Reinforcement Learning for Abstract Argumentation: A Q-learning approach

Sultan Alahmari
University of York
Department of Computer Science
Deramore Lane, Heslington, York, YO10 5GH, UK
smsa500@york.ac.uk

Tommy Yuan
University of York
Department of Computer Science
Deramore Lane, Heslington, York, YO10 5GH, UK
tommy.yuan@york.ac.uk

Daniel Kudenko
University of York
Department of Computer Science
Deramore Lane, Heslington, York, YO10 5GH, UK
daniel.kudenko@york.ac.uk

ABSTRACT

Being able to learn to communicate is an important attribute for an intelligent agent. Q-learning is a type of reinforcement learning method that solves AI problems through learning from experience. The main aim for reinforcement learning is to find an optimal policy and Q-learning is commonly used to achieve this. The paper is to apply Q-learning to argumentation in order to enable an agent to learn to argue with another agent. We start the research by investigating how to apply Q-learning in an abstract argument game context. The initial experiments show that generally speaking the reinforcement learning agent performs well with a long term delayed reward. The evaluations also provide suggestions on how to improve its performance by using more sophisticated dialogue state representations. It is anticipated that this research will contribute to the design and implementation of argumentative software agents.

Keywords

Abstract argumentation, Reinforcement learning, Argument game, Q-Learning algorithm.

1. INTRODUCTION

There has been increasing research in argumentation and dialogue systems in the past decade[22]. The agent as a dialogue participant needs sophisticated dialogue strategies in order to make high quality dialogue contributions. By reviewing the state of art literature in computerised dialogue systems (e.g. [20];[21]), their dialogue strategies (i.e. strategic heuristics) are hardwired into the computational agent. One of the main issue with this is that an agent may not be capable of dealing with new dialogue situations that have not been coded, and indeed this is an impossible task given the dynamic nature of argumentation. It would be ideal to enable an agent to search for an optimal strategy by itself e.g. via trial and error, and thus the agent with the best strategy will win the competition [8]. Machine learning has an important role to play in order to meet these challenges. By allowing agents learning dialogue strategies, it would be more flexible for them to make an argument through exploration (trial and error). It is believed that learning can make agents more flexible to adapt to new environments and new dialogue situations. One of the popular machine learning approaches when involving agents is reinforcement learning (henceforth referred to as RL). RL could manage a dialogue as MDP whereas dialogue moves transition between states and reward can be given by the end of the dialogue [5]. The work reported in this paper is to investigate the means to design an argumentative reinforcement learning agent and then study the consequences. Though there are some works reported in the literature that combine reinforcement learning with argumentation e.g. [7], they have a heavy focus on negotiation which is a distinctive kind of dialogue as opposed to persuasive argumentation according to Walton and Krabbe [16] .

This remainder of this paper is organised as follows. Abstract argumentation system, which is adopted for the reason of simplicity to represent the dialogue interaction environment, is introduced in section 2. Section 3 provides an argument game model which is used to regulate the evolving dialogue interaction. The design of an argumentative reinforcement learning agent including state, action, environment and rewards design, is presented in section 4. In order to facilitate the evaluation of the argumentative RL agent, an experimental testbed is constructed. The testbed contains agents with different argumentation strategies in order to act as the dialogue partner of the RL agent. These are provided in section 5. Section 6 presents a number of experiments that have been carried out with the preliminary results discussed. Section 7 concludes the paper and gives pointers for our immediate future work.

2. ABSTRACT ARGUMENTATION SYSTEM

Dung [2] defines an abstract argumentation system (henceforth referred to as AAS) as a pair of A=< X,$\leftarrow$ > where X is set of arguments and $\leftarrow$ is a set of attacking relations between arguments. For instance as shown in figure 1, X= $\{a, c, d, b, e, h, g, i, j, f, n, p, q, l, k, m\}$ and the set of relations between arguments is $\{a \leftarrow c, a \leftarrow d...\}$,as an example, $a \leftarrow c$ means argument c attacks argument a.

