CAPSTONE PROJECT DESCRIPTION

I. KEY INFORMATION

Groups of students: 5 students / group / project are allowed, but students will get individual grades.

Please upload the group information as required on Teams (instructions will be posted on Teams a bit later), before the 10th of October 2021, 23h59.

All students in a group should be able to explain their subject clearly, and to reply to any question about the work during the presentation.

Deadline for topic submission:

- 17th of October, 2021, 23h59
- **Any delay will be penalized with minus points.**

Discussions about project advances (students + teachers + possibly, TA):

- at the end of every lecture session, starting from the 20th of October

- if needed, some sessions might be mostly focused on that

Deadline for final project submission:

- December 26th, 2021, 23h59
- **Any delay will be penalized with minus points.**

To be submitted on December 26th (more details / guidelines hereafter):

- 1 report (PDF file not exceeding 15 pages)
- 1 .zip file containing all the code (everything in Python)
- 1 demo: video (screenshot) of the execution of your code in a Python notebook (*e.g.* Jupyter)
- 1 PDF file for presentation. After the 26th of December, no more changes should be made to the slides, whatever the date/time of your presentation (even if you present only in January). This is to ensure fairness among students, so that all have the same deadline. Late submissions / changes are still possible, but they will be penalized by minus points.

Topics

There are 2 types of topics and each group of students (5 students) chooses among 1 of these two topics. The two topics are programming subjects about: problem solving by searching, or model-based reflex agents.

Hereafter a list of examples of subjects that students might choose from.

Students can also propose their own subjects. If they do so, and their subject is sound, then they will get up to 1 bonus point (depending on the level of difficulty of their subject and their ability to solve it).

1. ***Problem solving by searching***

For this topic, students will select and implement some of the algorithms that are in Chapter 3 of the course, or their variants.

More precisely, depending on their problem, students will select and implement at least 2 algorithms among:

- basic search algorithms and informed search strategies for “non-adversarial” search problems

- adversarial search strategies algorithms

Then, the students will evaluate their effectiveness and efficiency (both in theory and in practice).

In the report, students should explain in details their choices in terms of algorithms:

- Why did they choose to implement algorithm A and algorithm B, but not algorithm C?

    - In other words, why did they consider that algorithms A and B better fit the specificities of the problem at hand than algorithm C?

Students should also analyze in-depth the results they obtain:

- *e.g.* why did algorithm A perform better than algorithm B eventually?

- if possible, give a time complexity comparison of the different algorithms

    - if you can, theoretically (in terms of number of operations)

    - experimentally (by giving the computing time and the characteristics of the processor used)

Examples of subjects are:

1.1. Maze Solver:

Write a program to automatically generate, and solve a maze. Each maze is a matrix with size ***n x m***, each matrix element being a Boolean variable, with values:

- 0 is clear path, or
- 1 is wall

The starting point and finish points are randomly initialized among the clear path cells: their coordinates are denoted as (x_s, y_s) and (x_f, y_f), with (0,0) being the top-left corner of the matrix.

We chose to fix $n = m = 10$.

The program will have several outputs:

- Time complexity (number of nodes expanded in order to solve the maze)
- Space complexity (number of nodes kept in memory)
- The path used to solve the maze (solution) or, if the maze is unsolvable, print "Unsolvable"

Multiple mazes (at least 50) should be generated and solved (or not, if they were not solvable). The above-mentioned indicators must therefore be aggregated (averaged for instance) before being analyzed in the report.

1.2. Route planning

Write a program to find the shortest route between two Vietnamese cities (*e.g.* Hanoi and HCMC). Similar to the Romanian route planning problem, the intelligent vehicle can only travel between 2 adjacent cities, and the objective is to minimize the number of kms between two cities. But, in this case, you'll have to generate yourself a map of Vietnam with at least the 22 cities cited in https://www.distantias.com/distance-calculator-vietnam.htm, and distances (in kms) between these cities.

The cities list and distance list between cities will then be read from a file having the following format

- 1st line: ***n, m*** (***n*** is the number of cities, ***m*** is number of city-pairs with road between them)
- Next ***n*** lines: city names
- Next ***m*** lines: <city_1_names> <city_2_names> <distance between 2 cities>
- Next 2 lines: names of 2 cities that need to plan the route

The starting city and ending city will be fixed randomly. We call **trajectory** the path between the starting city and the ending city.

