Learning in automated negotiation

Renting, B.M.

5

Towards General Negotiation Strategies with End-to-End Reinforcement Learning

This chapter has been published in the Reinforcement Learning Journal (RLJ) [126].

5.1 Introduction

Traditionally, negotiating agents were manually designed algorithms based on heuristics, which is still a commonly seen approach in recent editions of the Automated Negotiation Agents Competition (ANAC) [5]. However, manually designing such negotiation strategies is time-consuming and results in highly specialised and fixed negotiation strategies that do not generalise over a broad set of negotiation settings. In later work, optimisation methods were used to optimise the parameters of negotiation strategies using evolutionary algorithms [46, 43, 92], or algorithm configuration techniques [124].

In Chapters 3 and 4, we showed that algorithm configuration and portfolio selection methods can be used to learn autonomous agents to negotiate. The proposed approaches allow negotiation strategies to be more easily adaptable to different negotiation settings. However, they still require a relatively high degree of manual design to obtain a parameterised negotiation strategy, making them time-consuming to build, limiting their generalisability by specialisation on specific negotiation settings, and inducing human bias in strategy design.

With the advent of reinforcement learning (RL) [146], there have been attempts at using RL-based methods for creating negotiation agents [19]. There is, however, still an open challenge. In automated negotiation, it is common for agents to deal with various negotiation scenarios that would cause differently sized observation and action vectors for default linear layer-based RL policies. Up until now, this issue has been dealt with by either abstracting the observations and actions into a fixed-length vector (see, e.g., Bakker et al. [19]) or by fixing the negotiation scenario, such that the observation and action space remain identical (see, e.g., Higa et al. [68]). The first approach causes information loss due to feature design, and the latter renders the obtained policy non-transferable to other negotiation scenarios.

We set out on the idea that a more general RL-based negotiation strategy capable of dealing with various negotiation scenarios is achievable and that such a strategy can be learned using end-to-end reinforcement learning without using state abstractions and without the human bias induced in the design of parameterised agents. Developing such an RL negotiation strategy would open up new avenues for RL in automated negotiation as policy networks are easily extendable. End-to-end methods might also be able to learn complex relations between observations and actions, minimising the risk of information loss that is often imposed by (partially) manual design strategies.

To this extent, we designed a graph-based representation of a negotiation scenario. We applied graph neural networks in the RL policy to deal with the changing dimensions of both the observation and action space. We show that our method performs about as well as a recent end-to-end RL-based method designed to deal only with a fixed negotiation scenario. More importantly, we show that our end-to-end method can successfully learn to negotiate with other agents and that the obtained policy also performs well on previously unseen, randomly generated linear-additive negotiation scenarios.

5.2 RELATED WORK

Bakker et al. [19] applied RL to decide what utility to demand in the next offer. They abstracted the state to utility values of the last few offers and time towards the deadline. Translating utility to an offer, estimating opponent utility, and deciding when to accept were done without RL. Bagga et al. [18] also abstracted the state into a fixed representation with utility statistics of historical offers. They used an RL policy to decide whether to accept and a separate policy that outputs offers based on a non-RL opponent utility estimation model.

Sengupta et al. [137] encoded the state into a fixed length of past utility values. The action is the target utility of the next offer, translated to an actual offer through an exhaustive search of the outcome space. They trained a portfolio of policies and tried to select effective counterstrategies by classifying the opponent type. Li et al. [97] also build a portfolio of RL-based negotiation strategies by incrementally training best responses based on the Nash bargaining solution. During evaluation, their method searches for the best response in an effort to improve cooperativity. They only applied their method to fixed negotiation scenarios.

Another line of research on negotiation agents includes natural language. An environment for this was developed by Lewis et al. [95]. Kwon et al. [90] used this environment and applied a combination of RL, supervised learning, and expert annotations (based on a dataset) to iteratively train two agents through self-play. The negotiation scenarios considered are fixed, except for the preferences.

