Learning in automated negotiation

Renting, B.M.

2

Background

This chapter contains background knowledge for readers who are less familiar with some of the topics discussed in this dissertation. We start with an introduction to automated negotiation (Section 2.1) and further discuss algorithm configuration (Section 2.2), algorithm selection (Section 2.3), and reinforcement learning (Section 2.4).

2

2.1 AUTOMATED NEGOTIATION

The research topic of automated negotiation studies autonomous agents that participate in a negotiation game [50]. Here, negotiation can be seen as a distributed search through a space of potential outcomes [79]. The software agents used in this context can negotiate with other agents in multi-agent systems or with humans in human-agent negotiations. Automated negotiation combines concepts and insights from multiple disciplines, such as artificial intelligence, game theory, and social psychology [79].

In this dissertation, we focus exclusively on bilateral negotiation between two autonomous agents. We define a negotiation game as a tuple $N = \langle \mathcal{I}, \Omega, \mathcal{U}_\mathcal{I} \rangle$, where $\mathcal{I}$ denotes the set of agents, Ω denotes the set of potential outcomes the agents negotiate over, also known as the domain or outcome space, and $\mathcal{U}_\mathcal{I}$ is the set of utility functions for agents $\mathcal{I}$ such that $\mathcal{U}_\mathcal{I} = \{u_i | i \in \mathcal{I}\}$. $u_i : \Omega \rightarrow [0,1]$ is the utility function of agent i over the outcome space where higher utilities indicate more desirable outcomes. From the perspective of an agent, we often refer to its own utility function with u and to the opponent's utility function with u_o. How agents communicate to agree is specified through a negotiation protocol that dictates the rules of encounter and permissible actions.

2.1.1 NEGOTIATION PROTOCOL

Communication between agents is required to enable a negotiation. Humans primarily negotiate using natural language. This is also possible for negotiation agents, but adds additional challenges in generating and interpreting natural language [128, 95, 67, 90, 98]. Most of the work in automated negotiation uses a negotiation protocol that dictates what actions agents can perform and when. A few examples of protocols are:

- The Ultimatum Game [24]: One agent makes an offer and the other agent may only accept or refuse it, after which the negotiation is ended.
- Alternating Offers Protocol (AOP) [132]: Two agents take turns in making offers. An extended version of this protocol that supports more than two agents is the Stacked Alternating Offers Protocol (SAOP) [7].
- Monotonic Concession Protocol [129]: Requires agents to make progressively more attractive offers to their opponents.
- Multiple Offers Protocol for multilateral negotiations with Partial Consensus (MOPaC) [110]: Allows for multilateral negotiations where only a subset of negotiating agents can reach an agreement, enabling partial consensus through bidding, voting, and opt-in phases.

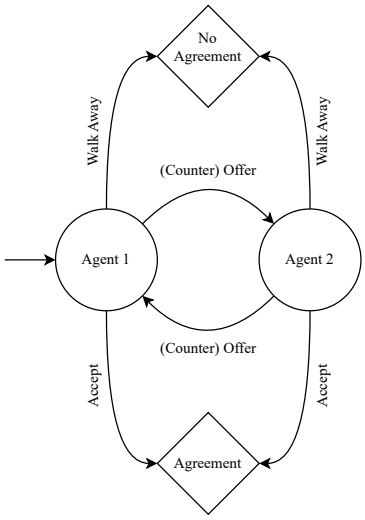

Figure 2.1: Visualisation of the Alternating Offers Protocol (AOP). Agent 1 starts the negotiation by making an offer. Note that the action "Accept" is inaccessible in the first step.

- Simulated Annealing Protocol [87]: A negotiation approach where agents or a mediator use simulated annealing techniques to explore the outcome space, allowing temporary concessions to escape local optima and ultimately achieve near-optimal agreements.

The choice of protocol can significantly impact negotiation dynamics and outcomes. For instance, the Alternating Offers Protocol (AOP) allows for more complex strategies and learning during negotiation, while one-shot protocols may encourage more truthful revelation of preferences.

In this dissertation, we exclusively use the AOP, due to its simplicity and its wide use in the automated negotiation literature. Here, agents take turns making a (counter) offer, accepting the opponent's offer, or walking away from the negotiation. The protocol is visualised in Figure 2.1. A (counter) offer is made by selecting one of the potential outcomes $\boldsymbol{\omega} \in \Omega$ and sending it to the opponent. A deadline T is imposed on the negotiation to prevent agents from negotiating indefinitely. Such a deadline is defined either in a maximum number of rounds that the agents can negotiate for or in seconds of wall-clock time or a combination of the two. The wall-clock time limit is often used as searching large (often combinatorial) outcome spaces for suitable candidate offers can take a long time.

