Swarm Intelligence Course (INFO-H-414)
Exams

July 14, 2017

1 Modalities

The exam is divided in two parts:

Project You can choose a project among the ones proposed below. You are asked to provide the required deliverables and to 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 one must obtain at least 5 points in each part of the exam.

1.1 Timeline

- The date of the exam is September 6th.
- The project submission deadline is August 25th at 23:59. There is a 1-point penalty for each 12 hours of delay, with a maximum delay permitted of 48 hours (August 27th at midnight).
- Email your choice (PSO or Swarm Robotics) to both the 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 (6-8 pages, single spaced, font size 12) written in English that describes your work. You have to explain both the idea and the implementation of your solution, follow the instructions provided for each project.
- 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 If your solution gets too complex, it is because it is the wrong one.

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 Particle Swarm Optimization

For this project your are required to design, implement, evaluate and analyze the use of a Particle swarm optimization algorithm (PSO), to minimize a set of single objective continuous optimization functions proposed in the context of the CEC 2015 Competition on Learning-based Real-Parameter Single Objective Optimization [1].

The CEC 2015 competition defines a set of 15 test functions that have different characteristics. A complete description of these functions can be found in the material provided by the competition in [3]. Table I of the referenced document, gives a list of the functions used in the problems and the value of the optima for each of them. These functions define problems that minimize a function $f(x)$, where x is a set of D continuous variables. More details about the problem used in this project in in section 2.2.2.

2.2.1 Goal

You have to implement a PSO for solving the CEC 2015 competition test problems, evaluate the performance obtained by your implementation, and write a report about your work analyzing the results. You can re-use the PSO implementation developed in class. You are free to define the components of the algorithm as you think best, as long as they follow the general definition of PSO. The implementation of ideas proposed in the literature (citation should be included) will be rewarded. Once the algorithm is implemented, experiments should be carried out in order to compare and analyze the performance. We remark that you are free to use ideas of the literature as long as the source is properly cited in your report.

2.2.2 Instances

This project will use as problem instances the 15 test problems defined by the CEC 2015 competition (`https://tinyurl.com/y9yat2tf`). Note that in these problems, the global optima of the original functions are shifted and the functions are rotated. For this project, the number of dimensions must be set to $D = 30$ and the domain of the variables (x_i) is restricted to the range $[-100, 100]$ for all problems. The competition provides code in C (cec15-c-code folder), Java and Matlab to compute the evaluation of solutions, this code can be found in: `https://tinyurl.com/yaurzv83`. **You must use the provided implementation to evaluate solutions**, modify the methods implemented in the provided code, if required, to adapt them to your PSO implementation.

2.2.3 Deliverables

The final deliverable report should include:

- Report (pdf format)
- Source code (include code for evaluating the functions)
- Results in *csv* format

Please follow this guideline for you deliverable:

Report:

1. Implement a PSO algorithm for solving the CEC functions. Please explain the relevant design decisions. Choose a set of initial parameter values for the PSO algorithm, justify your choices in the report.
2. Set as termination criterion $10000 * D$ evaluations. Using the *gbest* topology and the chosen initial parameter settings, execute the PSO algorithm 10 times on each problem.
   - For each problem report: best (B), worst (W), mean (M) and standard deviation (SD) of the 10 runs in a *csv* file (pso-initial-results.cvs), where each row is a problem.
3. Set the parameters of the PSO algorithm and analyze their values (do not include the topology in these tests). 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 [2]).
4. Execute the tuned version of your algorithm as previously described (10 repetitions, $10000 * D$ evaluations).
   - For each instance report: best (B), worst (W), mean (M) and standard deviation (SD) of the 10 runs in a *csv* file (pso-tuned-results.cvs), where each row is an instance.
   - Compare in the report the performance of the tuned PSO and the PSO with the chosen initial parameter settings. Compare the results on the relative percentage deviation to the optimum provided for the problems in the CEC document.