Takahashi et al. [148] and Higa et al. [68] are closest to our work, as they also train an end-to-end RL method for negotiation games. Their approach does not use state abstractions and linearly maps the negotiation scenario and actions in a policy. The policy obtained can only be used for a fixed scenario. They trained and tested only against single opponents

Graph Neural Networks (GNNs) [86] have been used to handle graph-structured input in policy networks, for example, in molecular design [165]. Wang et al. [156] and Yang et al. [163] applied them to transfer learn over variable action spaces of various multi-joint robots. However, they aimed to speed up learning on unseen tasks, while we strive for complete transferability without additional learning.

5.3 METHODS

We formulate the negotiation game as a turn-based Partially Observable Stochastic Game (POSG), a partially observable extension of a stochastic game [139]. We model the game as a tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, \mathcal{O}_i, \mathcal{A}_i, \mathcal{T}, \Omega_i, \mathcal{R}_i \rangle$, where $\mathcal{I} = \{1, \cdots, n\}$ denotes the set of agents, $\mathcal{S}$ the set of states, $\mathcal{O}_i$ the set of possible observations for agent i, and $\mathcal{A}_i$ the set of actions for agent i. For convenience, we write $\mathcal{A} = \mathcal{A}_i$, as we consider a turn-based game where the set of actions is identical for each agents. Furthermore, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto p(\mathcal{S})$ denotes the transition function, $\Omega_i : \mathcal{S} \times \mathcal{A} \mapsto p(\mathcal{O}_i)$ the observation function for agent i, and $\mathcal{R}_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ the reward function for agent i.

The game starts in a particular state s. Then, at timestep t, an agent i selects an action $a_{t,i}$ independently of other agents. Based on this action, the state of the

POSG changes according to $s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t, a_t)$. Subsequently, each agent receives its own observation $o_{t,i} \sim \Omega_i(o_{t,i}|s_t, a_t)$ and associated reward $r_{t,i} \sim \mathcal{R}_i(r_{t,i}|s_t, a_t)$.

Each agent i selects actions according to its own policy $\pi_i : \mathcal{O}_i \times \mathcal{O}_i \times \cdots \rightarrow p(\mathcal{A})$. At timestep t, agent i samples an action $a_t \sim \pi_i(a_t|o_{t,i}, o_{t-1,i}, \cdots)$. Note that we can vary the length of the historical observations by which we condition the policy for each agent. The more history we include, the more we can overcome partial observability.

Our goal is to find a policy π_i for agent i that maximizes cumulative expected return:

$$\pi_i^\star \in \arg\max_{\pi_i} \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{k=0}^{H-1} \mathcal{R}_i(s_{t+k}, a_{t+k}) \right], \tag{5.1}$$

where H denotes the horizon of the POSG (the number of rounds we select an action). Crucially, the performance of a particular policy π_i depends on the policies of the other agents.

5.3.1 Proximal Policy Optimisation

We will use reinforcement learning to optimize the policy π_i of our own agent i in the negotiation scenario. For simplicity, we will drop the subscript i and simply write π for the policy of our own agent. We also simplify by writing o instead of $\langle o_{t,i}, o_{t-1,i}, \cdots \rangle$. To optimize this policy, we use Proximal Policy Optimisation (PPO) [135] due to its empirical success and stability.

At each update iteration k, PPO optimises π relative to the last policy π_k by maximising the PPO clip objective:

$$\pi_{k+1} \in \arg\max_{\pi} \mathbb{E}_{o,a\sim\pi_k} \left[\min\left(\frac{\pi(a|o)}{\pi_k(a|o)} A_{\pi_k}(o,a),\ \text{clip}\left(\frac{\pi(a|o)}{\pi_k(a|o)}, 1 \pm \epsilon \right) A_{\pi_k}(o,a) \right) \right] \tag{5.2}$$

where ϵ denotes a clip parameter, and $A_\pi(a,o)$ denotes the advantage function of taking action a in observation o [146]. The ratio gets clipped to ensure that the new policy does not change too quickly from the policy at the previous step. Our PPO implementation is based on the CleanRL repository [73].