Beware: NOT ALL CITIES ARE ADJACENT! For instance, in order to go from Hanoi to Vinh, one must go through Thanh Hóa. Therefore, Thanh Hóa and Hanoi are adjacent, Thanh Hóa and Vinh are adjacent, but Hanoi and Vinh are not adjacent (not in the *m* pairs)!

The program will have several outputs

- Time complexity (number of nodes expanded in order to solve the route planning problem)
- Space complexity (number of nodes kept in memory)
- The path used to solve the route planning problem (solution) or, if there is no possible path between these two cities, print “Impossible”
- The cumulated number of km of the solution (if any)

Multiple trajectories (at least 50) should be generated and solved (or not, if they were not solvable). The above-mentioned indicators must therefore be aggregated (averaged for instance) before being analyzed in the report.

1.3. 8-puzzle

Write a program to solve the 8-puzzle problem from the Chapter 3’s slides. The initial state, and goal states must be randomly generated. The puzzle is a 3x3 matrix, where 0 denotes blank tiles.
The program will have several outputs

- Time complexity (number of nodes expanded in order to solve the puzzle)
- Space complexity (number of nodes kept in memory)
- The sequence of moves from the blank tile to solve the puzzle (solution) or, if there is no possible solution, print “Unsolvable”.

Multiple puzzles (at least 50) should be generated and solved (or not, if they were not solvable). The above-mentioned indicators must therefore be aggregated (averaged for instance) before being analyzed in the report.

Bonus points: Try to increase the matrix size, and apply the same algorithms. Analyze the changes that occur when you increase the matrix size.

1.4. Extended wolf-goat-cabbage problem

The original wolf-goat-cabbage problem seen in Chapter 3, is extended as follows:

- On top of the goat, wolf, cabbage and shepherd, we have 2 new objects: a wooden stick and a fire torch.
- Without the shepherd, if the stick and the wolf are together, the stick beats the wolf.
- Without the shepherd, if the stick and the torch are together, the torch burns the stick.
- The shepherd now can bring 2 objects or animals across the river at each step.

The program will have several outputs

- Time complexity (number of nodes expanded in order to solve the problem)
- Space complexity (number of nodes kept in memory)
- The sequence of actions that the shepherd must take in order to bring all objects / animals on the other side of the river (solution) or, if your algorithm does not find any possible solution, print "Solution not found"

Bonus points: The initial state can be generated automatically (any object / animal / shepherd can initially be on either side of the river, but not in a conflicting state). Run the program 50+ times and automatically solve the problems (or not, if your algorithms do not find a solution). Compare the results of the algorithms you implemented on these different instances of the problem.

2. Intelligent agents

For this topic, students will have to implement a single intelligent agent, as seen in Chapter 2 of the course, and evaluate the effectiveness and efficiency (in practice) of the algorithm they proposed. In the report, students should explain in detail their algorithmic choices.

Intelligent agents that should be implemented in this capstone project can be either model-based reflex agents, goal-based agents or utility-based agents. Indeed, simple reflex-based agents are considered too simple, and learning-based agents, too complex, for this capstone project.

Two examples of agents that are of adequate complexity for this project are the following. You can also propose your own problem, and gain up to 1 bonus point if the problem you propose is complex enough, and you are able to build an algorithm to solve it.

2.1. **Intelligent vacuum cleaner for rich people**:

Let's say that the family who own this vacuum cleaner is very rich. They are so much rich that they have not only dust, but also jewels on their floor.

The vacuum cleaner should therefore have 2 functionalities: vacuum dust and pick up the jewels. The vacuum cleaner is working on an environment with a ***n x m*** matrix floor tiles with each tile can have the following values

- (0,0) for a clean tile
- (1,0) for a tile with dust
- (0,1) for a tile with a jewel
- (1,1) for a tile with a jewel, AND dust

If a piece of jewelry is on a dusty tile "(1,1)", then the cleaner must pick up the jewelry first, to avoid sucking it. Each piece of jewelry has a location (x_i, y_i), where (0,0) is the top-left conner of the matrix. The top-left corner of the matrix has the specificity of being initially clean (no dust, no jewel), and it contains a jewel box. Every time it picks up one jewel, the agent must bring the jewel back to the jewelry box at **(*0,0*)** before continuing sucking the dust or collecting more jewels.