As depicted in figure 1, an AAS can be visualised as a directed graph, which contains nodes and arrows representing arguments and attacking relations respectively [23]. AAS concentrates on the high level abstraction of an argumentation systems and does not consider the internal structure of an argument i.e. premises and conclusion. AAS represents informal human reasoning in a way that a computer can easily perform computation on a collection of arguments, e.g. to decide whether an argument acceptable and compute var-

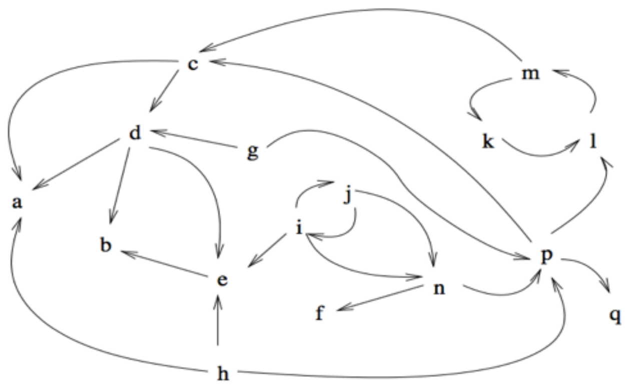


Figure 1: Example of abstract argumentation system

ious semantics for an AAS. In this way, AAS to some extent bridges the gap between human and machine reasoning [9].

The AAS as shown in figure 2 is used to illustrate different argument semantics in action. The abstract argumentation framework contains a set of arguments X=$\{p, q, r, s\}$ and a set of relations R= $\{(r, q), (s, q), (q, p)\}$ where r and s attack q, q attacks p. .

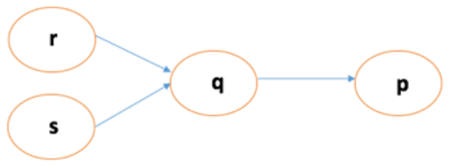


Figure 2: Example of abstract argumentation system

An example set of concrete arguments that matches figure 2 is given below:

1. p:"I think this movie is boring"
2. q:"I think this movie is interesting because it is fun"
3. r:"It is repetitive story"
4. s:"It does not even make much sense"

Because argument p is attacked by argument q, and argument q is attacked by arguments r and s, argument p is therefore defended by arguments r and s. A position S is conflict free (consistent) if there are no elements in S that attack each other. Mutual defence occurs if there is an element in S that is defended by other elements in S. We can call a set S admissible if it is conflict free and mutually defensive. To evaluate whether each argument is acceptable or not,[2] it is necessary to introduce various extensions. Set S is a preferred extension if it is a maximum admissible set. Another extension is called a (unique) grounded extension. It is unique because the grounded extension contains all the arguments that are not attacked, as well as the arguments that are defended by non-attacked arguments[12]. P is in a grounded extension if P is guaranteed to be acceptable which means there is no reason to doubt P. Thus, every acceptable argument is IN and every argument that has been attacked but not subsequently defended will be OUT. Dunne et al.,[4] developed an algorithm for computing the grounded extension, as shown in figure 3.

```
Function $ge(\mathcal{X}, \mathcal{A})$ returns a subset of $\mathcal{X}$
1. $in \leftarrow out \leftarrow \emptyset$
2. while $initial(\langle \mathcal{X}, \mathcal{A} \rangle) \neq \emptyset$ do
3.     $in \leftarrow in \cup initial(\langle \mathcal{X}, \mathcal{A} \rangle)$
4.     $out \leftarrow out \cup \{\alpha \in \mathcal{X} : \exists \alpha' \in in \text{ s.t. } \langle \alpha', \alpha \rangle \in \mathcal{A}\}$
5.     $\mathcal{X} \leftarrow \mathcal{X} \setminus (out \cup in)$
6.     $\mathcal{A} \leftarrow \mathcal{A}$ restricted to $\mathcal{X}$
7. end-while
8. return $in$.
```

Figure 3: Grounded Extension Algorithm

There are also other extensions defined by Dung [2],as follows:

1. Complete extension: Admissible set S is a complete extension iff all arguments defended by S are also in S [13].
2. Stable extension: A conflict free set of arguments S will be called a stable extension if and only if S attacks each argument that does not belong to S.

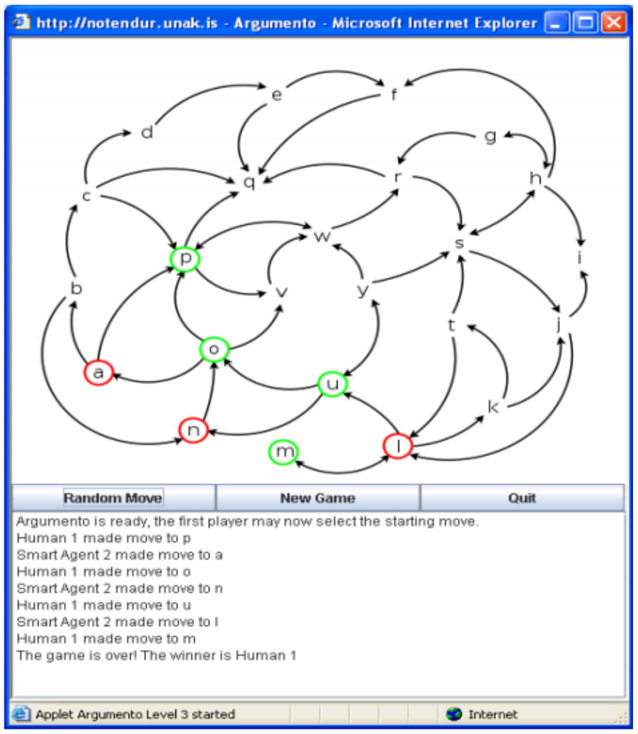
In this paper we will focus on the the use of grounded extension as the reward in RL, because it contains the set of acceptable argument that have been put forward by the dialogue participants, and for each individual agent wishing to maximise the acceptability of their own arguments after win the game in each episode. An abstract argument game, which facilitates the argumentation reasoning for two dialogue participants, will be discussed in the next section.