5.3.2 Graph neural networks

We aim to learn to negotiate across randomly generated scenarios where the number of objectives and values differ. This forces us to design a policy/value network where the shape and number of parameters are independent of the number of objectives and values. Networks of linear layers, often the default in RL, do not fit this criterion, as they require fixed input dimensions. We chose to represent the input of the policy network as a graph and make use of Graph Neural Networks (GNN) to deal with the changing size of the input space, more specifically, Graph Attention Networks (GAT) [152].

The input graph contains nodes that have node features. A layer of GNN encodes the features x_u of node u into a hidden representation h_u based on the features of the set of neighbour nodes $\mathcal{N}_u$ and on its own features. The specific case of GATs is defined in Equation 5.3. Here, neighbour features are encoded by a linear layer ψ

and then weighted summed through a learned attention coefficient $a(x_u, x_v)$. The weighted sum is concatenated with x_u and passed through another linear layer ϕ to obtain the embedding of the node h_u.

$$h_u = \phi\left(x_u, \sum_{v \in \mathcal{N}_u} a(x_u, x_v) \cdot \psi(x_v)\right) \tag{5.3}$$

5.3.3 IMPLEMENTATION

At each timestep, the agent receives observations that are the actions of the opponent in the negotiation game. Based on these observations, the agent must select an action. The action space combines multiple categorical actions: the accept action and an action per objective to select one of the values in that objective as an offer. If the policy outputs an accept action, then the offer action becomes irrelevant as the negotiation will be ended.

A negotiation scenario has objectives B and a set of values to decide on per objective Ω_b. We represent the structure of objectives and values as a graph and encode the history of observations $\langle o_{t,i}, o_{t-1,i}, \cdots \rangle$ of a negotiation game in this structure to a single observation o (see the left side of Figure 5.1). Each objective and value is represented by a node, where value nodes are connected to the objective node to which they belong. An additional head node is added that is connected to all objective nodes. The node features of each node are:

- 5 features for each value node: the weight $w_b(\omega_b)$ of the value, a binary value to indicate the opponent's last offer, a binary value to indicate the last offer of the agent itself, the fraction of times this value was offered by the opponent, and the fraction of times this value was offered by itself. Note that it might have been better to implement a recurrent network to condition the policy on the full history of offers instead of summary statistics. However, the added computational complexity would have rendered this work much more difficult. Our approach enables efficient learning, but future work should explore the use of the raw history of offers.

- 2 features for each objective node: the number of values in the value set of this objective $|\Omega_b|$, and the weight of this objective $w(b)$.

- 2 features for the head node: the number of objectives $|B|$, and the progress towards the deadline scaled between 0 and 1.

As illustrated in Figure 5.1, we apply GAT layers to the observation graph to propagate information through the graph and embed the node features (Equation 5.3). The size of the representation is a hyperparameter. We then take the representation of the head node and pass it to a linear layer that predicts the state value V. The head representation is also passed through a linear layer to obtain the two accept action logits. Finally, we take the representation of every value node and apply a single linear layer to obtain the offer action logits. These logits are concatenated per action and used to create the probability distribution over the action space. As we

