Noname manuscript No.
(will be inserted by the editor)

Optimal Time-based Strategy for Automated Negotiation

Yasser Mohammad

Received: date / Accepted: date

Abstract Recent years are showing increased adoption of AI technology to automate business and production processes thanks to the recent successes of machine learning techniques. This leads to increased interest in automated negotiation as a method for achieving win-win agreements among self-interested agents. Research in automated negotiation can be traced back to the Nash bargaining game in the mid 20th century. Nevertheless, finding an optimal negotiation strategy against an unknown opponent with an unknown utility function is still an open area of research. The most recent result in this area is the Greedy Concession Algorithm (GCA) which can be shown to be optimal under specific constraints on both the negotiation protocol (non-repeating offers), opponent (static acceptance-model) and search space (deterministic time-based strategies). In this paper, we extend this line of work by providing an algorithmically faster version of GCA called Quick GCA which reduces the time-complexity of the search process from $O(K^2T)$ to $O(KT)$ where K is the size of the outcome-space and T is the number of negotiation rounds allowed. Moreover, we show that GCA/QGCA can be applied in a more general setting; Namely with repeating-offers protocols and to search the more general probabilistic time-based strategies. Finally, we heuristically extend QGCA to more general opponents with general time-dependent acceptance-model and negotiation settings (real-time limited negotiations) in three steps called QGCA$^+$, PBS, and PA that iteratively and greedily modify the policy proposed by QGCA$^+$ applied to an approximate static acceptance model. The paper evaluates the proposed approach empirically against state of the art negotiation strategies (winners of all relevant ANAC competition winners) and shows that it outperforms them in a wide variety of negotiation scenarios.

Keywords Automated Negotiation · Concession Policy · Simulated Annealing

This work was conducted at NEC Corporation's Global Innovation Unit.

Y. Mohammad
NEC CORPORATION, Japan and Assiut University, Egypt
Tel.: +81-80-5652-4120
E-mail: y.mohammad@nec.com, yasser@aun.edu.eg

1 Introduction and Background

Automated negotiation has a long history. Early work in the field can traced back to Nash's bargaining game [1]. Interest in automated negotiations is seeing a renewed interest from the research community and industry due to the wider adoption of AI in businesses in recent years. Applications of automated negotiation include training students on negotiation skills [2], assignment of WiFi channels [3], and industrial supply chains [4].

Negotiation is a process of achieving agreement between self-interested agents using a *negotiation protocol* that defines the actions available to each negotiation agent (called *negotiator or agent* hereafter). An *agent* is defined in this work as an entity that can act in an environment. A *self-interested* agent, is an agent that acts to *maximize* some internal quantity (its *utility*) [5].

A negotiation scenario is defined by a set of *issues*, negotiators and their associated utility functions and the negotiation *protocol*.

Several protocols have been proposed over the years. The simplest and most widely used protocol for bilateral negotiation is the Alternating Offers Protocol (AOP) — sometimes called the bargaining protocol — which is an unmediated protocol allowing two agents to exchange offers until one of them accepts the other agent's offer, a timeout condition is met or an agent leaves the negotiation. This protocol can be traced back to the work of Rubinstein [6]. The Stacked Alternating Offers Protocol (SAOP) extends AOP to negotiations with more than two parties [7].

There is no known optimal strategy for general automated negotiations in realistic scenarios characterised by an unknown opponent strategy, partner utility function and a general domain [8,9]. Nevertheless, the research community in automated negotiation have produced a wide variety of negotiation *strategies* to control negotiators' behavior [8]. These methods can be classified into exact solutions of simplified problems [9] and heuristic solutions to the most general case [10].

To simplify the presentation, we focus on bilateral negotiations in most of this paper. Extensions to multilateral negotiations is easy to achieve and discussed briefly near the end.

To make the problem manageable, we assume that the negotiator has access to an *acceptance model* that describes the probability of acceptance of any outcome by the opponent over the course of the negotiation. This assumption is justified in industrial applications in which the negotiator conducts negotiations *repatedly* against the same opponents with similar utility functions. For example, in a supply chain management scenario, negotiations are conducted repeatedly with all suppliers over the same products and are expected to have different yet related utility functions. In this and similar scenarios, learning an acceptance model for each opponent can be achieved easily.

Moreover, we limit our attention to negotiation time-based strategies (i.e. negotiation strategies that depend only on the relative time of the negotiation session). This restriction reduces the search space of possible strategies enough for us to provide theoretical guarantees. We show empirically that the strategies learned by the proposed method can achieve state-of-the-art performance in terms of expected utiltiy.

The main idea behind the proposed method is to utilize fast $O(1)$ operations for evaluating the effect of different manipulations of a given strategy on the expected utility to provide an efficient solution which is guaranteed to be optimal for

static acceptance models and is an approximate solution for *general time-dependent* acceptance models.

An earlier version of this manuscript [11] appeared in the The 23rd International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2020). The main contributions of that earlier version is the introduction of QGCA+, an algorithm for finding optimal deterministic time-based policies for static acceptance models in $O(KT)$ operations compared with $O(K^2T)$ operations for the fastest known previous work, where K is the size of the outcome space and T is the negotiation time limit. Another contribution on that earlier manuscript was the introduction of a heuristic (PA) for finding an approximately optimal deterministic time-based policy for general acceptance models in $O(KT + T^2 + RT)$ operations where $R \gg T$.

Adding to these contributions, the current manuscript extends this work to probabilistic policies by proving that the optimal deterministic policy is the same as the optimal probabilistic policy and to negotiations with nonzero reservation value (i.e. when agreement is not always better than disagreement). Secondly, the current manuscript provides full proofs of all utility function operations used in QGCA+ and PA. Thirdly, we extend the applicability of the proposed method to real-time limited negotiations, and provide a version of the proposed method with less dependency on the negotiation domain called GPA hereafter. Finally, the evaluation was extended by comparing the proposed method with the top performing negotiation agents in the International Automated Negotiating Agents Competition (ANAC) [12,13] since 2010 up to the latest comparable iteration available in GENIUS platform [14].

The rest of this paper is organized as follows: Section 2 presents the GCA algorithm as the most related previous work. Section 3 introduces the reader to the negotiation problem focusing on acceptance models and agent behavior. and describes the notation used in this work ending with a rigorous definition of our research problem. Section 4 introduces the Greedy Concession Algorithm (GCA) which is the basis of our work. Section 5 introduces the proposed efficient operations for evaluating effects of different manipulations on the expected utility of a given policy and Section 6 builds upon this to provide a faster version of GCA for static acceptance models and an approximate solution for general acceptance models. Two evaluation experiments are reported in Section 7. Limitations and future directions are then discussed in Section 8. The paper is then concluded.

2 Related Work

Literature in automated negotiation is vast and extends to several decades. In this section, we discuss some of the most relevant works to the proposed approach. This litareature review does not try to be exhaustive but attempts to position the proposed method within existing work.

Automated negotiations have been modeled as games since the early days of Nash and Robinstein [1,6]. Recently, de Jonge and Zhang proposed a Game Description Language (GDL) formulation (GDL) of the negotiation problem that can represent both linear and nonlinear utility functions. Moreover, the proposed method was shown theoretically to be able to explore the set of possible agreements of any negotiation domain. The paper goes on to show empirically that generic negotiation algorithms can be used to explore this space using only the GDL description [15].