The vacuum cleaner agent can perform the following actions:

- Move forward
- Turn left
- Turn right
- Vacuum dust
- Pick up jewel
- Put jewel in the jewel box

The program will have several outputs

- Number of steps needed for the agent to finish cleaning the floor
- The percept sequence and corresponding actions of the agent
- Execution time

You must propose an adequate type of agent (model-based, goal-based, utility-based), then an algorithm to set up the environment and program the agent's behavior, and justify your choices at each step.

Several initial environments (50+) must be generated randomly, with varying number of dirty tiles / tiles with jewels. The program should be run on each of them. The above-mentioned indicators must therefore be aggregated (averaged for instance) before being analyzed in the report.

Bonus point: Assume that the environment also has several obstacles on it. Adjust the agent (in terms of percepts / algorithm) so as to avoid the obstacles (when possible).

2.2. **Pacman**

The main stage is a 10*10 matrix, with a fixed amount of coins

• Number 0 indicates a "clear" cell, with no wall or coin

• Number 1 indicates a wall

• Number 2 indicates a cell containing a coin, but no wall

• Number 3 indicates Pacman (only one in the main stage at each time)

• Number 4 indicates the ghost (for simplicity, there is only one ghost in the map)

The Pacman agent can perform the following actions:

- Move forward
- Turn left
- Turn right
- Pick up coin (pacman automatically collect a coin when it move to the coin's cell, doesn't cost another step to collect it)
- Die (if killed by the ghost)

The program will have several outputs

- Number of steps needed for the agent to finish collecting all coins while avoiding the ghost
- The percept sequence and corresponding actions of the agent
- Execution time

Some constraints about the environment:

- make sure that every coin is reachable by Pacman (depends on Pacman's initial position, the coins positions, and the walls positions

- the movements of the ghost are pre-defined (for instance, from left to right until it reaches a wall and then bounces back), each time pacman makes a move, the ghost also makes a move (according to its pre-defined behavior or "pattern", of course).

- therefore, the movements of the ghost do not depend on Pacman's actions

- At each time step, Pacman knows the position of the ghost, and its next step (because Pacman knows everything about the movement "rules" for the ghost.

You must propose an adequate type of agent (model-based, goal-based, utility-based), then an algorithm to set up the environment and program the agent's behavior, and justify your choices at each step.

Several initial environments (50+) must be generated randomly, with a fixed number of coins and walls. The program should be run on each of them. The above-mentioned indicators must therefore be aggregated (averaged for instance) before being analyzed in the report.

II. Requirements

NO PLAGIARISM!!! Nor for the code, nor for the report…

1. The whole project should be implemented from scratch in Python (basic libraries for data manipulation and visualization, such as pandas and matplotlib, are allowed)

2. **Project report:**

   Include specific details about the project and insights about its results.

   The project report should not exceed 15 pages.

   Reports MUST contain a list all the tasks in the project with the member(s) in charge.

   If there are 2 members that contribute to the same tasks, show the percentages of contribution for each member.

   No more than 2 members / task **for programming tasks**.

In short, the report must contain:

- The complete list of tasks
- For each task, each student's contribution to this task (with percentages that sum up to 100%/task, if more than 1 student/task)
- Problem analysis (purpose of the problem, problem formulation, choice of the algorithms to be used)
- Algorithms used (pseudocode or diagram)
- Results and result analysis
- Reference documents

More specifically, at the end of this file, you will find **MORE DETAILED GUIDELINES ABOUT WHAT TO WRITE IN YOUR REPORT.**

3. Demo video:
   - Video (screenshot) of the execution of your code in a Python notebook (tutorial for the ones who don't know what is a Python notebook: https://www.dataquest.io/blog/jupyter-notebook-tutorial/)

4. Oral presentation (15 mins maximum = 10m for presentation + 5m for QnA)

   The presentation should focus solely on the core of the topic, solution/technique, analysis of the results obtained. Presentations can be conceived as a (visual) summary of the report.

   ALL students in the group must present together.

   Presentation = PDF file.

