David Amin (dma3fq), Rupali Vohra (rv5rr)

CS 4710: Homework 3

Negotiating Agents

Agent 1: NegotiatorOrder

The NegotiatorOrder attempts to figure out the optimal ordering such that both negotiators get the best utility possible.

**Agent 2:**   
  
Approach:  
For our second agent, we decided to develop an incrementally more elaborate system. The benefit here was twofold, we were easily able to expand from the different points of our development as ideas struck, and we had a series of working agents to test against, to see if our changes were helping us improve relative to them. Initially, we built an agent that was quite simple, it simply enumerated all possible permutations of the list, then categorized them by how much utility each one was worth, and then determined how many different levels of utility there were. We consider most of the problem from the perspective of moving “up” or “down” in these levels, which is roughly the same as making an offer with more or less utility respectively. To start, the agent had a minimum utility it needed to achieve. This minimum decreases as the rounds go on, and if at any point it receives an offer of greater utility than the one it is currently on, it accepts. Obviously, this isn’t very complicated, and it is very easy for a system to beat, by holding out and making offers that are bad for the agent until the very end, then making a single offer that is reasonable for our agent and great for theirs.

To improve on this, we decided to add a simple predictive scheme to the agent. The agent keeps track of what the opponent has done in their moves so far, how many times they’ve increased their own utility, how many times they have conceded utility, and how many times they haven’t changed. It uses this information to determine the likelihood of the opponent taking a loss to utility, which is assumed as being a sign of cooperation. Strictly speaking, it might not always be, but it is assumed for the purposes of the agent that any decrease in utility is a sign of good faith. If the opponent is likely to cooperate, our agent cooperates by taking a hit to utility in our offer. If their agent is likely to drive their utility higher, ours will match them by sending a deal with a higher utility for us. The benefit to this approach is that it won’t fall victim to the mistakes of the first approach, it will tend to be tough with tough opponents and cooperative with cooperative opponents.

Our final approach, what we actually submitted as Agent 2, is a slight but important refinement on the probabilistic technique of its predecessor. Rather than constantly moving in the direction the opponent is most likely to move, it moves randomly up, down, or not at all, with the same likelihood that the opponent will impact their own utility the same way. In this way, it doesn’t fall into lowering its minimum utility with a cooperating opponent, then continuing to do so while the opponent is uncharacteristically uncooperative. The agent might move up or down, or even stay at the same amount of utility, even if opponent is cooperating or stonewalling negotiations, which allows for some variability and makes our agent more resistant to changes in the behavior of the opponent. This is additionally beneficial because it makes it harder for an opponent to read the strategy of the agent and exploit it.   
  
Results:

We ran our agent against a few others that we had available, to see how it stood up to their strategies. To test, we ran it in 3 rounds of 10 iterations each as negotiator A, then the same as negotiator B. We did this for 3 different scenarios, but kept the scenario consistent for any set of 3 rounds. The results of these tests are listed below, with the table showing our agents utility, followed by a slash and then the opponent’s utility  
  
Against the fully random agent that was given:

|  |  |  |  |
| --- | --- | --- | --- |
| As A | Input Size 5 | Input Size 7 | Input Size 9 |
| Round 1 | 7.42/-.58 | 18.15/-3.85 | -9/-9 |
| Round 2 | 7.42/3.42 | -7/-7 | 23.46/7.46 |
| Round 3 | 5.42/5.42 | 16.15/-3.85 | 17.46/13.46 |
| Final | 20.25/8.25 | 27.3/-14.7 | 31.92/11.92 |

|  |  |  |  |
| --- | --- | --- | --- |
| As B | Input Size 5 | Input Size 7 | Input Size 9 |
| Round 1 | 5.42/5.42 | -7/-7 | 23.46/9.46 |
| Round 2 | 11.42/3.42 | 16.15/-3.85 | -9/-9 |
| Round 3 | -5/-5 | 16.15/-3.85 | 23.46/7.46 |
| Final | 13.83/3.83 | 25.3/-14.7 | 37.92/7.92 |

Against the agent that looks for a minimum utility decreasing over time:

|  |  |  |  |
| --- | --- | --- | --- |
| As A | Input Size 5 | Input Size 7 | Input Size 9 |
| Round 1 | 9.42/5.42 | 8.15/0.15 | -9/-9 |
| Round 2 | 3.42/5.42 | 4.15/4.15 | -9/-9 |
| Round 3 | 7.42/5.42 | 2.15/2.15 | 15.46/11.46 |
| Final | 20.25/16.25 | 14.45/6.45 | -2.54/-6.54 |

|  |  |  |  |
| --- | --- | --- | --- |
| As B | Input Size 5 | Input Size 7 | Input Size 9 |
| Round 1 | 7.42/3.42 | 8.15/2.15 | -9/-9 |
| Round 2 | 9.42/3.42 | 8.15/-1.85 | -9/-9 |
| Round 3 | 7.42/1.42 | 8.15/-1.85 | 9.46/-6.54 |
| Final | 24.25/8.25 | 18.45/4.45 | -8.54/-24.54 |

Against the agent that determines the opponent’s probable move and matches it:

|  |  |  |  |
| --- | --- | --- | --- |
| As A | Input Size 5 | Input Size 7 | Input Size 9 |
| Round 1 | 9.41/5.41 | 6.15/4.15 | -9/-9 |
| Round 2 | 5.41/7.41 | 10.15/2.15 | 11.46/13.46 |
| Round 3 | 5.41/7.41 | 8.15/2.15 | 11.46/13.46 |
| Final | 20.25/20.25 | 24.45/8.45 | 13.92/17.92 |

|  |  |  |  |
| --- | --- | --- | --- |
| As B | Input Size 5 | Input Size 7 | Input Size 9 |
| Round 1 | 3.41/7.41 | 10.15/-1.85 | 9.46/9.46 |
| Round 2 | 7.41/7.41 | 8.15/.15 | 11.46/9.46 |
| Round 3 | 5.41/3.41 | 2.15/4.15 | 13.46/15.46 |
| Final | 16.25/18.25 | 20.45/2.45 | 34.38/34.38 |

From the above, it is clear that

**Findings**:

Both of these agents made use of Python’s itertool’s library’s permutation method. Since finding permutations is an exponential function, we decided to see how big a difference the number of elements for which we attempted to find a permutation made. The result can be seen in this graph:

From the chart, it is clear that there is a jump at the 10 second mark. We attempted to find the time it would take for more elements, but unfortunately received a memory error. Given this finding, neither of the agents would work if given elements of size greater than 10.  
  
Future Improvements:

Based on the results of our tests, we had several ideas for how to expand on the agent that might help it beat certain opponents.