3. ARGUMENT GAME

Argument game can be considered as a dialectical context for argument exchange between agents[23]. An argument game specifies dialogue rules that the argument participants need to follow during the argumentation process. There are different argument games that have been developed in the area of computational argumentation (e.g.[17];[15];[3]). In this paper, we adopt the argument game represented in[17] for reasons of simplicity and flexibility. The argument game can be represented as a tuple of: G=<A, D, R, P> where: A is the argumentation system, D is the dialogue history which contains a set of moves made by the players, R is the set of rules that players need to follow when making a move, P is the set of players, normally 2 denoted as 0, 1. In [17], Wooldridge identify six rules that each player (agent) must follow in a simple argument game and they are:

1. First move in D is made by player0 e.g. $P_0 = 0$
2. Players take turns making moves (one move per turn). $P_i = P_{imod2}$.
3. Players cannot repeat a move $\forall a_i, a_j \in D, a_i \neq a_j$.
4. Each move must attack (defeat) the previous move $a_i \rightarrow a_{i-1}$
5. The game is ended if no further moves are possible $\forall a_i \in A \wedge \notin D, a_i \nrightarrow a_n$
6. The winner of the game is the player that makes the final move $G_{winner} \doteq P_{nmod2}$

Figure 4 illustrates an example argument game in action with respect to the set of rules [23].


http://notendur.unak.is - Argumento - Microsoft Internet Explorer
Random Move
New Game
Quit
Applet Argumento Level 3 started
Internet

```
Argumento is ready, the first player may now select the starting move.
Human 1 made move to p
Smart Agent 2 made move to a
Human 1 made move to o
Smart Agent 2 made move to n
Human 1 made move to u
Smart Agent 2 made move to l
Human 1 made move to m
The game is over! The winner is Human 1
```

Figure 4: An example argument game scenario taken from [23]

4. ARGUMENTATIVE RL AGENT DESIGN

Over the past few decades, research and applications in machine learning have increased significantly. Machine learning occurs when a computer agent learns from data with the aim of improving its performance and behaviour[6]. Reinforcement learning is a type of machine learning which focuses on how to map an action for each state by interacting with the environment and observing the state change[14]. Sutton and Barto[14] define reinforcement learning (RL) as an agent learning what to do and how to connect each situation with an action to maximise the cumulative reward. The learner or agent is not told what action should be taken, rather the learner needs to explore a policy that yields the maximum cumulative reward by trying them out. In reinforcement learning, the agent interacts with the environment to take an action and receive the reward for the action taken as seen in figure 5.

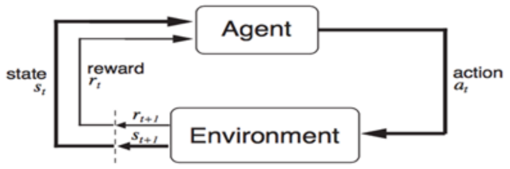
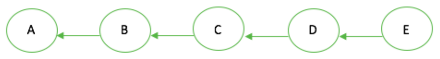


Figure 5: Reinforcement learning agent-environment interaction

In addition to the agent and the environment, there are various components for reinforcement learning, they include: policy, reward function, value function and the model of the environment[14]. To make an agent learn to argue we should identify states, actions, environment and the rewards. In abstract argumentation system, we declare the knowledge base for an agent. The classical state representation in agent literature (e.g.[18];[1]) is adopted where states are identified by nodes in the argumentation graph and action by the attack relation between arguments. The main objective of reinforcement learning is to allow the agent to learn how to act in the environment to maximize the long term cumulative reward, and to explore the optimal actions for each state to achieve the agent's goal. Yang and Gu[19] support the view that learning will occur iteratively and through a trial-and-error method, depending on the experience of interaction between the agent and the environment and the reward it received. In this paper we associated the reward for the agent as the number of acceptable arguments in the grounded extension for reasons discussed in section 2 above. The argumentation fragment in figure 6 shows the arguments E, C and A are in grounded extension.