Some of the proposed methods for bidding strategies in automated negotiation are domain specific. For example, Chakraborty et al. proposed a method to reach agreements in the smart energy domain that are shown to approach the Nash solution [16]. The offering strategy used in this work is a variation of the time-based boulware strategy (called Aspiration here after). Compared with this strategy, the proposed method does not assume that the opponent is using the same strategy as the agent. Moreover, the goal of maximizing the agent's own utiltiy is different from the goal of approaching a *fair* solution used in this work. Empirical comparison against the Aspiration offering strategy reported in Section 7 show that the proposed method outperforms Aspiration in terms of the agent's own utility which is the focus of this work.

Developing an effective offering strategy for automated negotiation agents is one of the core challenges in automated negotiation [8]. Several approaches have been proposed in the past decade to solve this problem. Most of these approaches are heuristics in the sense that they are not accompanied with any theoretical guarantees on performance.

This result, while proposed a method based on a multiarmed bandits formulation of repeated negotiations. This method uses ϵ-greedy for selecting between different predefined base offering strategies improving its expected utility over repeated encounters with the same agent. Other than having no theoretical guarantees on performance, this method assumes that a set of *good-enough* offering strategies is available to the agent.

Recently, Sengupta et al. proposed an RL based method that tries to learn empirical best responses against a set of opponents then uses a reviewer module to extend an existing switching agent if the performance of the new learned response was better than existing strategies by some predefined threshold [17]. This approach also utilizes a a classifer to switch in real-time class between the set of predefined base strategies.A key weakness of this method is the requirement to have active sessions with different opponents while training the base RL best response strategies (to create rollouts). The proposed method avoids this need and can be used to learn the acceptance model (the sole representation of the opponent in the proposed method) from existing negotiation logs with no need for direct repeated encounters.

In this work, we are more interested in finding an *optimal* or approximately optimal solution to the OPL problem which constraints the search space for policies enough to allow for theoretical advancement. The most related work to the proposed method is the Greedy Concession Algorithm (GCA) proposed by Baarslag et al. which provides an optimal solution for the special case of static acceptance models (SAM) [9], deterministic policy and with the further constraints that outcomes cannot be repeated within the policy (i.e. $i \neq j \rightarrow \pi_i \neq \pi_j$). Due to its importance as the baseline for our approach, we dedicate the following subsection to describing this approach.

3 Automated Negotiation

This section introduce the automated negotiation problem in general and defines the main components of automated negotiation: the outcome space, the negotiation protocol, the agent and acceptance model and ends with the problem statement.

3.1 Outcome Space

The space of all possible outcomes of a negotiation is called the *outcome-space* (Ω). Each element of this space is called an *outcome* ($\omega \in \Omega$). In most cases, the outcome-space is a multidimensional domain. Each dimension of the outcome-space is called a negotiation *issue* and has its own domain of possible values. For example, a negotiation between two companies about trade in a predefined product may have the following issues: price in dollars, quantity, color as one of ten predefined colors, and delivery date as some day of the next week, and a penalty for failure to deliver the agreed upon products in percentage of the unit price. This negotiation has five issues (price, quantity, color, delivery date, and penalty). Price and quantity, and delivery date are examples of nominal discrete issues. Color is an example of a categorical discrete issue while penalty is an example of a real-valued issue.

As clear from the previous example, a negotiation can in general have an infinite outcome-space. In this paper, we focus on negotiations with discrete outcome-spaces.

One possible outcome of any negotiation is *disagreement*. We represent this special outcome with the symbol ϕ and consider it a member of any outcome-space.

3.2 Negotiation Protocol

The second component of a negotiation scenario is the *negotiation protocol* which defines the actions available to each negotiator and timing of these actions.

Negotiation protocols can be *mediated* or *unmediated*. In a mediated, protocol, a third-party other than the agents seeking agreement is tasked with facilitating their interaction. For example, the single-text protocol is a mediated protocol in which a central agent generates a single offer that is then evaluated by the negotiators. This feedback is used by the mediator to modify the offer and the process is repeated until an agreement or a timeout condition is reached [18]. One problem with mediated protcols is that all negotiators need to *trust* the central mediator which is hard to achieve in real-world business environemnts.

In this paper, we consider the widely used Alternating Offers Protocols [6] for bilateral negotiations and the Stacked Alternating Offers Protocol (SAOP) [7] for multi-lateral negotiations. Both of these protocols are unmediated and use direct message exchanges between negotiators.

In AOP, negotiators take turns offering outcomes (called offers) to their partnres. An offer becomes a binding final agreement once the partner have *accepted* it. If any partner leaves the negotiation, it immediately fails. Moreover, a timeout condition is defined (e.g. a predefined number of *rounds* or seconds) and the negotiation fails also if this timeout condition is met without agreement. Failed negotiations result in *disagreement* and each agent receives its reservation value. Negotiations that result in an agreement assign to each agent the utility value of the resulting agreement.

SAOP extends AOP by defining an order over all partners. The first partner is asked to provide a first offer which is passed to the next partner. Each partner can either accept the offer it receives from the previous partner or rejects it and create a new counter offer which is passed to the next partner. This process is repeated until an offer is accepted by all partners (except the one offering it which is implicitly assumed to accept the offer).

In summary, we rely on the alternating offers protocol because of its simplicity (compared with argumentation based protocols), and having no requirement of a trusted third party (compared with mediated protocols).

3.3 Agents and Utility Functions

Every negotiating agent (negotiator) a has an associated *utility function* representing its preferences over different outcomes in the outcome space (including the disagreement outcome ϕ). The utility function maps outcomes into real numbers. In this work, we assume — without loss of generality — that the utility function is normalized between zero and one. Moreover, the utility value associated with disagreement is called the *reservation value* for the agent.

A commonly used architcture for automated negotiating agents is the BOA (Bidding-Opponent Model-Acceptance) strategy proposed in [8] In this architecture, the agent consists of three components: The acceptance strategy responsible of deciding whether an offer is to be accepted, the bidding strategy responsible of deciding the next outcome to offer to the opponent, and the opponent model responsible of keeping track of the opponent's behvior and the agent's believes about it. When an offer is received, it is passed to the opponent model to update agent's information about the opponent. It is then passed to the acceptance strategy which decides whether or not the offer is acceptable and — if it was not acceptable — the bidding strategy is called to decide the counter offer to be sent to the negotiation opponent.

This work relies heavily on the acceptance model (Section 3.4) which represents the *probability that the opponent would accept an offer*. This probability represents our agent's understanding of the opponent's *acceptance strategy*. Notice that we do not attempt to model the opponent's bidding strategy. This decision was made for two reasons: firstly, it is common to have a simple acceptance strategy and a complex bidding strategy [19] which makes it easier to capture the acceptance strategy of the opponent in a probabilistic form. By restricting our attention to the opponent's acceptance strategy, we restrict our seach space for the optimal policy enough to have a theoretically justified result. Secondly, basing decisions on the acceptance strategy alone makes our estimates of expected utilities in this paper a *lower limit* on the actual expected utility as the agent can simply ignore opponent offers with lower utility accepting only ones with higher utility compared with the expected utility estimate it has.