     $$rpd = \frac{(solution - optimum) * 100}{optimum} \quad (1)$$

     Use plots and/or tables to show the results. Aggregate the results of all the problems.
     Note: you **must** use paired seeds for the 10 runs as explained in class.
   - Compare the convergence of the tuned and non-tuned algorithm. Provide a convergence plot.
     Note: The goal is to compare the computation effort (evaluations) vs. solution quality.
5. Choose a different topology, it can be one described in class or any other you consider interesting to test. Execute experiments to compare the results obtained with this topology and *gbest* (used the tuned parameter settings).
   (a) Perform experiments as previously described. Report as above.
   (b) Compare the convergence of the different topologies. Provide a convergence plot.

(c) Are there any improvements? Why? Comment.

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 penalized.
- Make sure your implementation works on Linux.

2.3 Swarm Robotics: *Foraging with Collective Decision Making*

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 robot swarms with the capability of discriminating between two foraging sites and collectively forage from the best one.

2.3.1 Problem definition

The robot swarm operates in a dodecagonal environment composed of two target sources and a nest. Each target source is represented as a circle with different quality value v_i determined as the ground color value, with $v_i \in [0.05, 0.95]$ and $i \in [0, 1]$. Each of the source possesses an unlimited number of food items. The darker source yields items with energy level of value 3 while the lighter one yields items with energy level of value -1. The robots can perceive the ground color through the ground sensor. In order for the robots to differentiate the two sources, a cylinder object marked with a red or blue LED is placed in the middle of each source. The robots can perceive the LEDs through the omnidirectional camera. A light is placed above the rectangular nest. The robots can perceive it with the light sensor and can use it to navigate in the environment.

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

Objective 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

$$\max E = N_0 e_0 + N_1 e_1 \quad \text{with} \begin{cases} e_0 = 3, e_1 = -1 & \text{if } v_0 < v_1 \\ e_0 = -1, e_1 = 3 & \text{if } v_0 > v_1 \end{cases} \tag{2}$$

with N_0 being the number of items collected from source 0 and N_1 the number of items collected from source 1.

Swarm composition The swarm comprises 30 homogeneous robots. The robots are equipped with the following sensors and actuators:

- **omnidirectional camera** to detect and distinguish between the sources (marked with LEDs);
- **light** to perceive the light above the nest;
- **ground sensor** 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 1 then to source 0, the item collected at source 1 will be replaced by an item from source 0.
- The color used for the LED on top of the cylindrical objects that are placed in the middle of the sources is independent of the energy level of the source. The LEDs colors are fixed once and for all experiments.
- We assume that the robots have such a limited memory that they only can memorize the quality of the last visited source. Each robot that estimates the quality of a source must overwrite the previous quality estimate (if any) that it stored. This means that a single robot cannot compare directly the quality of the sources. Note that this constraint must not be violated in order to obtain a sufficient solution.