States S= {A, B, C, D, E}
Actions A= {B→A, C→B, D→C, E→D}

Agent 1: Choosing argument A (IN)
Agent 2: Choosing argument B (OUT)
Agent 1: Choosing argument C (IN)
Agent 2: Choosing argument D (OUT)
Agent 1: Choosing argument E (IN)

The Grounded extensions will be number of INS arguments which are E, C, A

Figure 6: Grounded extension example

This agent adopts a commonly used RL method, that is Q-learning. With sufficient training, Q-Learning algorithm can converge with a probability of 1 to very close to the action-value function for a target policy. Therefore, Q-Learning can learn the optimal policy even when actions are selected according to a more exploratory or even random policy. The formula for the Q-learning algorithm is stated below:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_t, a_{t+1}) - Q(s_t, a_t)]$$

The aim of Q-learning is to allow an agent to learn through experience and map each state with an action by choosing the maximum value from the Q-table which is updated after each episode. In the above formula, it can be seen that learning rate controls how fast learning takes place. The current reward r_{t+1} is set to 0 because a delayed reward or final reward for each argument episode is used. The delayed/final reward is the number of arguments in grounded extension that have been made by an agent in an argument episode.

We consider epsilon $\epsilon = 0.3$ to enable an agent to explore randomly for up to 30 percent of the time as opposed to choose an action based on experience so that the agent is able to explore new argument route in order to win the game. Before the agent takes an action, a random number is generated between 0 and 1 and if it is less than ϵ the agent will choose an action randomly rather than following the value in the Q-table, as shown in the algorithm below:

```
chooseQLearningAction(argument,epsilon,gameMoves):
   randomNumber=generateRandomNumber()
      IF randomNumber < epsilon
         chooseRandomAction(argument,gameMoves)
      ELSE
         chooseFromQTable()
      END IF
```

The Q-table is updated at the end of each game episode against the algorithm below:

```
updateQMatrixAfterGame(epsilon, gameMoves,reward)
Set the discount factor , learning rate
Initialize Q Matrix = 0
FOR all state :
  action=chooseQLearningAction(state,epsilon,gameMoves)
  nextState=parent(action)
  maxQ(nextState,action)=max(nextState)
  compute Q-table(state,action) value using the
      ↪ formula below:
    Q(state,action)=
    Q(state,action)+
    $\alpha$ (reward+ $\gamma$ max
        ↪ Q(nextState,action)-Q(state,action))
END FOR
```

For the running time of the Q-Learning updateQMatrixAfterGame operation, if the number of attacking relationships is r and the number of arguments is n. The running time is therefore $O(r * n)$, in a worst case scenario where all nodes attacks each other,$r \simeq n$, hence the running time is $O(N^2)$.

The aim of this research is to assess whether an agent can learn to argue and whether it can learn effectively. In order to facilitate the evaluation of the performance of the the RL agent discussed above, one approach is to enable the RL agent to play against agent with different strategies in a competitive environment and then analyse their performance. An experimental testbed with various agents is discussed next.

5. ARGUMENTO+

To test the RL agent discussed above in operation, a software testbed, Argumento+, named after its predecessor Argumento as reported in[23], has been built using the Java programming language. Argumento+ contains the RL agent as well as three other agents namely, random, maximum probability utility and minimum probability utility agent for the sake of the evaluation. RL agent plays game against them to maximise the cumulative reward by winning more games. Indeed, if RL agent win the game, it will receive rewards based on the number of acceptable arguments namely grounded extensions. We considered grounded extension because it contains an argument that has no doubt compared to other arguments [18], meaning it will be a more acceptable argument. To discuss Argumento+ we need first to identify other agents(opponents) which will be discussed in turn below.

5.1 Random agent

This agent has no major strategy but uses a pseudo random number to select an argument from the set of legally available argument moves following the algorithm below.

```
chooseRandomAction(argument, gameMoves):
availablemoves = getAvailableMove()
return randomly newArg∈ availablemoves and
     ↪ newArg ∉ gameMoves
```

A legal argument can be defined as an argument such as: $\alpha \in A \wedge \alpha \rightarrow last(D) \wedge \alpha \notin D$. A represents a list of valid argument in the argumentation system, D is the games dialogue history. Invariable, the relationship above means that an argument is said to be a valid argument if it attacks the last move of the game as contained in the game dialogue history while the argument itself has not been used in the game. For the Random Agent, the algorithm has to first traverse A to retrieve all arguments that attack last(D) and then traverse the game dialogue history to ensure that this argument has not been used in the game, hence the computation time for the worst-case scenario is $O(n^2)$ where n is the number of arguments in A.