For the acceptance strategy, two flavors will be discussed. The first approach is to follow the offering strategy and accept only an offer if it is the same as the next offer in the offering strategy. This is the approach taken in [9] and is called *strict-policy-adherence* hereafter. Another possibility is to accept an offer if its utility is higher than the expected utility from following the offering strategy. This is called *opportunistic-acceptance* hereafter as it enables the agent to take advantage of unexpected opportunities in the form of offers with high utility from the opponent.

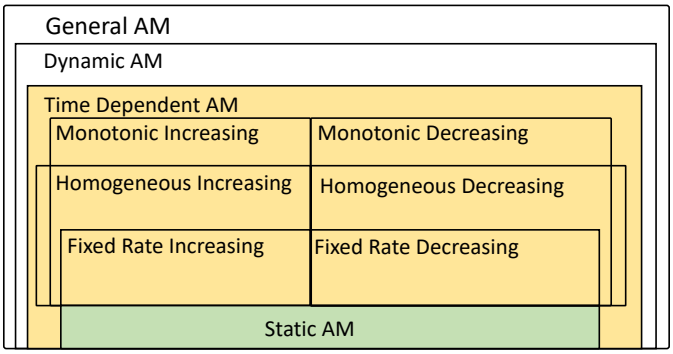


Fig. 1: A hierarchy of acceptance model. In this work, we provide an optimal algorithm for negotiating with opponents using static AMs and an effective heuristic for negotiating with opponents using a time-dependent AM.

3.4 Acceptance Models

When an agent repeatedly interacts with another agent, it can accumulate information about the likely behavior this that opponent. In this paper, we assume that this information is available in the form of an *acceptance model* (AM) which specifies the probability that the opponent will accept some outcome ω as $a(\omega)$.

Fig. 1 shows different types of acceptance models. The following paragraphs describe each type in this hierarchy and positions our work within it.

A general acceptance model can depend on negotiation time t, past behavior of both agents during the negotiation $H \in \Omega^t$ and external factors x with some domain $Dom(x)$:

$$a_{general} : \Omega \times [0, T] \times \Omega^t \times Dom(x) \rightarrow [0, 1]$$

A *dynamic* acceptance model is a special case of the general acceptance model in which there are no external variables which means that the probability of acceptance of an offer is only a function of time and previous offers *within the negotiation*:

$$a_{dynamic} : \Omega \times [0, T] \times \Omega^t \rightarrow [0, 1]$$

In this paper, we consider only acceptance models that do not depend on negotiation history (H) or external factors (x). This type of acceptance model is called *Time Dependent Acceptance Model* (TDAM) hereafter and can be represented as:

$$a_{tdam} : \Omega \times [0, T] \rightarrow [0, 1],$$

where $a(\omega, t) \in [0, 1]$ is the probability that outcome ω will be accepted at step t.

Earlier work in optimal offering for automated negotiation considered only a special case of TDAMs called *Static Acceptance Models* (SAM) [9]. This class of AMs models an opponent that has a fixed acceptance probability for each outcome independent of any other conditions:

$$a_{sam}(\omega, t) = c_\omega$$

where $0 \leq c_\omega \leq 1 \quad \forall \omega \in \Omega$. Due to its simplicity, it is possible to find *optimal* negotiation strategies against opponents using SAMs. Nevertheless, negotiators rarely — if ever — use SAMs as most negotiators tend to concede over the negotiation time [17].

It is important that when a negotiator employs a special type of AM (e.g. TDAM, SAM) for modeling the behavior of its opponents, *this assumption constraints the information it can learn about these opponents but it does not in any way constrain opponent(s) behavior.*

The static *acceptance model* cannot represent a negotiation opponent that changes its future behavior based on time or any other negotiation-related or negotiation-unrelated factors (e.g. offers exchanged, market condition).

A more flexible *acceptance model* is the homogeneously increasing model in which probability of acceptance of any given outcome goes up with time with factor $\gamma_{ij} > 0$ that depends only on time but not the outcome. Formally, a is a homogeneously increasing model iff

$$a(\omega, j) = \min(1, (1 + \gamma_{ij}) a(\omega, i)) \, \forall j > i.$$

Homogeneously increasing models represent a negotiation opponent under time-pressure. Yet, a more general *acceptance model* considered in this paper is the *monotonically increasing* model in which the only constraint is that the opponent does not decrease its acceptance probability with time. Formally, a is a monotonically increasing model iff $a(\omega, j) \geq a(\omega, i) \, \forall j > i$.

Finally a *general acceptance model* has no constraint on the relationship between acceptance probabilities of outcomes over the negotiation time and is capable of representing any acceptance policy.

Each one of these *acceptance model* types has a binary version in which $a(\omega, t)$ is constrained to be either zero or one and a *decreasing* version in which the acceptance probability decreases instead of increasing over time (e.g. modeling outside options). This type of *acceptance model* is useful for representing well-known negotiation strategies. For example time-based concession strategies (e.g. the well known boulware and conceder [8]), can exactly be modeled by a *binary* monotonically increasing model.

3.5 Problem Statement

A negotiation scenario is defined – from the point of view of an agent engaged in it – by a tuple $(\Upsilon \equiv (T, \Omega, u, a))$ where:

$T \in \mathbf{Z}$: The time-limit of the negotiation which can be specified as a number of rounds or a wall-clock time limit in a suitable time-unit[1].

$\Omega \equiv \{\omega\}$: Possible outcomes (assumed countable) where $|\Omega| \geq 2$. The special value ϕ represents the no-agreement outcome. For this rest of this paper we will let K be the cardinality of Ω.

[1] We start by assuming that the time-limit is specified as a number of rounds but will discuss extensions to wall-clock time limits in section 6.3.3.

$u : \Omega \rightarrow [0, 1]$ The utility function of the agent which maps every outcome to a real value representing the agent's utility of achieving this outcome. $u(\phi)$ is the *reservation-value* of the agent representing the utility of reaching no agreement.

$a : \Omega \times [0, T + 1] \rightarrow [0, 1]$ The acceptance model. A TDAM is assumed in this work. Static acceptance models are just a special case where $a(\omega, i) = a(\omega, j)\ \forall 0 \leq i, j \leq T$. Moreover, $\forall t \in [1, T + 1] \quad a(\phi, t) = 1$ representing the fact that the opponent has no choice if the agent decides to end the negotiation. We also know that $\forall \omega \in \Omega \quad a(\omega, T + 1) = 1$ representing the facts that the opponent has no choice if the negotiation timed out.

As explained in Section 3.3, the behavior of the agent is encapsulated by a *policy*. A probabilistic policy is defined as $\psi : [1, T + 1] \times \Omega \rightarrow [0, 1]$ (i.e. for each outcome and time, the policy gives the probability of offering this outcome at this step). A deterministic policy is a special case of a probabilistic policy and is defined as $\pi : [1, T + 1] \rightarrow \Omega$. For this class of policies, we can refer to the outcome at index i of policy π as π_i.

