University of Aberdeen

School of Natural and Computing Sciences

Department of Computing Science

MSc in Artificial Intelligence

2020 – 2021

CS551K – Software Agents and Multi-Agent Systems

Assessment Item 3 – Individual Coursework (no groupwork)

This assessment contributes with 50% of the overall mark for the module

Learning Outcomes

This assessment, through its various tasks, has the following learning outcomes

- Knowledge and understanding of specific topics covered in the course
- Communication skills
- Problem solving
- Critical thinking and evaluation

Plagiarism and Collusion

This is an *individual assessment*, not a team-based one; its purpose is to assess your knowledge, not your ability to find answers on the Internet. Submissions may be checked for similarity and investigated if there is evidence of collusion (i.e., students using materials from one another) or plagiarism (i.e., answers using materials taken from someone or somewhere else without giving proper credit or acknowledgement). You must appropriately cite all materials you have used in your work. Plagiarism and collusion are serious issues and will not be tolerated. It is your responsibility to familiarise yourself with the University's code of practice on Student Discipline (https://tinyurl.com/y92xgkq6). Please also read the following information provided by the university: https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

Timeline and Lateness Penalties

This assessment timeline is as follows:

- 25/01/2021– Specification made available in MyAberdeen
- 14/02/2021 – Submission deadline (23:59)

Lateness penalties apply to this assessment. Please familiarise yourself with them. If in doubt, ask the course organiser or anyone from the team.

Assessment Specification

Provide answers to the items below clearly indicating in your submission what each part refers to. If we cannot work out which item that parts of your submission refer to then you may miss marks.

1. (**Combining CNP and Social Choice**) Your task is to design, implement and evaluate a simple multi-agent system (MAS) combining the Contract Net Protocol (CNP) with social choice theory (i.e., voting procedures). Your MAS should work as follows:
   - The initiator has a set of tasks it needs to be done. It interacts with participants using the CNP, but with a difference: rather than having each participant making an independent

proposal (and the initiator selecting one of them), we will have the participants collectively agreeing on which proposal should be made, that is, the participants use voting procedures (see item iii. below) to decide on which proposal the initiator should be provided.

- When a participant receives a call-for-proposal (with a task), it prepares a proposal as a triple ⟨*task*, *cost*, *days*⟩ establishing that it can perform *task* at *cost* in a number of *days*. For instance, ⟨t_1, 100, 5⟩ establishes that the participant can carry out task t_1, charging £100 and the task will be done within 5 days. If the agent is not able to carry out a task *t*, then its response to the call-for-proposals will be "refuse". You should also assume that all participants will reply (either with a proposal or a refusal). Your participant should have as a parameter a set of ready-made proposals ⟨*task*, *cost*, *days*⟩ it uses in response to the call-for-proposals.
- The initiator receives the responses (some are proposals, and some are refusals) from all participants and gathers all proposals for voting. It interacts with all participants providing them with all proposals and receives from each participant its preferences. After the initiator receives all preferences, then it computes the winner using a social choice function. The initiator then informs the winner and the losers.
- The participants provide their preferences based on utility functions which depend on the values of the cost and days (see item d below).

Your solution to this item should have the following components clearly indicated in your submission (the documents could be one single file but each item should be clearly indicated in it):

a) *Provide the protocol* as an AUML diagram. Solutions to this item should be a PDF document of at most 2 pages. (**2 marks**)

b) *Provide the pseudo-code for the initiator*. Your solution must clearly indicate any input parameters required. Solutions to this item should be a PDF document of at most 2 pages. (**4 marks**)

c) *Provide the pseudo-code for the participants*. Your solution must clearly indicate any input parameters required. Solutions to this item should be a PDF document of at most 2 pages. (**4 marks**)

d) *Implement a multi-agent system based on your answer to items a-c above*. Your simulation should have 12 participants or more (and one initiator). Your solution to this item must use Python as a programming language; implementations in other programming languages will merit 0 marks. Solutions to this item should be Python source code and instructions on how to run/execute it. (**8 marks**)

e) *Evaluate your system using runoff and Borda count as the social choice functions* (which you should implement as part of item d). You should use a population of participants in which 25% prefer a higher cost over the number of days, 25% prefer a higher number of days over the cost, 25% use a weighted function (which you must design/define) combining cost and number of days, and 25% have random preferences. Your evaluation should measure any advantage/difference between the two social choice functions (you should define these measures/metrics). Solutions to this item should be the code for experiments, instructions to run the code, and a PDF document of at most 3 pages (1,500 words, approximately) explaining the evaluation. (**7 marks**)