5.2 Max-probability utility agent

A dialogue tree is first generated from an initial argument by using the algorithm below :

```
parentList = parent(argument) where
     ↪ parenList∈gameMoves
probabilityList = ∅
   FOR  ∀ parent ∈  parentList
      parentProbability
         ↪ =computeProbability(argument,parent,gameMoves)
      probabilityList.add(parentProbability)
   END FOR
return max(probabilityList)
```

The dialogue tree computes the probability of each parent node. The computeProbability algorithm recursively calculates the parent probability of each node from the initial argument till a node without a parent is reached. The algorithm is specified below as below: NB: depthOfTree computes the depth of the nodes of parents emanating from the initial argument.

```
computeProbabilty(argument,parent,gameMoves)
parentList = parent(parent) where
     ↪ parentList∉gameMoves
   IF parentList==∅
      depthOfTree=depthOfTree+1
      probability = depthOfTree%2
   ELSE
      depthOfTree=depthOfTree+1
         FOR ∀ parent ∈ parentList
            probability=probability+
            computeProbability(argument,
                ↪ parent,gameMoves)
         END FOR
      probability=probability/SIZE(parentList)
   END IF
return probability
```

Figure 7 illustrates an example resulting dialogue tree with utility values for each node. It can be seen that the two branches on the left are winning branches for the agent and the probability utility value is therefore 1, and the branch on the right is losing and its value is 0 [23].

The running time of the dialogueTree is therefore $O(r*n)$, where n is the number of argument needed to be traversed to get all argument attacking last(D) and r being the total

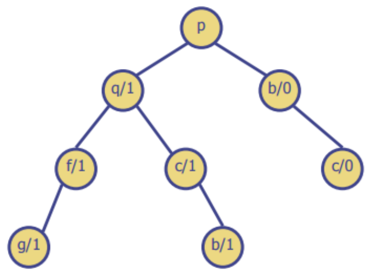


Figure 7: A dialogue tree with probability utility

number of attacking relations in the system i.e. assuming the root of the tree is the first argument when running the game. Computing the running time for the computeProbability method therefore is $O(r)$, therefore the general running time is $O(r * n)$. In a worst-case scenario where each argument attacks each other, the computation time will be $O(n!)$, but generally, the number of attackers a is$<< n$ hence the running time will be $O(a!)$.

5.3 Min-probability utility agent

Utilizing similar strategy as max-probability utility agent but instead of choosing the maximum probability of the parent of the argument, it chooses the minimum probability.

6. EXPERIMENT AND RESULT

To understand the performance and efficiency of the RL agent, it is necessary to conduct experiments and then analyse the results.The agents discussed above can be paired up with RL agent in order to facilitate the evaluation. Argomento+ first creates a text file which contains an argumentation graph with nodes and edges. It then reads the graph from the file and asking to play the argument game. User needs to choose the discounted factor value which is believed to have a direct impact on agent behaviour. Then, user needs to choose an agent who plays with (i.e. random agent, max or min-probability utility agent) to play with RL agent and the simulation is ran for 30000 episodes. The data will be saved into two files: a csv file containing the final rewards for each agents and a text file for reinforcement learning agent to store the final Q-table data. Figure 8 illustrates how the program is operated.

The purpose of the experiment is to assess how reinforcement learning agent behaves and how effective is the learning. We collected data from three different game settings:

1. RL agent vs Random agent.
2. RL agent vs Max-probability utility agent.
3. RL agent vs Min-probability utility agent.

For each game setting, the simulation is run with the following configurations: i) for episode 0, we put epsilon = 0 and run the game 10 times then take the average of the final rewards. At this stage, all values in q-table remain as zeros. ii) The agent then carries out exploration and updates Q-table accordingly. iii) After every 10th episode of exploration epsilon is set to 0 to evaluate the agent's policy, and the evaluation is done by running the game 10 times then taking an average of the final rewards for each episode.