The problem targeted in this paper (called Optimal Policy Learning [OPL]) can be stated as:

Given a negotiation scenario (Υ), find a time-based policy (ψ^*) that maximizes the agent's expected utility ($\mathcal{EU}$). Formally:

$$\psi^* = \arg\max_{\psi} \mathcal{EU}(\psi|\Upsilon)$$

Throughout this paper, we will always refer to deterministic policies using π and to general probabilistic policies using ψ.

4 Greedy Concession Algorithm (GCA)

Algorithm 1 Greedy Concession Algorithm (GCA)[9]

1: $\pi \leftarrow <>$ ▷ Empty policy
2: **for** $k \leftarrow 1 : T$ **do**
3: $\quad \omega^* \leftarrow \arg\max_{\omega \in \Omega - \pi} \mathcal{EU}(sort_{\mathcal{EU}}(\pi \circ \omega))$
4: $\quad \pi \leftarrow sort_{\mathcal{EU}}(\pi \circ \omega^*)$ ▷ Operator o inserts an element in a set

The most relevant automated negotiation method for this work is the Greedy Concession Algorithm (GCA) [9] which is presented in Algorithm 1.

The most important contributions of GCA [9] were two proofs showing that the optimal deterministic policy for such acceptance models are always conceding policies and they can be be found by greedily building longer optimal policies given shorter ones.

A conceding policy π^c is defined as one satisfying the condition: $i < j \rightarrow u(\pi^c_i) \geq u(\pi^c_j)$. The greedy construction condition implies that the optimal policy of length T is included within the optimal policy of length $T+1$ *up to a permutation of outcomes.*

Using these two properties (and the aforementioned three constraints on the problem: SAM and deterministic policies with no repeated outcomes), Baarslag et al. proposed GCA which incrementally builds the optimal policy starting by the

optimal policy of length one known to be $\omega^* = \arg\max_\omega u(\omega)a(\omega)$. Algorithm 1 describes this approach.

The goal of GCA is to build a policy that is guaranteed to be optimal given a static acceptance model. In Line 1, the policy is initialized as an empty policy. The algorithm is a single loop (Lines 2-4) that incrementally adds outcomes to the current policy to build a one step longer policy until the required length (T) is reached (Line 2). In every iteration, GCA starts by generating all possible one-outcome extensions that have outcomes in decreasing order of utility. It then calculates the expected utility of each one of these policies and finds the outcome that leads to the extension with maximum expected utility (Line 3) using it as the new policy (Line 4). Because of the greedy construction and conceding policy lemmas, Baarslag et al. proved that this approach is optimal [9].

The time complexity of GCA is $O(K^2T)$ and its space complexity is $O(T)$ (See Appendix C.1). In this paper, we extend that work in three ways: Firstly, we show that GCA extends naturally to finding probabilistic optimal policies and requires no changes to handle policies with repeated offers. Secondly, we provide a faster approach for finding optimal policies for SAMs. Finally, we provide an approximate algorithm for finding optimal policies for TDAMs.

5 Policy Operations

GCA can find the optimal deterministic policy against an opponent using a static acceptance model. Nevertheless, GCA repeatedly calculates the expected utility from scratch (Algorithm 1) in every iteration. This can be avoided if we could find a way to calculate the *effect* of simple policy operations (e.g. inserting an offer, replacing an offer, etc) on expected utility efficeintly. This section defines these efficient policy opoerations for both TDAM and SAM. These operations are then used by the proposed algorithms described in Section 6.

Firstly, we note that there is no need for searching the much larger space of probabilistic policies based on Lemma 1.

Lemma 1 *The optimal probabilistic policy ψ^* is a deterministic policy (π^*).*

Proof The proof is straight forward: Given any probabilistic policy ψ, a set of supported policies Π_ψ is induced where $\pi \in \Pi_\psi$ is a deterministic policy each with a probability $p_\psi(\pi) = \prod_{k\in|\pi|}\psi(\pi_k)$ and $\sum_{\pi\in\Pi_\psi} p_\psi(\pi) = 1$.

Because the expectation function (Eq. 2) applies to each such deterministic policy π, we can calculate the expected utility given the probabilistic policy ψ as:

$$\mathcal{EU}(\psi) = \sum_{\pi\in\Pi_\psi} p_\psi(\pi)\mathcal{EU}(\pi) \tag{1}$$

Let $\pi^*_\psi \in \Pi_\psi$ be the policy that maximizes $\mathcal{EU}(\pi)$:

$$\pi_\psi* \equiv \arg\max_{\pi\in\Pi_\psi} \mathcal{EU}(\pi)$$

Given the $0 \le p(\pi) \le 1\ \forall\pi \in \Pi_\psi$ and that $\pi^*_\psi \in \Pi_\psi$, if follows that:

$$\mathcal{EU}(\psi) \le \mathcal{EU}(\pi_\psi*)$$

where equality is attained when:

$$p(\pi) = \begin{cases} 1 & \pi = \pi^*_\psi \\ 0 \text{ otherwise} \end{cases}$$

□

Given Lemma 1 we can focus completely on deterministic policies. For the rest of this paper, we will only consider deterministic policies.

The *expected utility* for a deterministic policy π can easily be found as:

$$\mathcal{EU}(\pi) = \sum_{i=1}^{T} u(\pi_i)a(\pi_i) \prod_{j=1}^{i-1} 1 - a(\pi_j) + u(\phi) \prod_{j=1}^{T} 1 - a(\pi_j) \tag{2}$$

We define the following components which will be used later in the proposed algorithm:

$P : [0, T+1] \to [0, 1]$ The aggregate multiplication of probability of rejection used in the evaluation of $\mathcal{EU}$.

$$P_i = \prod_{j=0}^{i-1} 1 - a(\pi_j) \tag{3}$$

$S : [0, T+1] \to [0, 1]$ The cumulative sum of the expected utility value.

$$S_i = \sum_{j=0}^{i} u(\pi_j)a(\pi_j)P_j = S_{i-1} + u(\pi_i)a(\pi_i)P_i \tag{4}$$

Using this notation, the expected utility $\mathcal{EU}(\pi)$ is simply S_T. Calculating the expected utility of a policy using Equation 2 is an $O(|\pi|) = O(T)$ operation. To simplify the notation further we use a^j_i to mean $a(\pi_i, j)$ and u_i to mean $u(\pi_i)$ when the policy π is clear from the context. Moreover, the time superscript will be removed for SAMs to emphasize the fact that the acceptance probability does not depend on time.

Table 1: Effect of different operations on the expected utility of a policy for a TDAM

Operation	Symbol	Effect on $\mathcal{EU}(\pi)$
Replacing π_i with ω	π^ω_i	$P_i(u(\omega)a(\omega, i) - u(\pi_i)a^i_i) + (S_T - S_i)\frac{a^i_i - A(\omega,i)}{1-a^i_i}$
Swapping π_i, π_j	$\pi_{i\leftrightarrow j}$	$u_i\left(a^j_i P_j \frac{1-a^i_j}{1-a^i_i} - a^i_i P_i\right) + u_j\left(a^i_j P_i - a^j_j P_j\right) + (S_{j-1} - S_i)\frac{a^i_i - a^i_j}{1-a^i_i}$
Swapping π_i, π_{i+1}	$\pi_{i\to 1}$	$u_i\left(a^{i+1}_i P_i(1-a^i_{i+1}) - a^i_i P_i\right) + u_{i+1}\left(a^i_{i+1}P_i - a^{i+1}_{i+1}P_{i+1}\right)$

Using P and S values, it is possible to find the change in expected utility for various manipulations of the policy π efficiently in $O(1)$ operations. Table 1 shows how to calculate the effects of replacing an outcome, and swapping two outcomes for TDAMs. These calculations simplify further for SAMs as shown in Table 2.

