Project Deliverable 4

Andrew Berry

November 17, 2016

I pledge my honor that I have neither given nor received any unauthorized aid on this work.

1 Achievements Since Demonstration

During the demonstration of my project, suggestions arose for addressing a potential memory leak in my software (or the various open-source software libraries I have included in my project). Suggestions for diagnosing this include limiting the number of agents in my MAS approach to the problem to an arbitrarily small number (i.e. 2 agents) and caching diagnostic output to write to a file instead of printing to the console immediately (for performance's sake). I was also advised to reorganize the agent graph to resemble a star formation rather than a linked list. Also mentioned was the possibility of running the simulation in an infinite loop in which a threshold is defined to determine the utility at which the program has "succeeded." Conversely, the infinite loop should exhibit delta improvement termination (meaning that if the utility does not improve by a sufficient amount over some number of time steps, the program should terminate with a "failure."

I have attempted to diagnose the memory leak in my project, so far unsuccessfully. Lowering the number of agents has proven to decrease the overall runtime of the program, but the issue of starting the program with no movement still occurred on the first attempted run of my program after restarting the SPADE server. I have tried both decreasing the number of agents in the system and decreasing the complexity of the random forests the agents use. I found a forum posting on Quora stating that the user was having memory issues with the random forest in scikit-learn, but his example used over 18,000 columns, which is likely the culprit in his case. Since I am only training on fewer than 10 metrics, I do not anticipate that the learning algorithm is necessarily the cause of my hang ups, but more investigation is needed still to determine the cause of the issue and resolve it. Caching output to a string and writing that string to a file hasn't proven fruitful in improving performance over writing directly to stdout.

Per recommendations from the demonstration meeting, I did try running my simulation on one of the Featheringill lab computers. Once I got the environment set up and ran the program, it crashed with a MemoryError. Upon searching the internet for instances of this error arising when using scikit-learn, I found some recommendations for better memory practices (deleting objects that are no longer needed, mainly). I have found some python packages online that are meant to be used for memory usage profiling. My plan is to use one of these over Thanksgiving break to try to more finely track down where my memory issues are arising.

I have begun a refactoring of the agent framework to accomodate a broadcast type of system rather than a linked-list style communication. That is, the current organization of my system has one agent that only sends data, one agent that only receives data, and all agents in between send and receive in sequence from the original sender to the final receiver. The new setup will have all agents but one sending communications to a universal listener, which must decide (using a random number and a probability distribution built from the parameters of the other agents) which "guess" it will choose for each record in the test set. This is currently a work in progress, with significant workload, since this possibility was not adequately accounted for in the original code.

The program in its current state runs through all agents in the line from beginning to end exactly once. When run with only a small number of agents, we are unable to tell whether any improvements occur as the communication and decision-making occurs among the agents. To address this, I intend to alter the system to repeat its execution with additional information to improve the classifications it makes, terminating after

it either reaches a sufficiently high accuracy in prediction or fails to make a marked improvement after a certain number of repetitions. These changes have yet to be implemented in my system.

2 Potential Issues Foreseen

The memory issues mentioned above can be of detriment to my ability to complete the project successfully. The ability to scale up the number of agents is important in being able to determine if the method fails to produce results on the whole, or if it just fails for small numbers of agents.

3 Distributed Planning

My project, in its current state, does not incorporate elements of distributed planning.

One way in which decentralized POMDPs (DEC-POMDPs) could be incorporated into this project is by having each agent choose a classification for a data element, the world state changes because of all these choices, a reward is assigned, each agent observes the reward in regard to their action and we repeat these steps. Some advantages of including this distributed planning approach include the fact that it still allows inter-agent communications (either explicitly or implicitly) and that it encourages agent cooperation, since there is only one reward function (meaning all agents work toward the same goal).

One major disadvantage of incorporating DEC-POMDPs into the project would be the time complexity (NEXP-hard). Since my project already has memory issues and takes a long time to run (even in very restricted circumstances), adding additional time complexity would not be desirable.

In order to incorporate DEC-POMDPs into my project, agents' actions (choices for classification) would need to be evaluated on a row-by-row basis, while I currently determine agents' rewards based on the accuracy of their classifications over the whole set.