The negotiation ends with an agreement if one of the agents accepts the offer received from the opponent. Both agents then receive a pay-off according to their (discounted) utility function. The negotiation ends without an agreement if one of the agents decides to walk away or if the deadline passes. In that case, agents are awarded their reservation value or 0 if there is no reservation value. The reservation value can be used to represent a Best Alternative To a Negotiation Agreement (BATNA) [53], e.g. an agreement reached with another negotiation partner, or

deciding to build something yourself instead of negotiating with a contractor.

2.1.2 NEGOTIATION SCENARIO

A negotiation scenario is a combination of an outcome space and utility functions that reflect the preferences of the agents. The outcome space generally consists of a set of objectives, known as issues $B = \{1, \cdots, n\}$, that are being negotiated. For example, in a GPU compute node purchase negotiation, issues might include price, support period, GPU type, number of GPUs, and delivery date. Issues can be:

- Categorical (e.g., GPU type: A6000, A100, or H100)
- Continuous (e.g., price of the compute node)
- Integer (e.g., number of GPUs: 2, 4, or 8)

We focus solely on categorical issues in this dissertation, which is the most general type, as continuous issues can also be discretised. The set of possible values for an issue $b \in B$ is denoted as Ω_b. The Cartesian product of all the issue values comprises the total outcome space $\Omega = \Omega_1 \times \cdots \times \Omega_n$ of the negotiation scenario. During a negotiation, the agents attempt to agree upon an outcome $\boldsymbol{\omega} = (\omega_1, \omega_2, \cdots, \omega_n) \in \Omega$, where n is the number of issues and $\omega_b \in \Omega_b$. Simple scenarios might have 2–3 issues with a small number of categorical values each. Complex scenarios can have dozens of issues, continuous values, and large numbers of categorical issue values, increasing the computational complexity of searching for suitable outcomes in the outcome space.

UTILITY FUNCTION

Utility functions are mathematical representations of an agent's preferences over the outcome space. They map potential outcomes to a real number, indicating the relative desirability of different outcomes. Both agents have a utility function $u_i : \Omega \to [0, 1]$, where 1 is the best possible utility for an agent. Such utility functions can be complex non-linear functions that capture interdependencies between issues, but in this dissertation, we focus on linear additive utility functions like the one shown in Equation 2.1. Here, weights are assigned to all values and issues through weight functions $w : B \mapsto [0, 1]$ and $w_b : \Omega_b \mapsto [0, 1]$ such that $\sum_{b \in B} w(b) = 1$, $\max_{\omega_b \in \Omega_b} w_b(\omega_b) = 1$, and $\min_{\omega_b \in \Omega_b} w_b(\omega_b) = 0$.

$$u(\omega) = \sum_{b \in B} w(b) \cdot w_b(\omega_b) \tag{2.1}$$

RESERVATION VALUE & DISCOUNT FACTOR

A negotiation scenario can optionally be extended with a reservation value and a discount factor. A reservation value is specified per agent and is the utility that an agent obtains when they fail to reach an agreement. This can be used to simulate the Best Alternative to a Negotiated Agreement (BATNA) [53], which is a commonly used concept in negotiation literature. This can be used to, for example, simulate an alternative agreement that is achieved with another party.

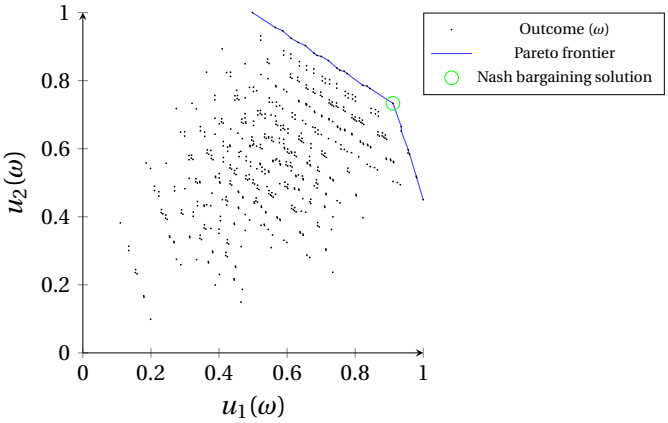

Figure 2.2: Visualisation of example negotiation scenario. The dots represent outcomes, which are plotted according to their utility value for two agents. The blue line connects the Pareto efficient possible outcomes, which form the Pareto frontier. The Nash Bargaining solution, the possible outcome that maximises the product of utilities, is circled in green.