The first operation in Table 1 is the replacement of an outcome at a specific location of the policy (i) with another outcome (ω). This is referred to as π^ω_i. The effect of this operation on the expected utility of the policy can be calculated in

$O(1)$ operations by reusing the calculations encoded in S and P. The corresponding operation for SAMs is shown in Table 2. Note that for the spacial case when the acceptance probability of the outcome at location i in the policy is exactly 1, we have $S_T = S_i$ because the rest of the plicy does not matter (the opponent will certainly accept the outcome at step i). In this special case, the expresion can be simplified to $P_i(u(\omega)a(\omega, i) - u(\pi_i)a_i^i)$ which is true for both TDAMs and SAMs. The proof for this operation is given in Appendix A.1.

The second operation in Table 1 is swapping outcomes at locatins i, and j of the policy ($\pi_{i\leftrightarrow j}$) where $i < j$. As with the previous opration, the effecg of this operation on the expected utility can be calculated in $O(1)$ operations by utilizing S and P. The corresponding operation for SAMs is shown in the second row of Table 2. The proof for this operation is given in Appendix A.2.

A special case of the swapping operation is when $j = i + 1$ which is shown in the third rows of Table 1 and Table 2 for TDAMs and SAMs respectively ($\pi_{i\to 1}$). The form of this calculation can be found by subsituting $i+1$ for j in the equations for $\pi_{i\leftrightarrow j}$ shown in the second row of the corresponding table. The proof for this operation is given in Appendix A.3.

The main reason these calculations can be done in $O(1)$ operations is that replacement and swapping have only local effects on the index of different outcomes within the policy which in turn localizes their effects on P and S.

Insertion and removal of outcomes on the other hand cannot be done in $O(1)$ operations for TADMs except in the special case of SAMs. The effect of insertion on the expected utility of a policy for SAMs is shown in the final row of Table 2 and its proof is given in Appendix A.4.

Because our goal is to handle TADMs in general, we restrict our algorithms to use replacement and swapping only.

Table 2: Effect of different operations on the expected utility of a policy for a SAM

Replacing outcome at i with ω	π_i^ω	$P_i(u(\omega)a(\omega) - u(\pi_i)a(\pi_i)) + (S_T - S_i)\frac{a(\pi_i)-a(\omega)}{1-a(\pi_i)}$
Swapping outcomes at i, j	$\pi_{i\leftrightarrow j}$	$u_i a_i \left(P_j \frac{1-a_j}{1-a_i} - P_i\right) + u_j a_j (P_i - P_j) + (S_{j-1} - S_i)\frac{a_i - a_j}{1-a_i}$
Swapping outcomes at i, $i+1$	$\pi_{i\to 1}$	$P_i a(\pi_i)a(\pi_{i+1})(u(\pi_{i+1}) - u(\pi_i))$
Inserting outcome ω at location t	$\pi_{\omega@i}$	$u(\omega)a(\omega)P_i - a(\omega)(S_T - S_{i-1})$

6 Proposed Algorithms

Based on policy manipulations with efficient $\mathcal{EU}$ calculations presented in Section 5, we provide two algorithms to deal with SAMs and general TDAMs in this section. Both of these algorithms rely on the efficient calculation of policy operations presented in Table 2 and 1 to incrementally improve or construct policies.

6.1 Static Acceptance Models: QGCA

A faster version of GCA (called Quick GCA hereafter) that employs the swapping operation defined in Table 2 (specifically the insertion operation) is shown in Algo-

rithm 2. This algorithm is designed for static acceptance models. The main difference between QGCA and GCA lies in how the policy of some length is extended at every iteration. By keeping track of more information about the policy and applying efficient $O(1)$ operators to it, QGCA can find the exact policy found by GCA much faster. Note that both GCA and QGCA find optimal policies with the same expected utility by construction.

Algorithm 2 Quick Greedy Concession Algorithm (QGCA)

1: $\pi \leftarrow <>$ ▷ Empty *linked* list
2: $\mathcal{L}_\omega = 0 \forall \omega \in \Omega$ ▷ Initialize location of all outcomes
3: $S_{-1}, P_{-1} \leftarrow 0, 1$
4: **for** $k \leftarrow 1 : T$ **do**
5: $\quad d^*, \omega^* \leftarrow -\infty, \phi$
6: $\quad$ **for** $\omega \in \Omega$ **do**
7: $\quad\quad i \leftarrow \mathcal{L}_\omega$ ▷ Lookup location of insertion
8: $\quad\quad d_\omega \leftarrow S_{i-1} + (1 - a(\omega))(\mathcal{EU}(\pi) - S_{i-1}) + P_i \mathcal{EU}(\omega) a(\omega)$
9: $\quad\quad$ **if** $d_\omega \geq d^*$ **then**
10: $\quad\quad\quad d^*, \omega^* \leftarrow d_\omega, \omega$
11: $\quad \pi \leftarrow \pi \circ_{\mathcal{L}_{\omega^*}} \omega^*$ ▷ Insert best outcome in its correct place
12: $\quad$ Update $S, P, \mathcal{L}$
13: $\quad$ **if** no-repetition **then**
14: $\quad\quad \Omega \leftarrow \Omega - \{\omega^*\}$

The main idea of QGCA is to use the S, P arrays and the fast evaluation of the effect of outcome insertion (Table 2) to speedup the process of finding the optimal outcome to insert when increasing the policy length. To achieve that we further keep track of the *optimal* insertion location of each outcome within the policy. From the fact that the optimal policy must be a conceding policy, it is directly inferred that – in order to keep optimality – an outcome ω can be inserted in an optimal policy π at location k only if the following condition is satisfied:

$$u(\pi_i) \geq u(\omega) \qquad \forall i < k$$

We keep the first of these locations in a map called $\mathcal{L} : \omega \leftarrow [0, T]$. Updating this map is straightforward after an outcome is inserted and can be done in $O(T)$ operations.

The first step (Lines $6 - 11$) is to find an outcome to add to the existing policy (remember that in the first step this policy is empty). For each outcome we get the location to insert that outcome in the list (Line 8) which is known and stored in $\mathcal{L}$ because outcomes with higher utility appear first in the policy (Line 8). We then calculate the new expected expected utility ($\mathcal{EU}$) after this insertion (Line 9) using :

$$\begin{aligned} \mathcal{EU}(\pi \circ_i \omega) = S_{i-1} + (1 - a(\omega))(\mathcal{EU}(\pi) - S_{i-1}) \\ + P_i \mathcal{EU}(\omega) a(\omega) \end{aligned} \tag{5}$$

where $\pi \circ_i \omega$ is the policy that results from adding ω at location i, and $i \equiv \mathcal{L}_\omega$ is the correct location to insert ω in order to maximize $\mathcal{EU}$ according to the concession lemma [9].

