0
 0  0  0  0  0  0  0 88  0  0  0  0  0 40  0
 0  0  0  0  0  0  0  0 79  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  3  0  0  0  0 77
 0  0  0  0  0  0  0  0  0  0 40  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0 77  0  0

 0 84 81 10 40 39  5 83 10 78 88 74 88 16 21
84  0 39 67 62 38  3 16 43 47 70 97  4 17 34
81 39  0 53 12 50 98 94  4 16 24 17 20 38 60
10 67 53  0 43 90 84 60 50 43 46 80 53 20  1
40 62 12 43  0 86 18  1 28 14 58 34  2  3 55
39 38 50 90 86  0 58 92 90 62 99 93 64 50 83
 5  3 98 84 18 58  0 53 93 69 98 27 19 50 86
83 16 94 60  1 92 53  0 30 35 39 55 25 12 41
10 43  4 50 28 90 93 30  0 16 10 96  7 11 52
78 47 16 43 14 62 69 35 16  0 28 13 97 63 91
88 70 24 46 58 99 98 39 10 28  0 94 91 44 62
74 97 17 80 34 93 27 55 96 13 94  0 48 36 11
88  4 20 53  2 64 19 25  7 97 91 48  0 69 80
16 17 38 20  3 50 50 12 11 63 44 36 69  0 41
21 34 60  1 55 83 86 41 52 91 62 11 80 41  0
```

2.2.4 Deliverables

The final deliverable report should include:

- Report (pdf format)
- Source code
- Results in *csv* format

Please follow this guideline for you deliverable:
Report:

1. Implementation of two ACO algorithm variants to solve the QAP
   (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 [4] and [3]).

(c) For each algorithm, perform 10 runs on each instance of the instance set.

Note 1.: you **must** use the same budget (number of evaluations) 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) For each instance/algorithm report: best (B), worst (W), mean (M) and standard deviation (SD) in 2 *csv* files (algorithm_name-results.cvs), where each row 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 computation effort (evaluations) 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 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. Analyse 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 ordered (indent!).

- 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.

2.3 Swarm Robotics: *Foraging with 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.

In this project, we consider food items with different energy levels: one positive and one negative. The food is gathered in two areas of the environment according to their energy level. The location of the source that yields items with positive energy level is unknown to the robots. The students are asked to provide and study the performance of two control software for robot swarms.

2.3.1 Problem definition

The robot swarm operates in a dodecagonal arena composed of two target sources and a nest. The nest is represented by a black rectangle drawn on the floor. The sources are represented by circles drawn on the floor with two shades of gray: `Dark` (i.e., 0.15) and `Light` (i.e., 0.85). The robots can perceive the ground color through the ground sensor. Some of the walls of the arena comport LEDs that can be blue or green. The robots can perceive the LEDs of the walls through the omnidirectional camera. Finally, a light is placed above the nest. The robots can perceive it with the light sensor and can use it to navigate in the environment.

Each of the source possesses an unlimited number of food items. One of the source yields items with energy level of value 3, while the other one yields items with energy level of value -1. The walls of the arena indicate which source yields items with positive energy level:

1. the `Dark` source if the walls are GREEN;
2. the `Light` source if the walls are BLUE.

Each experiment is automatically terminated after 1000 seconds (10000 time-steps).

Goal The goal of the robot swarm is to maximize the energy level E retrieved at the nest during the whole duration of the experiment, that is

$$\max E = N_d e_d + N_l e_l \quad \text{with} \begin{cases} e_d = 3, e_l = -1 & \text{if walls are GREEN} \\ e_d = -1, e_l = 3 & \text{if walls are BLUE} \end{cases} \tag{4}$$

with N_d being the number of items collected from source `Dark`, and N_l the number of items collected from source `Light`.

Swarm composition The swarm comprises 30 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.

Additional Remarks

- The maximal wheel velocity should not exceed 15 cm/s.
- ARGoS has been configured so that the robots collect items on sources and drop them at the nest automatically. A robot can only carry one item at a time. If a robot goes to source `Dark` then to source `Light`, the item collected at source `Dark` will be replaced by an item from source `Light` (independently of the energy level of the items).
- The positions of the sources are fixed, but their colors can change from one experiment to another. See Figure 1.

2.3.2 Requirements

The goal of this project is to design, implement, test, and compare **two control softwares** that try to maximize the energy value collected at the nest. The first control software to be implemented is an *individualistic* one: a solution in which the robots do not cooperate to accomplish the task. The second control software has to demonstrate a *cooperative* behavior: one that take advantage of the swarm's principles.

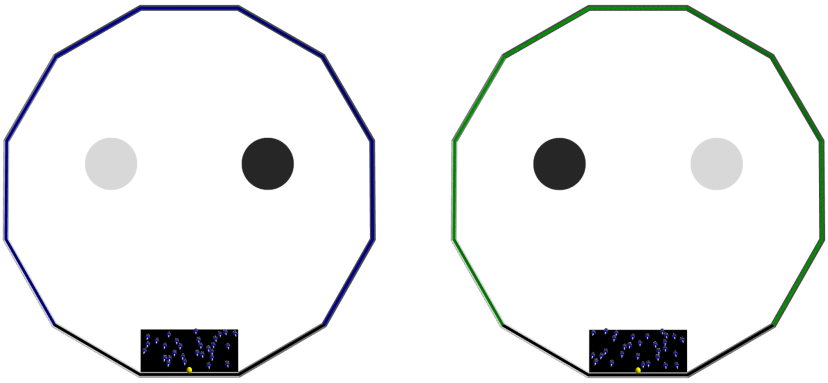

Figure 1: Top view of possible arenas. *Left*: BLUE walls indicate that source `Light` yields items with positive energy values. *Right*: GREEN walls indicate that source `Dark` yields items with positive energy values.

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 is composed of a line indicating which source has the high energy level food items, and a table containing four columns:

- The current step
- The number of items collected from source `Dark`
- The number of items collected from source `Light`
- The total energy level E.

A complete analysis must be performed to evaluate and compare the quality of the two controllers implemented. In particular, to present statistically meaningful results, we suggest you to execute and collect the results over at least 30 runs of each controller. The 30 runs of a controller should differ from each other for the random seed specified in the .argos experiment. To ensure a fair comparison, you should use the same 30 random seeds for both controllers. In your report, you will include two tables, one for each controller implemented, which will contain the following columns:

- Random seed
- Final number of items collected from source `Dark`
- Final number of items collected from source `Light`
- Final energy level E.

Beside these two tables, quantitative numerical measures of the performance of the two solutions must be produced. Specifically, you should produce i) plots that show the trend over time of the two controllers implemented—e.g. number of items collected from one source against number of items collected from the other source, ii) a plot that compares the performance of the two controllers, and iii) a statistical significance test that proves whether the two controllers are significantly different. 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.

Setting up the code

- Clone the argos3-arena repository into your $HOME directory