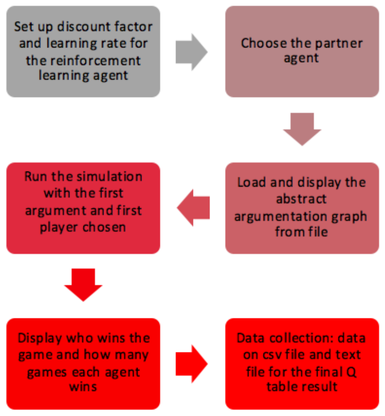

Set up discount factor and learning rate for the reinforcement learning agent
Choose the partner agent
Load and display the abstract argumentation graph from file
Run the simulation with the first argument and first player chosen
Display who wins the game and how many games each agent wins
Data collection: data on csv file and text file for the final Q table result

Figure 8: Process of the program

30000 game episodes had been run against each of the game settings. For the sake of the appropriate display of the result in a graph, we have taken the average of the final rewards for every 500 episodes. The performance for the pair of agents in each setting can be found from figure 9, 10 and 11 respectively.

From the lower side of figure 9, it could be clearly seen that RL agent learns quickly from the beginning to the 1500th episodes. It then still learns but at a lower pace. It does outperform the max-probability agent at some point as shown in figure 9. In contrast, the performance of the max-probability agent drops which means that the learning agent is experiencing a good learning curve. A similar learning curve is also observed when playing against the min-probability agent as shown in figure 10 and the reward is stabled at 1.7 for the rest of episodes. The performance of the opponent agent dropped rapidly to 0.6. which means the learning agent is performing well. However, the performance of the leaning agent drops from episode 0 to 1000 and then hardly learns when playing against a random agent as illustrated in figure 11. To investigate this problem, we prefix the starting argument as well as which agent starting the game. We put the argument b in figure 1 as the initial argument as while reinforcement learning agent always starts the game. Unfortunately, the performance of the learning agent against random agent is still consistently as bad as shown in figure 11. The negative results from the experiment triggered us to conduct further analysis into the game scenario. The analysis shows that that the problem is related to the state representation in the current RL design. For instance the tree in figure 12 shows all possible argumentation paths that are rooted at argument b, when random agent chooses argument e, learning agent should choose the argument with the maximum Q value from d, i and h against the Q-table in figure 13. In this case, the q-value for d, i, and h are: 12.17, 19.38 and 2.03 respectively, therefore i will be chosen because it contains the highest value but i will cause the learning agent lose the game because the last move is for the random agent. However, if h-the argument with the lowest Q-value is chosen, it will lead to victory. So the question becomes why a bad move like i in this case has a high Q-value. The reason for this is that there are a few another occurrences of state i in the tree which give a much better payoff, and on aggregation, they have lifted the Q-value

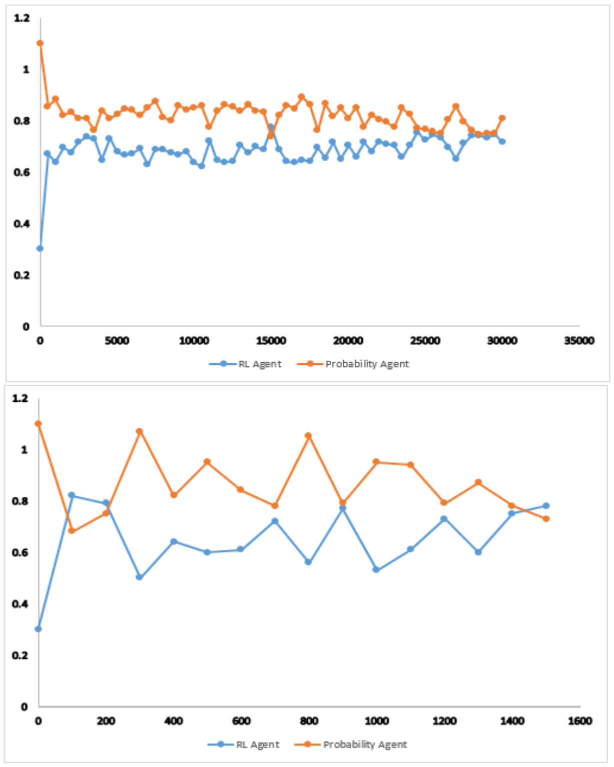


Figure 9: RL agent against Max probability agent (1500 episodes in the lower side)