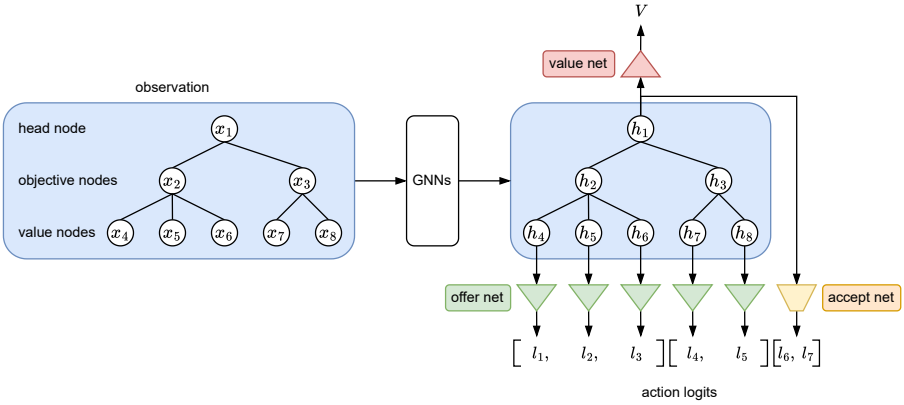


Figure 5.1: Overview of our designed policy network based on GNNs. Observations are encoded in a graph representation (left) and passed through GNNs. Action distribution logits and state value are obtained by passing the learned representation of the head node and value nodes through linear layers.

Table 5.1: Description of baseline negotiation agents used for benchmarking.

Name	Type	Description
BoulwareAgent	Time-dependent	Utility target decreases concave with time
ConcederAgent	Time-dependent	Utility target decreases convex with time
LinearAgent	Time-dependent	Utility target decreases linearly with time
RandomAgent	Random	Makes random offers, accepts any utility > 0.6

use the same linear layer for all value nodes, the number of output logits is independent of the number of parameters in the policy, thus satisfying our requirement. We also note that although the size of the outcome space suffers heavily from the curse of dimensionality when the number of objectives increases, our approach does not. Our code implementation can be found on GitHub[1].

5.4 Empirical evaluation

To train our agent, we need negotiation scenarios as well as opponents to negotiate against. The negotiation scenarios were randomly generated with an outcome space size $|\Omega|$ between 200 and 1000. As opponents, we used baseline agents and agents developed for the 2022 edition of the Automated Negotiation Agents Competition (ANAC). The baseline agents are simple negotiation strategies often used within automated negotiation to evaluate and analyse new agents. We provide a description of the opponents in Table 5.1. All agents were originally developed for the GENIUS negotiation software platform [99].

We set a negotiation deadline of 40 rounds. An opponent is randomly selected during the rollout phase, and a negotiation scenario is randomly generated. The policy is then used to negotiate until the episode ends, either by finding an agree-

[1] https://github.com/brenting/RL-negotiation/tree/RLC-2024

Table 5.2: Hyperparameter settings

Parameter	Value
total timesteps	$2 \cdot 10^6$
batch size	6000
mini batch size	300
policy update epochs	30
entropy coefficient	0.001
discount factor γ	1
value function coefficient	1
GAE λ	0.95
# GAT layers	4
# GAT attention heads	4
hidden representation size	256
Adam learning rate	$3 \cdot 10^{-4}$
Learning rate annealing	True
activation functions	ReLU

ment or reaching the deadline. The episode is added to the experience batch, which is repeated until the experience batch is full. We apply 4 layers of GATs with a hidden representation size of 256. A complete overview of the hyperparameter settings can be found in Table 5.2.

5.4.1 FIXED NEGOTIATION SCENARIO

As a first experiment, we compared our method to a recent end-to-end RL method by Higa et al. [68] that can only be used on a fixed negotiation scenario. Their method was originally only trained and evaluated against single opponents. We chose to train the agent against the set of baseline players instead, as we consider that a more realistic scenario. The baseline agents show relatively similar behaviour, making this an acceptable increase in difficulty.

We generated a single negotiation scenario and trained a reproduction of their and our own method for 2 000 000 timesteps on 10 different seeds. The training curve is illustrated in Figure 5.2, where we plot both the mean of the episodic return and the 99% confidence interval based on the results from 10 training sessions. Every obtained policy is evaluated in 1000 negotiation games against every opponent on this fixed negotiation scenario. We report the average obtained utility of the trained policy and the opponent, including a confidence interval based on the 10 evaluation runs in Figure 5.3.