The discount factor can be used to apply time pressure in a negotiation game. Here, the obtained utility is discounted as follows:

$$u_i^{\gamma}(\boldsymbol{\omega}) = (\gamma)^t \cdot u_i(\boldsymbol{\omega}) \tag{2.2}$$

where u_i is given in Equation 2.1, $\gamma \in [0,1]$ is the discount factor and t is a discrete time step variable. We do not consider reservation values and discount factors as the increased complexity they cause is outside of the scope of this dissertation.

EXAMPLE SCENARIO

To provide some intuition, we visualise an example negotiation scenario with categorical issues in Figure 2.2. All possible outcomes of the scenario are plotted based on their utility value for agent 1 (x-axis) and agent 2 (y-axis). We also visualise the Pareto frontier, which is the set of possible outcomes that are not strictly dominated by another possible outcome based on its utility value for both agents. Finally, we visualise the Nash bargaining solution [111], which is the outcome that maximises the product of utilities of all involved agents and is often used as a reference outcome for performance analysis. The Nash bargaining solution is defined as:

$$\omega_{Nash} \in \arg\max_{\omega \in \Omega} (u_1(\omega) \cdot u_2(\omega)) \tag{2.3}$$

2.1.3 NEGOTIATION AGENTS

Negotiation strategies determine an agent's behaviour during the negotiation process within the used protocol. They encompass decision-making about which offers to make, when to accept offers, and how to model the opponent's behaviour and respond to it. Strategies are at the core of automated negotiation research, with a

wide variety of approaches developed over the years. A commonly used framework to structure components of negotiation strategies is the Bidding, Opponent modelling, and Accepting (BOA) framework [8]. Although we will not always use this framework in this dissertation, we will use it in this section to provide insights into negotiation strategies.

BIDDING STRATEGY

The bidding strategy decides which offers to make. This involves balancing between pursuing the interests of the agent (e.g. maximising utility) and making concessions to reach an agreement. Common approaches include:

- Time-dependent tactics (see, e.g. [48], [51]): Concession is based on the remaining time.
  - Boulware: Concedes slowly at first, then rapidly as the deadline approaches.
  - Conceder: Makes large concessions early.
  - Linear: Concedes at a constant rate.
- Behavior-dependent tactics (see, e.g. [4]): Adapt based on the opponent's actions. These might mimic, reciprocate, or exploit the opponent's concession pattern.
- Trade-off strategies: Attempt to generate offers of similar utility to the previous offer but more attractive to the opponent.

ACCEPTANCE STRATEGY

The acceptance strategy determines when to accept an offer. Common criteria include:

- Threshold-based: Accept if the offer's utility exceeds a certain threshold.
- Time-dependent: Lower the acceptance threshold as the deadline approaches.
- Aspiration-based: Accept if the offer exceeds the utility of the planned counter-offer.

Acceptance strategies are often combinations of these criteria or more complex designed heuristic-based methods. For a comparison of acceptance strategies, we refer the reader to Baarslag and Hindriks [17].

OPPONENT MODELLING

With opponent modelling, the agent attempts to learn about the opponent's preferences, which are commonly considered to be private information. Being able to accurately predict the opponent's utility function helps in finding mutually beneficial outcomes, i.e., find outcomes that are (close to) Pareto efficient, which generally improves the pay-off of the agent. Techniques to predict the opponent's utility function include:

- Frequency analysis: Infer the importance of outcomes based on how often they are offered.
- Regression: Assume the concession behaviour (e.g., Boulware) of the opponent and fit a regression model to the opponent's offers based on that concession behaviour.
- Bayesian learning: Fit a probability model to the observed offers of the opponent.

We refer the reader to Baarslag et al. [12] for a more elaborate review and comparison of opponent modelling techniques.