To avoid any format problem, the presentation files should be PDF. If you need to add animation, you can use multiple slides with the same slide number. The presentation PDF file should contain **at most** 10 slides of **contents** (excluding animations). More than 10 slides of content will **not** be allowed / more than 10 minutes presentation will **not** be allowed).

The first slide of the presentation MUST contain the (recent) pictures of the students in the group, with, underneath of each picture, the corresponding student's name/number.

III. Topic description assignment (deadline 17th of October, 2021)

Each group must create 1 PDF file **named with their group #**, and 1 student / group must upload on the dedicated assignment on Teams this PDF file, with the names, student numbers, and e-mail addresses of the students involved, and a description of your problem.

This problem can be either one of the example problems listed above (then you just need to put the title in the PDF file), or your own problem. If you come up with your own problem, then you need to add a 1-page description of your problem in the PDF file. Your own problem can also be a variation of the above-mentioned example problems.

The advantage with coming up with your own problem is that, if it is complex enough AND you are able to solve it, you might get up to 1 bonus point.

But, this problem should not be taken from the internet, and in particular no already implemented solution should be available on the web.

For the groups who propose their own problems, me and my TA will give you feedback on your proposal, and we will iterate as many times as needed until each group has a suitable subject (hard deadlines will be given later on to each group depending on their specificities).

IV. Final assignment (deadline 26th of December, 2021)

Each group must create 1 directory **named with their group #**, and 1 student / group must upload on the dedicated assignment on Teams:

- Report: PDF file, no more than 15 pages (reference pages / first page / table of contents are NOT counted in these 15 pages).
- Code: put all your python files in a single .zip file, with a readme.txt file explaining your code (for better readability / re-usability, please break your code into parts and explain the functionalities of each part).
- Video demo of your Python code running in a notebook: only most standard formats are allowed: .mpeg4, .wav...
- Presentation: PDF file, no more than 10 slides. Do not put too much text in each slide! (a drawing is sometimes worth 1000 words...) The first slide of the presentation MUST

contain the (recent) pictures of the students in the group, with, underneath of each picture, the corresponding student's name/number.

No more changes are allowed after the deadline (26th of December, 23h59), including in the presentation slides. Any delay will be penalized with minus points.

No plagiarism is allowed. If you copy-paste /paraphrase from other groups or from the internet without citing your source, you will get A LOT OF minus points.

V. MORE DETAILED GUIDELINES ABOUT WHAT TO WRITE IN YOUR REPORT.

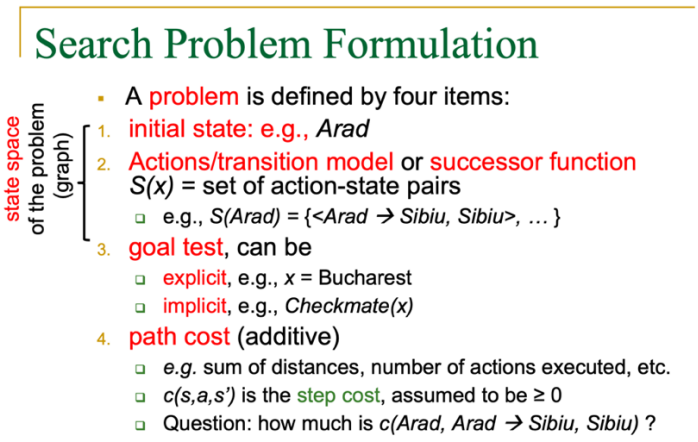
N.B. The following are general guidelines, that might not apply perfectly to your specific problem, especially that many groups chose different kinds of problems. But, in most cases, these guidelines apply. And, even when they are not 100% adapted, they will give you more precise ideas of the general focus of your report. This focus should be **analytic** (analyzing the problem, explaining your choices, analyzing the results of different algorithms). The focus should **not** be on the programming part.

1. Presentation of the subject

- Detailed description of your subject (in English, including graphics to illustrate your problem if needed)

2. Description of the problem

- Detailed description purpose of the problem (using the specific vocabulary from the course: what kind of problem are you dealing with? What are the specificities of your problem (*e.g.* in terms of branching factor…)?)
- Problem formulation:
    - If you work on a search problem, then the problem formulation should include the following information:



Depending on your problem, if your problem can be formulated using agents, please add the PEAS formulation and the type of environment.