We can see in Figure 5.3 that our method performs similarly to the method proposed by Higa et al. [68]. This result is mostly a sanity check that our method can successfully learn to negotiate in a relatively simple setup despite being more complex and broadly usable.

5.4.2 RANDOM NEGOTIATION SCENARIOS

We now evaluate the performance of our end-to-end method on randomly generated negotiation scenarios. Negotiation scenarios will continuously change during both training and evaluation.

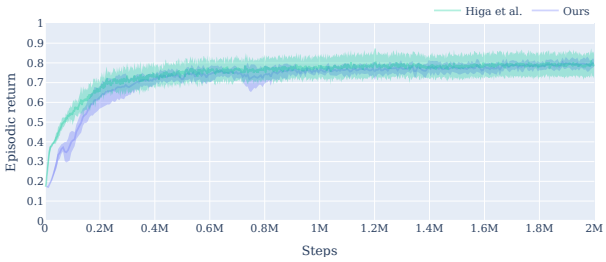


Figure 5.2: Mean and 99% confidence interval of episodic return during training based on results from 10 random seeds . The results of the policy designed by Higa et al. [68] and our policy are plotted.

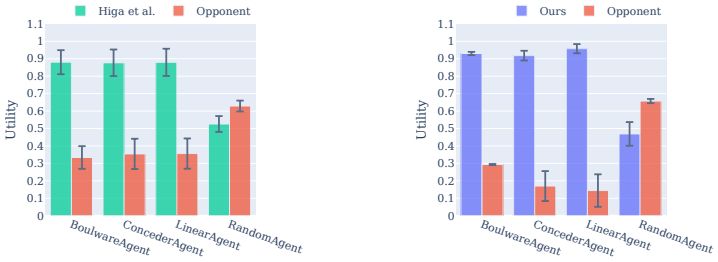


Figure 5.3: Evaluation results of the policy designed by Higa et al. [68] and our GNN-based policy. Results are obtained by evaluating each trained policy for 1000 negotiation games against the set of baseline agents. Mean and 99% confidence interval are plotted based on 10 training iterations.

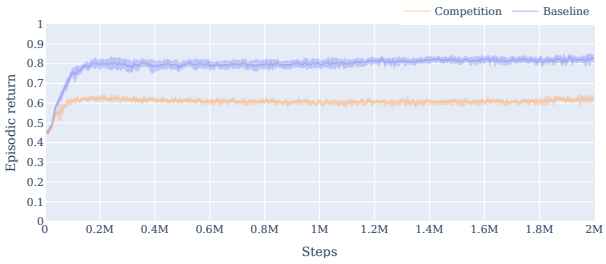
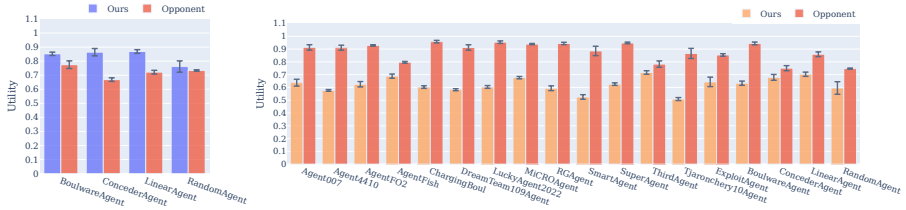


Figure 5.4: Mean and 99% confidence interval of episodic return during training of our GNN policy based on results from 10 different random seeds. The results from training against the baseline agents and training against the competition agents are plotted.



(a) (b)

Figure 5.5: Evaluation results of our GNN-based policy on randomly generated negotiation scenarios both against the set of baseline opponents (left) and against the full set of opponents (right). Results are obtained by evaluating each trained policy for 1000 negotiation games against the set of agents. Mean and 99% confidence interval are plotted based on 10 training iterations.