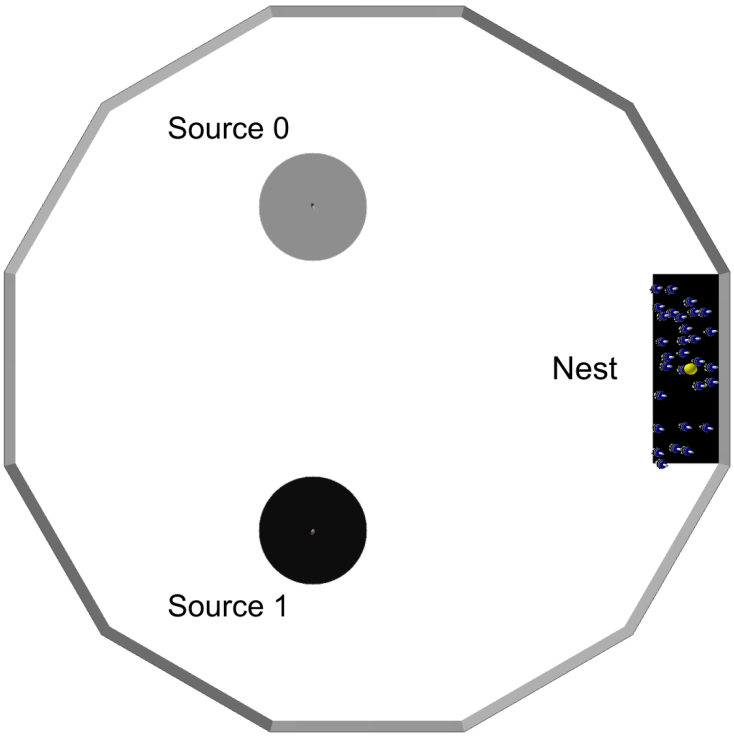

Figure 1: A top view of the environment. The nest is black and a light, visible from everywhere in the arena, is placed on top of it. Each source has a different quality which may vary in each experiment and is determined by the ground color. In this case, the best source is source 1 with $v_1 = 0.0519$.

2.3.2 Goals

The goal of this project is to design, implement, test, and compare **two robot controllers** that try to maximize the energy value collected at the nest.

The two controllers should differ from one another for the use of a *path formation* strategy. Path formation has the goal of assisting the navigation of robots between the nest and the sources. In practice, the implementation of the controllers can be divided in two steps: (1) the student should focus on the implementation of a solution in which the robots collectively decide which source is the best and collect food items from that source; (2) in a second controller, the student should add a path formation strategy to the collective decision making strategy developed in (1) in order for the robots to forage more effectively.

Remember to follow the principles of swarm robotics and apply what you learned during the course. Concerning the decision making strategy, solutions in which robots choose individually the best source to forage by comparing the quality of the two sources will be considered not sufficient.

ARGoS is configured to automatically dump data on a file whose name can

be changed in the .argos experiment configuration. 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 0
- The number of items collected from source 1
- 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 0
- Final number of items collected from source 1
- 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. Analyze, in particular, the effect that the path formation strategy has on the performance of your controller: does it improve the performance? Under which conditions? How could you improve it?

Be aware that, for the project evaluation, **the analysis is as important as the implementation**. A project presenting a controller that is capable of selecting the optimal source wonderfully will be evaluated poorly if the analysis part is missing or limited.

Setting up the code

- Download the experiment files: SR_Project_H414.tar from http://iridia.ulb.ac.be/~lgarattoni/h-414/SR_Project_H414.tar.

- Unpack the archive and compile the code:
  - $ tar xvf SR_Project_H414.tar # Unpacking
  - $ cd SR_Project_H414 # Enter the directory
  - $ mkdir build # Creating build dir
  - $ cd build # Entering build dir
  - $ cmake -DCMAKE_BUILD_TYPE=Release ../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=/path/to/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 /path/to/SR_Project_H414 # Make sure you are in the right directory
  - $ argos3 -c foraging.xml # 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 48 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 51 of the same file. Just comment line 48, uncomment line 51 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 6-8 pages structured as follows:

- Main idea of your approaches.
- Structure of your solutions (the state machines).
- Analysis and comparisons of the results.

3 Contacts

Marco Dorigo	mdorigo@ulb.ac.be	for general questions
Mauro Birattari	mbiro@ulb.ac.be	for general questions
Leslie Pérez Cáceres	leslie.perez.caceres@ulb.ac.be	for PSO
Hayfa Hammami	haifa.hammami@ulb.ac.be	for PSO
Lorenzo Garattoni	lgaratto@ulb.ac.be	for swarm robotics
Antoine Ligot	aligot@ulb.ac.be	for swarm robotics

References

[1] Competition on real-parameter single objective optimization. congress on evolutionary computation 2015 (cec 2015). http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2015/CEC2015.htm. Accessed: 14-07-2017.

[2] 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.

[3] P. N. Suganthan J. J. Liang, B. Y. Qu and Q. Chen. Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization. *Technical Report*, November 2014.

[4] Manuel López-Ibá nez, 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.