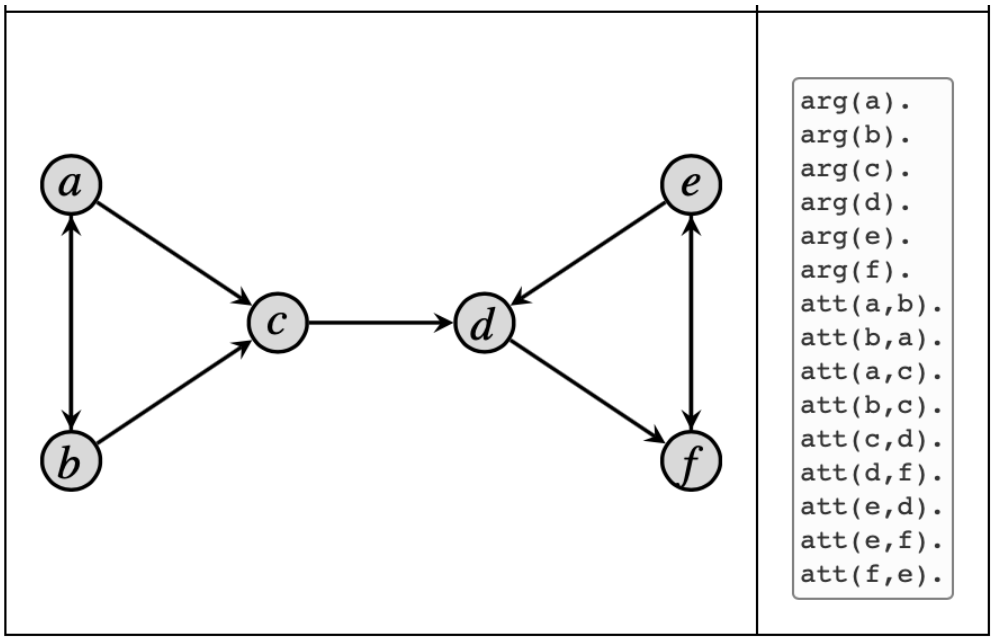
Hints and assumptions:

- Please refer to FIPA's (standard) specification of the Contract Net Protocol, available at http://www.fipa.org/specs/fipa00029/SC00029H.html.
- Consider as starting points the pseudo-code provided as answers to the second daily class test.
- Assume that all agents will behave exactly as they are programmed, that is, there will be no faults or crashes with any agent. All agents will reply to messages and no messages will be lost or corrupted/tampered with.

- Your implementation should *simulate* the agent interactions, that is, there is no requirement to actually have message passing or distributed programming. Agents can be, for instance, objects of a class (with specific parameters) with methods that are invoked by a main controller/loop of the system. This should reduce the complexity of the implementation and also provide more control for the experiments.
- The main controller of the system establishes the order in which the agents should interact. This order could be, for instance, round-robin. This is another simplification and it is acceptable because the system does not depend on agents being faster than one another.

2. (**Formal Argumentation**) In this part of the assessment, we will create an end-to-end system that can parse files and reason with arguments. You will need to provide a report (**maximum 4 pages**) containing the answers to each of the items below. You will need to clearly indicate in your report what each part refers to. If we cannot work out which item that parts of your submission refer to then you may miss marks. The code files for each question should also be attached to your submission.

   The ASPARTIX format allows one to represent Abstract Argumentation Frameworks using a syntax with only two predicates ("**`arg`** "and "**`att`***")*. In the example below, we show an argumentation graph with the corresponding ASPARTIX input. The statement "**`arg(a)`***"* declares the argument "**`a`**" whereas "**`att(a,b)`***"* declares that argument "**`a`**" attacks argument "**`b`**". Each statement must end with a period "**`.`**" and argument names should only be composed of alphanumeric characters "**`a`**" to "**`z`**", "**`A`**" to "**`Z`**", and "**`0`**" to "**`9`**".



   a. Design a solution to parse ASPARTIX files and provide the corresponding pseudo-code/code-snippets as well as explanation on how your solution works in the report.
   (2.5 marks)

   b. Implement a Python program **`aspartix_parser.py`** containing a function **`read_file`** that will take as input the path to a file in the ASPARTIX format and return a pair consisting of a set of strings (corresponding to arguments) and a set of pairs (corresponding to attacks) with the first argument assumed to attack the second. The **`read_file`** function should also print the number of arguments and attacks in your terminal. The box below shows the expected output of the **`read_file`** function on our example.

