COMP2009 Intelligent Agents

2021 Assignment

General Requirements

Submission

You will submit (a) a PDF document that contains answers to the theory questions of the assignment as well as any notes or discussion needed relating to the practical part and (b) an archive (such as a ZIP file) that contains your code for the practical part. Your PDF document must start with the cover sheet (aka Declaration of Originality) that can be found in the same folder as this assignment.

The PDF document will be submitted via the TurnItIn link on Blackboard and the archive will be submitted via the Blackboard assignment link; both are clearly marked. If you don't submit your PDF via TurnItIn we cannot mark it for policy reasons and we also can't submit it through there on your behalf. Submitting your archive via the TurnItIn link (if it's even possible) may result in your archive not being usable, resulting in a mark of zero for this part. If you have any questions about what goes where, please ask the UC well before the deadline.

Code

You are asked to write several pieces of code. You will submit an archive that contains only text files (no object or other compiled files) and it must be possible to compile all files in the archive with the 'make' command. For C or C++ this will mean creating a Makefile and for other languages this may mean writing a bash or similar script that calls the building mechanisms (such as *javac* for Java).

Once any compilation/linking/etc has been done using 'make', your software needs to be able to be called from the command line to perform its intended function. This will be done by a script, so if your program does not have the right name or accept the correct command line arguments the script will not function. At that point you will lose some marks and your code will be executed by hand.

Each code section has execution marks, which are between ¼ and $^1/_3$ of the marks for that question. You will lose around half of those marks if your code does not execute using the script and the rest if your code does not execute at all. The marker(s) will not be able to spend time trying to get your code working due to time constraints. All code will be tested on the computers in the lab used for tutorials in the unit.

Academic Integrity

This assignment tests particular skills and particular thought processes. Each question will have a focus and your answer to that must be entirely yours. You will need to make other decisions and use other skills for some of the questions, but if they are not a focus of the question you may discuss these with others, use code from legal sources and otherwise obtain information as you see fit as long as you reference the source(s) appropriately.

For example, if asked to write code that loads in a file, implements an algorithm and displays the result with the focus being on the algorithm, you should feel free to discuss the choice of data structures with others, use public domain code for loading the file or displaying the results or similar. You should not discuss the details of the algorithm implementation with anyone, and all that code must be entirely your own.

If you discuss concepts with others, you can simply state this in your documentation. If you are using sources you should reference this in a consistent manner and all code used must be referenced in a

way that follows our Code Referencing Standard (which is available in the same Blackboard folder as this assignment).

Late Submission

Lateness will be assessed as via the Unit Outline. If your work is late (is submitted past the hour) then your work will be penalized by the amount stated. Noted that this is a deduction of total marks allocated to the task, not the marks you achieve. So, if you submit 5 minutes late for an assessment worth 100 marks you will lose 5% of those 100 marks (or 5 marks) irrespective of your score in that task. If this penalty would reduce your mark to a negative value, zero will be used instead.

If you do submit late, send an e-mail to the UC after your submission or they may not notice. In that case you may not get your marks until much later.

If your work is impacted by factors beyond your control that were impossible to plan for, you may apply for an extension using the online form. (Talk to Connect if you cannot find it.) If you submit this after the submission time is over, you will also need to justify why this lateness is necessary. You will need to provide evidence of any claims – for example, a medical certificate for illness or a note from a counsellor if impacted by stress or other mental health challenges.

Note that if you have a Curtin Access Plan (CAP) that allows you to make late submissions, please ensure that you have sent this to the UC. The requirement is that this happens at the start of the semester but if it happens later the CAP will probably be accepted if you have sufficient grounds. The UC will reply to you, telling you what parts of the CAP will apply to this unit, and specifically this assignment.

Extensions cannot be given past the release date of the marks, irrespective of grounds (including the CAP) but exemptions will be considered in cases where this is clearly warranted. If you do get an extension this may delay your work being marked and returned.

Assignment Questions

Each of the following questions has a written component or a code component, or sometimes both. Answers such as diagram and explanations are part of the theory and need to be submitted in the PDF and only code should be stored in the archive. You are encouraged to comment your code but any specific answers to questions must be in the PDF document submitted through TurnItIn. Code comments will be taken as explanations of the code (include referencing) and will not be considered towards answers to these questions.