We keep track of the outcome that leads to the maximum increase in $\mathcal{EU}$ (call it ω^*) (Line $10 - 11$). Once we have the best outcome (ω^*), we simply insert it into

the policy (Line 12) and update the $S, P, \mathcal{L}$ lists to reflect the new policy. Updating S and P is simple using the equations 3, 4 respectively. $\mathcal{L}$ is updated by increasing $\mathcal{L}_\omega$ by one for all outcomes ω with a utility less than the utility of ω^* and keeping the rest of *oc* without change (this is an $O(K)$ operations).

Finally, we remove the just-added outcome (ω^*) from the list of outcomes to consider in the future (Ω) if the protocol does not allow repetition of offers.

The time complexity for finding the optimal *deterministic time-based policy* using QGCA is $O(KT)$ for static models compared with $O\left(K^2T\right)$ for GCA (See Appendix C).

6.2 General Time Dependent Acceptance Models: PA

QGCA is only optimal for static acceptance models. See Appendix B for a proof by counter example that GCA and QGCA are not optimal for TDAM (or even for the special case of monotonic acceptance models). Even the simplest opponents cannot be modeled by these acceptance models as explained in Section 3.4. In this section, we provide a method for approximating the optimal deterministic policy against an agent with a time-dependent acceptance model. Extension to dynamic acceptance models is not discussed in this paper. . The proposed method consists of three steps: Firstly, we use QGCA on an approximate static acceptance model to generate an initial policy (QGCA+). Secondly, we apply a bubble-sort like algorithm to improve find a better permutation of the initial policy that needs not be monotonic in utility value (PBS). Finally, we apply a simulated annealing like approach to modify the permuted policy and find a better policy for the original non-static acceptance model (PA)

The first step in our approach is to approximate the time-dependent acceptance model with a static acceptance model and apply QGCA to it. This approach is called QGCA$^+$ hereafter. Several approaches can be devised for carrying out this approximation. We simply use a SAM and a TDAM for each scenario (Section 6.3.2) and use the SAM for this step. This means that QGCA$^+$ does not utilize the timing information. Call this initial policy π^c (for concession). As discussed earlier, the time-complexity of this step is $O(KT)$.

The second step is to improve this policy by finding the best permutation of π^c using a form of bubble sorting. Algorithm 3 gives the details of this process. The resulting policy is still not guaranteed to be optimal because it is restricted by the choices made by the QGCA$^+$ algorithm based on an approximation of the acceptance model. This policy is called π^b (for *bubble*). The time complexity of this step is $O(T^2)$. Note that this performance is only possible because we can find the change in expected utility for the swap in constant time and can also update the P and S matrices in constant time for this operation.

The final step is a form of simulated annealing starting from the result of PBS (π^b) resulting of the final policy π^a (for annealing). At every iteration r, A modification of the policy by changing one of its outcomes randomly is proposed π^ω_s. If the modification leads to a higher expected utility, the proposal is accepted, otherwise the proposal is accepted only with the probability $\exp^{-\delta/\tau(r)}$ where δ is the decrease in expected utility due to the proposed operation and τ is a monotonically decreasing temperature term. This process is repeated a predefined number of iterations R. This

Algorithm 3 Policy Bubble Sort (PBS)

$\pi \leftarrow \pi^c$ ▷ Start from the result of QGCA$^+$
for $r \leftarrow 1 : T$ **do**
  $\pi^- \leftarrow \pi$
  for $k \leftarrow 1 : T - 1$ **do**
    if $\mathcal{EU}(\pi_{k\to 1}) - \mathcal{EU}(\pi) > 0$ **then**
      $\pi \leftarrow \pi_{k\to 1}$
  if $\pi^- = \pi$ **then**
    return $\pi^b = \pi$
return $\pi^b = \pi$

algorithm is called *Policy Annealing* hereafter (PA) and is presented in Algorithm 4. The time complexity of this step is $O(RT)$ where R is the number of iterations.

The time-complexity of the complete system is $O(KT + T^2 + RT)$. By setting $R = KT$ which is what we used in our experiments, the total complexity becomes $O(KT^2)$ which is on the same order of magnitude as the original GCA algorithm – $O(K^2T)$ – but with the advantage of being usable for general TDAMs.

Algorithm 4 Policy Annealing

$\pi \leftarrow \pi^b$ ▷ Start from the result of PBS
for $r \leftarrow 1 : R$ **do**
  Randomly select a site s and an outcome $\omega \in \Omega - \{\pi_s\}$
  $\delta \leftarrow \mathcal{EU}(\pi_s^\omega) > \mathcal{EU}(\pi)$
  if $\delta > 0 \vee rand() > \exp^{-\delta/\tau(r)}$ **then**
    $\pi \leftarrow \pi_s^\omega$
  for $i \leftarrow 1 : T$ except s **do** ▷ Find the best permutation
    if $\mathcal{EU}(\pi_{i\leftrightarrow s}) > \mathcal{EU}(\pi)$ **then**
      $\pi \leftarrow \pi_{i\leftrightarrow s}$
return $\pi^a = \pi$

6.3 Extensions

As presented so far, the proposed method is domain dependent (as the acceptance model and utility function are both defined on the outcome-space), assumes that the agent has an *accurate* acceptance model of its partner, requires hard deadlines, and can only work for bilateral negotiations.

In this section, we discuss different extensions of the proposed methods to more complex — yet more realistic — negotiation scenarios by removing these restrictions. Each one of the following subsections discusses one of the aforementioned four extensions

6.3.1 Reducing Domain Dependency

Up until now, the proposed method generates a policy that is tied to both a specific acceptance model (representing an opponent strategy) and a specific domain (represented by Ω). This is enough for applications in which the agent is designed to

negotiate repeatedly with the same opponents in a single domain. Nevertheless, the proposed method can easily be extended to become applicable to multiple domains.

The key modification is in the way we model the opponent. Rather than using an acceptance probability mapping from the outcome space, we can use a more general input to the acceptance model.

The simplest such modification is to define the acceptance model as a mapping from the *opponent*'s utility of an outcome to the acceptance probability. This leads to:

$$\mathcal{EU}(\pi) = \sum_{i=1}^{T} u(\pi_i) a(u_{opp}(\pi_i)) \prod_{j=1}^{i-1} 1 - a(u_{opp}(\pi_i))$$

This assumes that the agent can estimate the utility of the opponent for every outcome in the outcome space. Several approaches to opponent modeling that can be used to achieve that have been proposed [20].

In this work, we use a slightly different approach that requires no extra modeling of the opponent (above learning their acceptance model) by defining the acceptance probability in terms of the agent's *own* utility function. This approach uses only publicly available information during the training phase.

$$\mathcal{EU}(\pi) = \sum_{i=1}^{T} u(\pi_i) a(u(\pi_i)) \prod_{j=1}^{i-1} 1 - a(u(\pi_i))$$

At the first glance, it seems unwarranted to use the agent's own utility value for modeling the opponent's acceptance probability. After all, the opponent will base its acceptance decision on either the outcome itself or its utility *for itself* and not for the agent. When considering a single domain, using the outcome itself or its utility for the agent as an input to the acceptance model are exactly equivalent up to outcomes with exactly the same utility for the agent. When working with multiple domains, as long as the correlation between the agent's and opponent's utility does not change much, using either of them should lead to approximately the same results. Using the agent's utility here has the obvious advantage of requiring no further modeling of the opponent.