({"a", "b", "c", "d", "e", "f"}, {("a", "b"), ("b","a"), ("a","c"), ("b","c"), ("c", "d"), ("d","f"), ...})

There are 6 arguments and 9 attacks.

You can use existing Python libraries such as the Regular Expression library or the Lark toolkit. In your report, show the output of your implementation on an input sample with 10 arguments and 5 attacks. **(2.5 marks)**

c. Design and implement solutions to compute the maximal (for set inclusion) conflict-free, admissible, preferred, and stable extensions. To answer this question, you will need to provide the corresponding pseudo-code/code-snippets as well as explanation on how your solutions work in the report. **(5 marks)**

d. Implement a Python program **`semantics.py`** that returns the maximal (for set inclusion) conflict-free, admissible, preferred, and stable semantics for a given argumentation graph represented as a file in the ASPARTIX format. Your implementation should be run from the command line as "**`python3 semantics.py example.apx`**" in the terminal. If, **`example.apx`** is the ASPARTIX file containing our argumentation graph above, your program should output:

```
Maximal conflict-free extensions:

[
{"a", "d"},
{"a", "e"},
{"a", "f"},
{"b", "d"},
{"b", "e"},
{"b", "f"},
{"c", "e"},
{"c", "f"}
]

Admissible extensions:

[
{},
{"a"},
{"a", "e"},
{"b"},
{"b", "e"},
{"e"}
]

Preferred extensions:

[
{"a", "e"},
{"b", "e"}
]

Stable extensions:

[
{"a", "e"},
{"b", "e"}
]
```

Moreover, the arguments in the extensions and the extensions should be printed using the lexicographical order (`0-9` symbols appear before `a–z` symbols which in turn appear before `A–Z` symbols). For instance, the set {“10”, “9a”, “a1”, “b2”, “A1”} follows the lexicographical order whereas the set {“9a”, “10”, “A1”, “b2”, “a1”} does not. In the example of item 1 above, {“e”, “a”} is a preferred extension and should be printed as {“a”, “e”}. Please note that although you are not allowed to use existing code/libraries for this question, you can make use of the **`read_file`** function implemented in item 1. **(4.5 marks)**

e. In your report, show the output of your implementation on the five test files (test1.apx to test5.apx) provided in MyAberdeen. **(3 marks)**

f. Given an argumentation graph, a modification consists of removing an existing attack or adding a new attack between arguments. Design a solution that will take as input an

argumentation graph and a set of arguments X and output a new graph with the least number of modifications such that X is a preferred extension for the new graph. To answer this question, you will need to provide the corresponding pseudo-code/code-snippets as well as explanations in the report. **(3.5 marks)**

g. Implement a Python program **`dynamic_AF.py`** containing a function **`enforce_extension`** that will take as input both the path to a file in the ASPARTIX format and a path to a file with a set of arguments X and output a new graph with the least number of modifications such that X is a preferred extension. In your report, show the output of your implementation on the inputs below. **(4 marks)**

File	X
test1.apx	{"a","b","g","h"}
test1.apx	{"a","b","d","f","h"}

Further Marking Criteria

- Depth and breadth of knowledge
- Technical details of formalisation, implementation and pseudo-code
- Communication skills (clear, technical contents and sound reasoning)
- Structure of document

Submission Instructions

You should submit a single PDF file with your answers to the items above and upload it onto MyAberdeen by the deadline. Your PDF file should be named "CS551K-ASMNT3-YourSurname-YourName-YourIDNo". For instance, "CS551K-ASMNT3-Smith-John-999999.pdf", where 999999 is your student ID. Please try to make your submission file less than 10MB as you may have issues uploading large files onto MyAberdeen.

If you have any questions related to this assessment, please address them to the teaching team Wamberto Vasconcelos, w.w.vasconcelos@abdn.ac.uk and Bruno Yun, bruno.yun@abdn.ac.uk.