Q1: Agent Design

Choose two of the agents or problems below. If it is a problem, the task will relate to an agent that you must design to solve this problem. If it is an agent, the task is to specify the design of an agent that fulfils the same task(s).

1. An agent that will take the part of the player in the game Lemmings.
2. Rio Tinto's Autonomous Haulage System technology.
3. Boeing's Airpower Teaming System.
4. The drones being used by the Indian army for border patrol.
5. The drones reported in the New Zealand Police's *Review of Emergent Technologies* report.
6. The implementation of Dynamic Difficulty Adjustment in Resident Evil 4.

For the chosen agent, give details of the environment that the agent will need to work in and the PEAS description of the agent. Explain your decisions, especially with regards to the performance measure.

You will also need to give an explanation of what you see the agent as needing to do, just in case your interpretation is different to that of the marker. If there is any interpretation involved (i.e., if something is not entirely certain) then you will need to explain your decision.

Focus: The agent design is the focus, including the environment and PEAS. You may discuss your interpretation of what the agent needs to do with others.

Q2: Uninformed Search

Write code that reads in a graph file (as per the specifications in Tutorial 3) and runs a Depth First Search on the graph. The search should start at the node with the lowest number (generally 0) and the goal is the node with the highest number. Your program should then display the search path taken or state that no solution was found. For the search path, display all of the nodes traversed (in order from the root down to the goal).

The program must be able to be run from the command line using *dfs* <*filename*> where <*filename*> is to be replaced by the name of the input file. For example, if the file is called 'graph1.nt' then this should be able to be read by typing *dfs graph1.nt*.

The program must allow an optional second argument; if an integer is specified then this will form a limit to the depth of the DFS. So, if the command is *dfs graph1.nt 4*, then the program should run DFS on the file to a depth of 4.

Focus: The coding is the focus. You should not show anyone your code, make it available to anyone or otherwise allow them to access it. Feel free to discuss general issues about implementing DFS but you may not discuss anything at the code level.

Q3: Informed Search

Write code that reads in a graph file (as per the specifications in Tutorial 4) and runs an A*search on the graph. The search should start at the node with the lowest number (generally 0) and the goal is the node with the heuristic value of zero. Your program should then display the search path taken and the cost of the path, or state that no solution was found. For the search path, display all of the nodes traversed (in order from the root down to the goal) showing the f-cost of each node.

The program must be able to be run from the command line using *astar* <*filename*> where <*filename*> is to be replaced by the name of the input file. For example, if the file is called 'graph1.nt' then this should be able to be read by typing *astar graph1.nt*.

Focus: The coding is the focus. You should not show anyone your code, make it available to anyone or otherwise allow them to access it. Feel free to discuss general issues about implementing DFS but you may not discuss anything at the code level.

Q4: Local Search

Write code that uses simulated annealing to attempt to solve to 16 Queens problem (i.e., the 8 Queens problem but with a 16x16 chess board and 16 queens). Recall that simulated annealing is basically hill climbing that includes randomness in the choice of the next state. At every step, you should display the step number, the current temperature and your current problem state and its "goodness". Recall that when the temperature drops to nothing then simulated annealing functions very much like hill climbing.

The program must be able to be run from the command line using *san* <*temperature*> <*step*> where <*temperature*> is to be replaced by an integer to be used as the initial temperature and <*step*> is an

integer by which the temperature is decreased every step. For example, a command of *san 100 1* would run your simulated annealing with a starting temperature of 100 with a decrease of 1 each step.

Simulated Annealing picks a random next state from the queue – you may make this truly random based on the current length of the queue. If the temperature check fails and that state is not used, it can be discarded, and another picked. If the queue becomes empty, the search stops. Obviously, this cannot happen if any of the next states have a higher goodness than the current one.

Hint: You will need to provide a way to measure the length of the queue and a way to pop a random state from it. You probably have not implemented either of those for hill climbing but should be able to use much of the rest of your code.

Focus: The coding is the focus. You should not show anyone your code, make it available to anyone or otherwise allow them to access it. Feel free to discuss general issues about implementing DFS but you may not discuss anything at the code level.

In addition, you should not discuss the following with others but may research efficient ways of doing this (and include appropriate references):

- How you are storing the state.
- The exact mechanism for calculating the goodness measure.
- The way in which you are generating the next states to be added to the queue.
- How you are implementing the randomness and choosing a random next state.
- How you are modifying the randomness based on the temperature.