2.2 ALGORITHM CONFIGURATION

Negotiation agents commonly have parameters that influence the behaviour or strategy of the agent. Algorithm configuration, also known as parameter tuning or hyperparameter optimisation, is the task of selecting the best parameters for an algorithm to optimise its performance on a given set of problem instances. The need for algorithm configuration arose from the observation that many algorithms, particularly in areas such as optimisation and machine learning, have parameters that significantly affect their performance. Traditionally, experts set these parameters manually through a process of trial and error. However, as algorithms became more complex and included more parameters, manual tuning became increasingly time-consuming and often suboptimal [74]. Early approaches to configure parameters included simple techniques like grid search and random search [52]. However, these methods scale poorly with the number of parameters, also known as the curse of dimensionality [22]. This led to the development of more sophisticated, automated algorithm configuration approaches, of which we will mention a few.

2.2.1 PROBLEM DEFINITION

The algorithm configuration problem can be formally defined as follows [74]. Given an algorithm with a configuration space Θ, a set of problem instances $\mathcal{P}$, and a performance metric $m(\theta, p)$ for $\theta \in \Theta$ and $p \in \mathcal{P}$, find $\theta^* \in \Theta$ that maximises (or minimises depending on m):

$$\theta^* \in \arg\max_{\theta \in \Theta} \sum_{p \in \mathcal{P}} m(\theta, p) \tag{2.4}$$

CONFIGURATION SPACE

The configuration space Θ represents all possible parameter settings for the algorithm. It can include various types of parameters, such as;

- Categorical parameters (e.g., choice of heuristic).
- Continuous parameters (e.g., learning rate in neural networks).
- Integer parameters (e.g., population size in genetic algorithms).

The configuration space can be highly complex, with dependencies between parameters and varying impacts on algorithm performance.

PERFORMANCE METRICS

The choice of performance metric $m(\theta, p)$ is crucial and depends on the specific application. Common metrics include;

- Solution quality: For example, the distance travelled for optimisation problems like the travelling salesperson problems or the prediction accuracy for machine learning methods on classification or regression tasks.
- Running time: Time taken to solve a problem or reach a certain quality threshold.
- Composite measures: e.g., PAR (Penalised Average Running time), which assigns a penalty to unsuccessful runs in, e.g., Boolean satisfiability or travelling salesperson problems.

2.2.2 CONFIGURATION METHODS

Several methods have been developed for automated algorithm configuration. Two separate parts within these methods can be identified: how new configurations are selected for evaluation and how a set of configurations is compared. In the following, we briefly outline examples of prominent, high-performance configuration approaches:

ParamILS uses Iterated Local Search (ILS) to find high-performance parameter settings for a given target algorithm. It navigates the configuration space by iteratively applying a local search phase, typically involving single parameter changes to find improvements and perturbation steps to escape local optima. Key features include always accepting better-performing configurations and using random restarts for diversification.

- ParamILS [74] uses Iterated Local Search (ILS) to find high-performance parameter settings for a given target algorithm. It navigates the configuration space by iteratively applying a local search phase, typically involving single parameter changes to find improvements and perturbation steps to escape local optima. Key features include always accepting better-performing configurations and using random restarts for diversification.
- F-Race [25] is a statistical racing algorithm that eliminates suboptimal configurations by first using the Friedman test to detect significant overall performance differences among remaining candidates across multiple instances. If such differences exist, it then performs pairwise comparisons against the best-performing configuration (incumbent) to discard those that are statistically significantly worse. This saves computational budget, as not all configurations have to be tested on the full target instance set. The set of configurations to test can be selected either manually, as a grid search, or at random. Balaprakash et al. [20] extended upon F-Race by implementing

it as a model-based search [166], which iteratively models and samples the configuration space in search of promising candidate configurations.

- Sequential Model-based optimisation for general Algorithm Configuration (SMAC) [75] builds a random forest model to predict algorithm performance based on parameter settings and instance features. It uses this model to find promising configurations to evaluate. It also incorporates an early elimination mechanism for new configurations by comparing them with a dominant incumbent configuration on individual problem instances.

- Gender-based Genetic Algorithm (GGA) [3] uses a genetic algorithm approach. It maintains a population of configurations, evolves them over generations, and incorporates gender separation to maintain diversity.

- Bayesian Optimisation HyperBand (BOHB) [47] is designed for hyperparameter optimisation of machine learning algorithms. It combines Bayesian optimisation with hyperband [96], where Bayesian optimisation is used to find promising configurations and multi-armed bandit strategies are used to allocate resources to evaluate configurations.

2.2.3 INSTANCE FEATURES

Some model-based configuration methods (e.g., SMAC and BOHB) allow the use of problem instance features to guide the search process. These features aim to capture relevant properties of the instances to better model the relation between parameter settings, problem instances, and performance. Examples of such features are:

- SAT solving: Number of variables, clause-to-variable ratio, etc.