- If you work on an intelligent agent, then the problem formulation should focus on:
  - the PEAS formulation
  - the types of environments (observable, partially observable, etc)
  - the most adequate type of agent (model-based, goal-based, utility-based): see chapter 2.

3. Explaining your choice of algorithms to be used for solving the problem, and their parameterization

For solving the problem, you must apply at least 2 algorithms (more than 2 is welcome, less than 2 should be discussed with Prof. Visani beforehand).

- In most cases, these algorithms should be from the algorithms that we extensively studied during the course (*e.g.* breadth-search, depth-search, uniform-depth search, A* search…) or, exceptionally, techniques that we studied less extensively in this course (when Prof. Visani authorized you to).
  - If you apply only algorithms that we extensively studied in the courses, then you don't need to explain the algorithm itself, but you **do have** to explain how you applied it to your problem (for instance, explain and **justify** the choice of your heuristic for A* search).
  - If you apply techniques that we studied less extensively in this course, then you **have** to explain the algorithm itself (using pseudo-code /diagrams, but also a description of its main principles in English), and how you applied it to your problem.

- In all cases, you have to explain your choice of algorithms: for instance, let's say that you chose to apply algorithms A and B, but not algorithm C. Then, you have to explain why you expect algorithms A and B to work fine on your problem, when on the other hand, you expect that algorithm C should give poorer results / would be too difficult / complex to apply on your problem. In most cases, your choice should be linked to the specificities of your problem, such as a high/small branching factor for instance.

4. Implementing the algorithms to be used for solving the problem

Here, you will **not** make a copy-paste of your code. Instead, you will just point out the main difficulties you had to face for implementation, and explain how you solved these difficulties.

If you need to set some parameters for your algorithm or performance measure (for instance the costs associated to some actions), then you must clearly justify which values you chose for these parameters: according to the specificities of the problem, why do you think that these

values are good? How did you choose this values: did you try several sets of parameters and compare their performances in order to choose the values, or did you just set the parameter values *a priori*?

5. Comparing the results of the algorithms used for solving the problem

a. Providing quantitative performance indicators

In most cases, you will have run the algorithms on multiple instances (generally hundreds) of random instances of your problem. Then, you can compare the results of the algorithms you chose to use for solving the problem according to different angles:

- %age of the times where the algorithm successfully solved the problem, *vs.* total number of problem instances
- Average time and space complexities observed in practice (for each problem "size", if this applies to your problem).

This comparison can be done in a synthetic way using tables.

b. Explaining these results

Once you have presented the quantitative performance comparison, you can try to explain the differences between algorithms that you obtained, using sentences such as (for instance):

- Algorithm A is (on average) more complete than algorithm B (solves the problem in more instances of the problem), because [your explanation – might be linked to the specificities of your problem, or to the algorithms themselves]…
- Algorithm A has (on average) a smaller time complexity than algorithm B, but a higher space complexity than algorithm B, because [your explanation – might be linked to the specificities of your problem, or to the algorithms themselves]…

6. Conclusion and possible extensions

What are your (analytic) conclusions about your work? What did you learn by achieving this project? If you had more time, what would you have wished to do, as possible extensions of the work you already achieved? What would be the main difficulties of these extensions? Any idea about how to overcome those (if not, it's fine, just list the difficulties).

7. List of tasks

Provide a list of the tasks that you performed in order to solve the problem. Distinguish between **programming tasks** and **analytic tasks**.

- Examples of programming tasks: running random instances of the problem; implementing algorithm A, etc.
- Examples of analytic tasks: proposing our subject, selecting the algorithms to be used to solve this problem, writing parts 1/2/3/4 of the report, writing parts 1/2/3/4 of the presentation, etc.

For each task, provide a detailed contribution of each member to this task:

- e.g.: Programming Task 1- running random instances of the problem: member1 Mr X 40% ; member 2 Mrs Y 60% **(no more than 2 members /task for programming task**s)...).
- e.g.: Analytic Task 1– proposing our subject: member1 Mr X 40% ; member 2 Mrs Y 30% ; member 2 Mr Z 30% **(can be more than 2 members /task for analytic task**s)...).

8. **List of bibliographic references**

List of books / web resources that you used (with citations in the main text), using the Vancouver system explained in: https://en.wikipedia.org/wiki/Citation.