```
$ git clone https://github.com/demiurge-project/argos3-arena.git
```

- Compile the code

```
$ cd argos3-arena                # Enter the directory
$ mkdir build                    # Creating build dir
$ cd build                       # Entering build dir
$ cmake ../src                   # Configuring the build dir
$ make                           # Compiling the code
```

- Download the experiment files: SR_Project_H414.tar from `http://iridia.ulb.ac.be/courses/2018/h-414/SR_Project_H414.tar.gz`.

- 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 console.

- 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 10 pages structured as a scientific article and containing:

- Main idea of your approaches.
- Structure of your solutions (the state machines).
- Analysis and comparisons 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	christian.camacho.villalon@ulb.ac.be	for ACO
Hayfa Hammami	haifa.hammami@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

[1] Rainer E Burkard, Eranda Cela, Panos M Pardalos, and Leonidas S Pitsoulis. The quadratic assignment problem. In *Handbook of combinatorial optimization*, pages 1713–1809. Springer, 1998.

[2] Rainer E Burkard, Stefan E Karisch, and Franz Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997.

[3] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configurationframework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[4] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[5] Vittorio Maniezzo and Alberto Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on knowledge and data engineering*, 11(5):769–778, 1999.

[6] Thomas Stützle and Marco Dorigo. ACO algorithms for the quadratic assignment problem. *New ideas in optimization*, (C50):33, 1999.