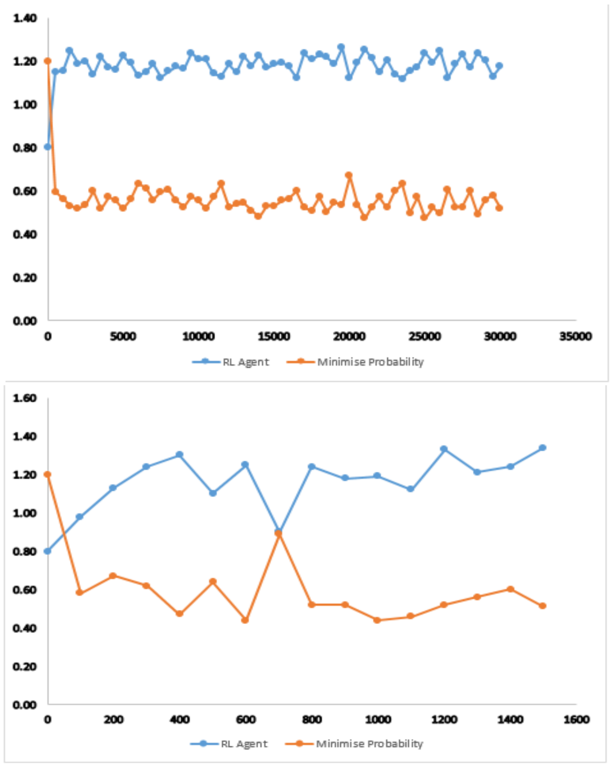


Figure 10: RL agent against Min probability agent (1500 episodes in the lower side)

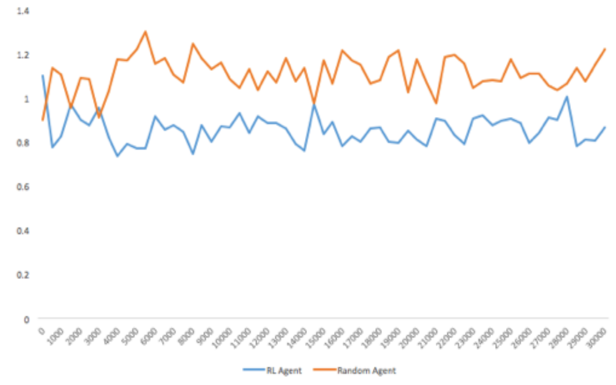


Figure 11: RL agent VS Random agent

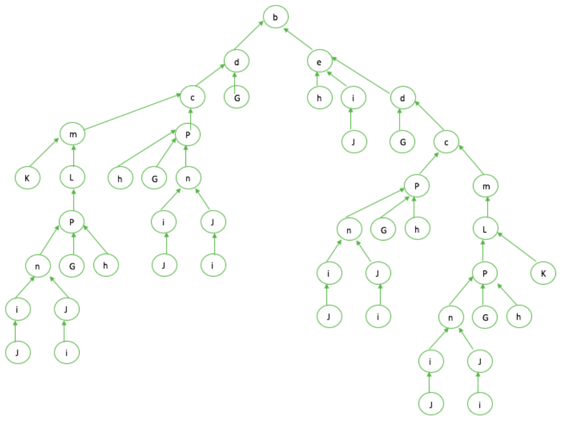


Figure 12: Arguments tree rooted at argument "b"

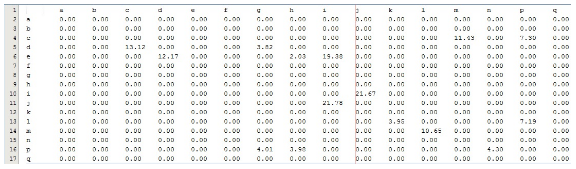
for state i. Fundamentally, the classical approach to state representation[1] adopted in this paper falls short for the argumentation domain.

To tackle this issue, more sophisticated dialogue state representation is needed in order to make each state is unique. We are planning to use the concept of combined state which contains the previous move and the second last move in the dialogue history. For example in figure 14, instead of representing the state as e, the combine state would be (b, e).

7. CONCLUSION AND FUTURE WORK

We have designed an agent that is able to learn to argue by using the Q-learning approach. We initially adopted the classical state representation in the agent literature and the number of arguments in grounded extension as the delayed reward. A number of experiments have been carried out by engaging RL agent with max, min-probability and random agent in an argument. The experiments show that generally speaking, agents are learning well and as a result the methods are fairly encouraging.

The experiments also revealed a weakness of the state representation in the argumentation domain, we are planning to investigate it further. A further observation is that the reward function seems to encourage longer dialogue that leads to a win. In argumentation, an alternative strategy might be to win with minimum number arguments in Oren et al.[10].