Section 7 reports an experiment in which we show that this approach works surprisingly well for a wide variety of domains.

When using the agent's utility as the input to the acceptance model, we can also define the policy in terms of the agent's utility directly leading to:

$$\mathcal{EU}(\pi) = \sum_{i=1}^{T} \pi_i a(\pi_i) \prod_{j=1}^{i-1} 1 - a(\pi_j)$$

where $\pi[1, T] \rightarrow [0, 1]$ is now generating a utility level instead of an outcome. The agent will then have to solve the inverse utility problem by finding an outcome with a utility value matching the required level (which is always possible by construction because the optimization problem is still solved in terms of Ω).

This approach requires the agent to calculate the utility value of *all possible values* in advance in order to guarantee that for any utility value in π, a corresponding outcome exists. This is impractical for large outcome spaces. In such cases, it is possible to use a predefined fixed number of utility levels (say 100) and then solve the inverse problem of finding an outcome with a utility value near the policy dictated

value in real-time. This is specially effective for the ubiquitous class of linear aggregation utility functions for which the inverse problem can be solved efficiently as a constrained linear program. In the experiments reported in Section 7, we used this heuristic (Called GPA hereafter) even though the outcome space sizes were small to evaluate the quality of this approximation.

6.3.2 Learning The Acceptance Model

In the real-world, acceptance models can either be known from domain knowledge (e.g. the probability of buying at a very high price is almost zero) or learned from past experience. In this paper, we employ a simplistic method for learning the acceptance model to ensure that the performance reported is not just a side effect of an exceptionally good acceptance model.

During training, the agent acts according to an exploration policy collecting all the negotiation logs. When an opponent accepts or proposes an offer, a counter for this outcome in the current negotiation round is incremented (current step for real-time scenarios). We then interpolate missing data linearly.

Four base exploration policies were used in this work:

Random offers randomly and never accepts. This method has the advantage of that all outcomes can in principles be sampled at any time-step. Nevertheless, it has the disadvantage of short negotiation times because the explorer agent may offer outcomes with very high utility to the opponent at any step leading to acceptance by the opponent and ending of the negotiation. The following two strategies avoid this pitfall at the expense of sampling only from a small set of outcomes at every time-step.

Aspiration offers according to a boulware time-based strategy [21] and never accepts.

Naive Tit for Tat offers according to the naive tit-for-tat policy [21] and never accepts.

ANAC Top Agents In this case, we use one of the winner agents of the self-utility track of ANAC since 2010, namely, AgentK (2010) [22], HardHeaded (2011) [23], CUHKAgent (2012) [24], AgentM (2013) [25] , Atlas3 (2015) [26] and Caduceus (2016) [10]. This exploration policy was only used to evaluate the sensitivity of the proposed method to different exploration policies.

During training, the agent samples one of these three strategies randomly per negotiation and uses it consistently until the negotiation ends updating the acceptance model based on the opponent's behavior.

6.3.3 Handling Real-Time Deadlines

Up until now, we modeled the policy as a sequence mapping from round number to either an outcome or a utility value. This is an appropriate choice when the time-limit of the negotiation is defined in terms of round numbers. When the time-limit is defined in real-time (i.e. in seconds), the agent cannot know in advance the length of the policy to use.

A simple extension to such circumstance is to discretize time into a predefined number of steps (T) keeping the policy a discrete mapping. We still need to know the number of such steps to use for solving the optimization problem which depends on

the number of steps that will actually be executed. To deal with this problem, while learning the acceptance model (Section 6.3.2) the agent can also learn a probability distribution over different time-lengths of the negotiation ($p(T)$). A policy can be generated for each length with a non-zero probability (for SAMs that requires no further calculations as these are already computed by QGCA). The agent can then use the policy corresponding to the length with the highest probability and switch dynamically to the more appropriate policy during the negotiation. This extension is analyzed in the last experiment reported in Section 7.

6.3.4 Multilateral Negotiations

This paper focused on bilateral negotiations using the alternating offers protocol (AOP). As inidcated in the introduction, extending the proposed method to multilateral negotiation using the Stacked Alternating Offers Protocol (SAOP) is straightforward. Here we sketch a general procedure for achieving this goal.

The most important aspect of SAOP for our purposes is that an offer is accepted *if and only if all negotiators — except the one offering it — accepted it*. This suggests that if we have an acceptance model for each of our partners a_m in a multilateral negotiation with M partners, we can construct a corresponding bilateral negotiation with a single *virtual* partner that has an acceptance model $\hat{a} \equiv \prod_{m=1}^{M} a_m$ which will be exactly equivalent to the original multilateral negotiation for the purpose of evaluated the expected utility of any policy.

Learning the individual acceptance models of each partner in each thread can be done using the same approach discussed in Section 6.3.2 because the same data about offers and acceptance decision is available for each partner.

7 Evaluation

The speed improvement of QGCA over GCA is algorithmic of order O(K). The improvements brought by PBS and PA are on the other hand heuristic. In this section we report three experiments to evaluate these improvements.

All evaluations reported in this paper were conducted using the open-source NegMAS [27] negotiation library. State of art agents were run on the GENIUS [14] platform and connected to NegMAS through its Genius-bridge.

7.1 QGCA vs GCA

QGCA and GCA find the same policy up to ties in expected utility. In the rare cases of having such ties, the two algorithms may generate different policies that turn out to provide different final utilities to the agent. In our first experiment, we directly compare QGCA and GCA on outcome spaces with increasingly larger sizes and policy lengths.

Fig. 2 shows the effect of outcome space size on the performance of QGCA and GCA. By construction both algorithms return the same policy (up to changes that does not affect expected utility) which shows in the right-hand figure. QGCA is *always* faster than GCA even for small outcome spaces of 4 outcomes (with a

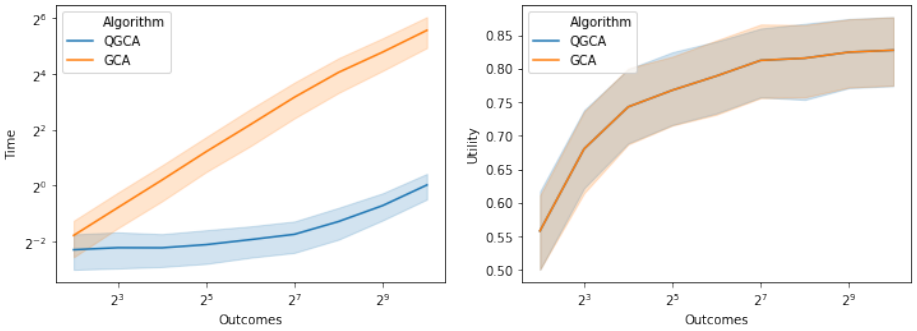


Fig. 2: Scalability of GCA and QGCA with outcome-space size

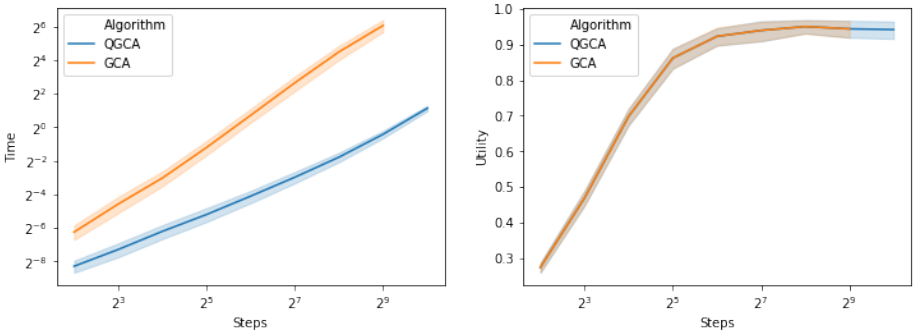


Fig. 3: Scalability of GCA and QGCA with policy length

speedup of 40%) and the speedup increases with outcome-space size reaching 4674% with for 1024 outcomes.