BASELINE OPPONENTS

We first train and evaluate against the set of baseline agents as described in Table 5.1. We train our method for 2 000 000 steps on 10 random seeds. The learning curve is plotted in Figure 5.4. Results are again obtained by running 1000 negotiation sessions against the set of baseline opponents, but this time, all negotiation scenarios have been randomly generated and were never seen before. We note that the observation and action space sizes are constantly changing. Results are plotted in Figure 5.5a.

As seen in Figure 5.5a, our method performs well against all baseline agents while negotiating on various structured negotiation scenarios it has never seen before. It is promising that an end-to-end learned GNN-based policy appears to generalise over such different scenarios.

COMPETITION OPPONENTS

We now repeat the experiments, but increase the set of agents we negotiate against. More specifically, we add the agents of the 2022 edition of the Automated Negotiation Agents Competition (ANAC)[2]. The learning curve and results are plotted in Figure 5.4 and Figure 5.5b, respectively.

The results show much lower performance against all opponents, including those previously outperformed. Our current method of encoding the observations and design of the policy likely leads to limited capabilities of learning opponent characteristics. Past work has shown that adapting to opponents is important to improve performance [76, 137, 123], which is currently impossible. However, this goes beyond the core contribution of this work, which is about handling different-sized negotiation scenarios in end-to-end RL methods. We discuss potential solutions in Section 5.5.

5.5 CONCLUSION

We developed an end-to-end RL method for training negotiation agents capable of handling differently structured negotiation scenarios. We showed that our method

[2] https://web.tuat.ac.jp/~katfuji/ANAC2022/

performs as well as a recent end-to-end method that is not transferable beyond a single fixed negotiation scenario. We see the latter as a restriction since, in real-world applications, it would be unlikely to encounter the exact same negotiation scenario more than once.

In this chapter, we have demonstrated how the difficulty of dealing with changing negotiation scenarios in end-to-end RL methods can be overcome. Specifically, we have shown how an agent can learn to negotiate on diverse negotiation scenarios in such a way that performance generalises to never-before-seen negotiation scenarios. Our method is conceptually simple compared to previous work on reinforcement learning in negotiation agents. Our agent performs well against strong baseline negotiation strategies, but leaves room for improvement when negotiating against a broad set of highly competitive agents.

Our approach is based on encoding the stream of observations received by our agent into a graph whose node features are designed to capture historical statistics about the episode. This manual feature design likely leads to information loss and goes against the end-to-end aim of our approach. For example, the expressiveness of history is limited, as the graph only encodes the last offer and frequency of offers. This likely also causes limited adaptivity to a broad set of opponent strategies, which in turn may well cause the low performance observed in Figure 5.4.2.

We note that, due to the competition setup of ANAC, competitive agents often play a game of chicken. Performing well against such strategies means that a policy must also learn this game of chicken. This can be challenging for RL, due to exploration problems, as it must learn a long sequence of relatively meaningless actions before having a chance to select a good action. We could attempt to improve upon this, but it might be more beneficial to prioritize mitigating this game of chicken behaviour, as it is inefficient and (arguably) undesirable.

The negotiation scenarios we generated have additive utility functions and outcome spaces that are comparable in size and competitiveness to the benchmarks used in the ANAC competition. Real-world negotiation scenarios, however, can have huge outcome spaces [82]. Our designed policy can be applied to larger scenarios without increasing the trainable parameters, and the effects on the performance of doing this should be investigated in future work.

Further promising avenues for future work include extending end-to-end policies with additional components that, e.g., learn opponent representations based on the history of observations in the current or previous encounter. Changing a negotiation strategy based on the opponent characteristics has been shown previously to improve performance [76, 137, 123], but is likely difficult to learn through our current policy design. Furthermore, improving our method to handle continuous objectives would eliminate the necessity of discretizing them.

Overall, we believe that the work in this chapter is a substantial step towards the effective use of end-to-end RL for the challenging and important problem of training negotiation agents whose performance generalises to new negotiation scenarios and opens numerous exciting avenues for future research in this area.