- Classification tasks: Dataset size, number of features, class distribution, etc.

- Optimisation tasks: Dimensionality, constraint density, objective function properties, etc.

- Negotiation games: Size of the outcome space, average utility of outcomes, last obtained utility against opponent, etc.

2.3 ALGORITHM SELECTION

It is a common observation that a single negotiation strategy does not necessarily work well in every negotiation game [99]. This raises the question of whether selecting between negotiation strategies depending on the negotiation game could improve performance. Algorithm selection is the problem of choosing the best algorithm from a set of candidates for a given problem instance. It is based on the idea that no single algorithm is superior for all problem instances and aims to exploit the complementary strengths of different algorithms. The algorithm selection problem was first formally described by Rice [127], who recognised that for many computational problems, different algorithms performed best on various

instances and proposed a framework for selecting the best algorithm based on features of the problem instance.

The research area gained significant traction with the advent of portfolio-based approaches in areas such as Boolean satisfiability (SAT) solving and Constraint Programming (CP). Notable early systems include SATzilla [161] for SAT solving and CPHydra [113] for CP.

2.3.1 PROBLEM DEFINITION

The per-instance algorithm selection problem can be formally defined as follows. Given a set of algorithms $\Psi = \{\psi_1, \psi_2, \cdots, \psi_n\}$, a set of problem instances $\mathcal{P}$, a performance metric $m(\psi, p)$ for $\psi \in \Psi$ and $p \in \mathcal{P}$, and an instance feature mapping $f : \mathcal{P} \mapsto \mathbb{R}^k$ (Section 2.2.3) that characterise each problem instance p, find a selection mapping $M : f(\mathcal{P}) \mapsto \Psi$ that optimises the performance metric on the set of problem instances:

$$\min_{M} \sum_{p \in \mathcal{P}} m(M(f(p)), p) \tag{2.5}$$

2.4 REINFORCEMENT LEARNING

The negotiation game defined in Section 2.1 can be formulated as a Markov Decision Problem (MDP). We discuss the similarities in Section 2.4.2. This enables the usage of Reinforcement Learning (RL) in negotiation games. RL is a form of machine learning where an agent learns by interacting with an environment. Unlike supervised learning, where the agent learns from labelled examples, or unsupervised learning, where the agent finds patterns in unlabeled data, RL agents learn from the consequences of their actions.

The research area gained significant momentum with the development of algorithms like Q-learning [157] and the publication of a book by Sutton and Barto [146]. More recently, the combination of RL with deep learning, known as deep reinforcement learning, has led to breakthroughs in areas such as game playing (e.g., AlphaGo [141]) and robotics [89].

2.4.1 MARKOV DECISION PROCESS

RL problems are typically formalised as Markov Decision Processes (MDPs). An MDP is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto p(\mathcal{S})$ denotes the transition function, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ the reward function. The transition function describes how the environment transitions to a new state based on the current state the environment is in and the action taken. The reward function describes the reward that the agent obtains by taking an action in the state that the environment is in.

The goal in an MDP is to find the optimal policy $\pi^* : \mathcal{S} \mapsto \mathcal{A}$ that maximises the expected cumulative discounted reward:

$$\mathbb{E}_{\pi, \mathcal{T}} \left[\sum_{k=0}^{H-1} \gamma^t \cdot \mathcal{R}(s_t, a_t) \right] \tag{2.6}$$

Table 2.1: Similarities between a negotiation game and a Markov Decision Process (MDP). Each row describes a similar concept.

Negotiation game		**MDP**	
Utility	u	Reward	$\mathcal{R}$
Outcomes	Ω	Actions	$\mathcal{A}$
Discount	γ	Discount	γ
Deadline	T	Horizon	H
Time step	t	Time step	t

where H denotes the horizon of the MDP (the number of rounds we select an action), $\gamma \in [0,1]$ is a discount factor that discounts future reward, and t is the time step.

2.4.2 MDPS AND AUTOMATED NEGOTIATION

The negotiation game, as defined in Section 2.1, shows similarities with the definition of an MDP. We provide an overview of similarities in Table 2.1. Although unifying the notation is possible, we intentionally kept them separate to maintain the relation in this dissertation to the specific literature of the automated negotiation and reinforcement learning communities. We mention this relation here to make the reader aware of the close resemblance between both research areas.