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	p	q
a	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
b	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
c	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	11.43	0.00	7.30	0.00
d	0.00	0.00	13.12	0.00	0.00	0.00	3.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
e	0.00	0.00	0.00	12.17	0.00	0.00	0.00	2.03	19.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00
f	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
g	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
h	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
i	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	21.67	0.00	0.00	0.00	0.00	0.00	0.00
j	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	21.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00
k	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
l	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.95	0.00	0.00	0.00	7.19	0.00
m	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.65	0.00	0.00	0.00	0.00
n	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
p	0.00	0.00	0.00	0.00	0.00	0.00	4.01	3.98	0.00	0.00	0.00	0.00	0.00	4.30	0.00	0.00
q	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 13: The Qtable

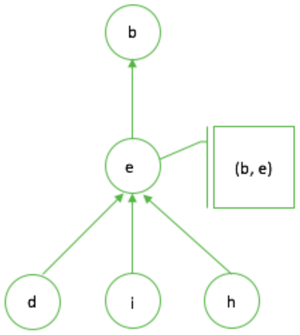


Figure 14: Combine state representation

It might be worthwhile to investigate further along this line. And finally, the current argument game disallow backtracking while in argumentation literature it is often seen as a good strategy (e.g. Yuan et al. [23]; Prakken[11]), we are planning to introduce backtracking into Argumento+ and then study the consequences. We also plan to generalise the policy of Q-value estimates between states in order to apply different argumentation graphs.

REFERENCES

[1] H. Cuayáhuitl, S. Keizer, and O. Lemon. Strategic dialogue management via deep reinforcement learning. *arXiv preprint arXiv:1511.08099*, 2015.

[2] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[3] P. E. Dunne and T. J. Bench-Capon. Two party immediate response disputes: Properties and efficiency. *Artificial Intelligence*, 149(2):221–250, 2003.

[4] P. E. Dunne, A. Hunter, P. McBurney, S. Parsons, and M. Wooldridge. Weighted argument systems: Basic definitions, algorithms, and complexity results. *Artificial Intelligence*, 175(2):457–486, 2011.

[5] K. Georgila and D. R. Traum. Reinforcement learning of argumentation dialogue policies in negotiation. In *INTERSPEECH*, pages 2073–2076, 2011.

[6] M. Hall, I. Witten, and E. Frank. Data mining: Practical machine learning tools and techniques. *Kaufmann, Burlington*, 2011.

[7] T. Hiraoka, K. Georgila, E. Nouri, D. Traum, and S. Nakamura. Reinforcement learning in multi-party trading dialog. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 32, 2015.

[8] P. S. Kośmicki. A platform for the evaluation of automated argumentation strategies. In *International Conference on Rough Sets and Current Trends in Computing*, pages 494–503. Springer, 2010.

[9] S. Modgil, F. Toni, F. Bex, I. Bratko, C. I. Chesnevar, W. Dvořák, M. A. Falappa, X. Fan, S. A. Gaggl, A. J. García, et al. The added value of argumentation. In *Agreement technologies*, pages 357–403. Springer, 2013.

[10] N. Oren, T. J. Norman, and A. Preece. Loose lips sink ships: A heuristic for argumentation. In *Proc. of the 3rd Int'l Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2006)*, pages 121–134. Citeseer, 2006.

[11] H. Prakken. Coherence and flexibility in dialogue games for argumentation. *J. Log. Comput.*, 15(6):1009–1040, 2005.

[12] I. Rahwan, K. Larson, and F. A. Tohmé. A characterisation of strategy-proofness for grounded argumentation semantics. In *IJCAI*, pages 251–256. Citeseer, 2009.

[13] I. Rahwan and F. Tohmé. Collective argument evaluation as judgement aggregation. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 417–424. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[14] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[15] G. A. Vreeswik and H. Prakken. Credulous and sceptical argument games for preferred semantics. In *European Workshop on Logics in Artificial Intelligence*, pages 239–253. Springer, 2000.

[16] D. Walton and E. C. Krabbe. Commitment in dialogue, 1995.

[17] M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2002.

[18] M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[19] E. Yang and D. Gu. Multiagent reinforcement learning for multi-robot systems: A survey. Technical report, tech. rep, 2004.

[20] T. Yuan, D. Moore, and A. Grierson. A human–computer debating system prototype and its dialogue strategies. *International Journal of Intelligent Systems*, 22(1):133–156, 2007.

[21] T. Yuan, D. Moore, and A. Grierson. A human-computer dialogue system for educational debate: A computational dialectics approach. *International Journal of Artificial Intelligence in Education*, 18(1):3–26, 2008.

[22] T. Yuan, J. Schulze, J. Devereux, and C. Reed. Towards an arguing agents competition: Building on argumento. In *Proceedings of IJCAI'2008 Workshop on Computational Models of Natural Argument*, 2008.

[23] T. Yuan, V. Svansson, D. Moore, and A. Grierson. A computer game for abstract argumentation. In *Proceedings of the 7th Workshop on Computational Models of Natural Argument (CMNA07)*, 2007.