Fig. 3 shows the effect of policy length on the performance of QGCA and GCA. Again, there is no difference in the expected utility. QGCA is *always* faster than GCA even for short policies of 4 steps (with a speedup of 433%) and the speedup increases with policy length reaching 8916% with for 512 steps.

7.2 Probabilistic Opponents on Random Domains

The goal of this experiment was to evaluate the effectiveness of the PA algorithm and confirm the value added by each of its three stages (see Section 6) *when the assumptions of the OPL problem are satisfied*. To this end, we applied the proposed method to 100 domains with outcome space cardinality ranging between 10 and 100 with randomly selected utility functions for the evaluated agent each repeated 100 times to smooth out random effects. All negotiations were limited to a number of rounds equal to 50% of the number of outcomes. The opponent used a randomly generated TDAM acceptance model. Each configuration was repeated 100 times to smooth out random variations. The opponents did not offer and used a probabilistic acceptance strategy controlled by a randomly driven TDAM (a) with acceptance

probability limited under $2/T$ leading to a total acceptance probability of no more then 0.5 for any outcome during the negotiation.

The evaluated agent received an acceptance model which is calculated as:

$$A(\omega, i) = \tilde{a}(\omega, i) + e(\omega, i)$$

where e is sampled independently from a uniform distribution with a range of 20% the mean acceptance probability in a to simulate errors in learning the acceptance model.

The three stages of the proposed method (QGCA$^+$. PBS, PA) were evaluated against random offering, two baseline strategies (time-based boulware and naive tit-for-tat) and four state-of-the-art agents: PonPokoAgent (Winner of ANAC2017), Caudasaus (Winner of ANAC2016), ParsCat, and YXAgent (Second places in ANAC2016).

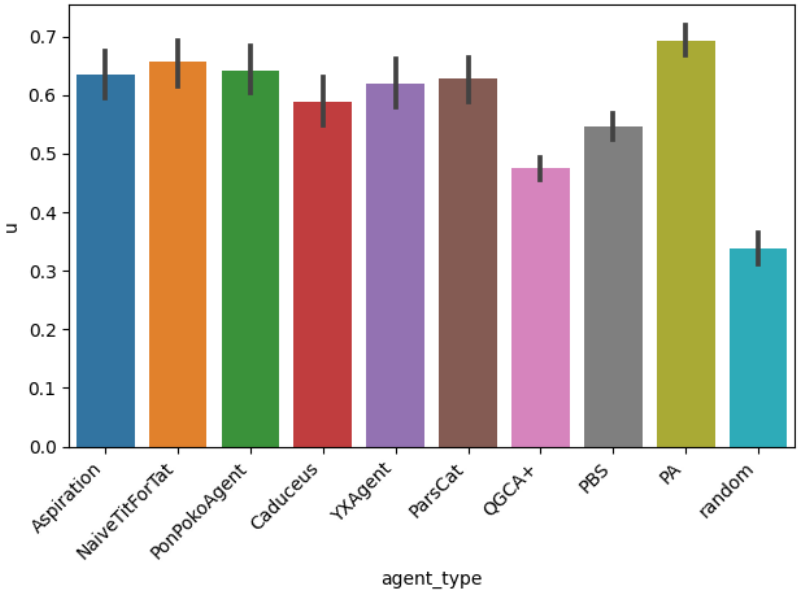


Fig. 4: Evaluation against probabilistic opponents.

Fig 4 shows the results of this analysis. As expected, PA outperforms PBS which outperforms QGCA$^+$ which supports the usefulness of every stage of the proposed algorithm. These differences were all statistically significant with a $p-$value less than 0.05 using a factorial t-test with Benferroni's multiple comparisons correction.

The differences that were *not* statistically significant were also suggestive. PA's performance was not different from the top state-of-the-art agents on statistical level even though it did not react in any way to the opponent's behavior. This is not as surprising as it sounds as the opponent did behave randomly in this experiment. Another finding from this experiment is that simple baseline agents behaved at the same level as state-of-the-art agents.

7.3 Realistic Domains and Opponents

For the third experiment, we evaluate the proposed method using all negotiation domains with a cardinality between 10 and 100 in the GENIUS [14] package which contains all the domains used for the International Automated Negotiation Agents Competition (ANAC) since 2010 [28]. For multilateral scenarios of N agents, $N-1$ different bilateral scenarios were constructed by selecting consecutive pairs of utility functions (leading to 25 different domains).

The opponents selected were state-of-the-art negotiation strategies. Namely: YXAgent, Caduceus, ParsCat (Top 3 places of ANAC2016), PonPokoAgent (Winner of ANAC2017).

The experiment was conducted in two phases. In the training phase, the agent acted randomly to learn the acceptance models using the simple approach highlighted in Section 6.3.2. Different TDAM and SAM were learned for each opponent at every domain. Fifty training sessions were conducted for every domain and opponent.

In the testing phase, the proposed approach (PA), plus two baseline strategies (time-based boulware and naive tit-for-tat) and four state of the art agents PonPokoAgent (2017), Caduceus (2016), ParsCat (2016), YXAgent (2016) negotiated against each of the aforementioned opponents in each domain 10 times. The number of rounds was limited to 10 which amounts to the agents being able to explore between 25% to 100% of the outcome space.

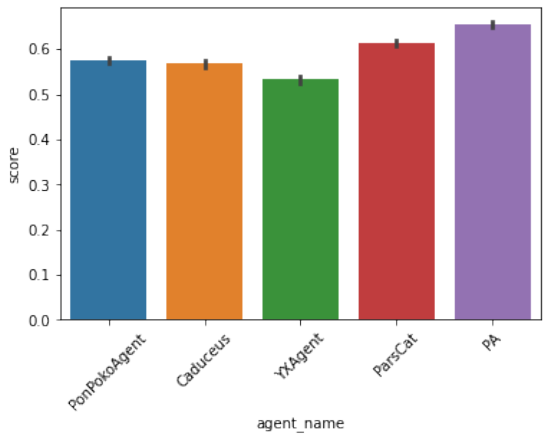


Fig. 5: Results of Evaluation in GENIUS domains against state of the art (SOTA) opponents on GENIUS all domains of sizes between 10 and 100 outcomes (10 rounds) with PA trained with the same agent it is tested against.

Fig 5 shows the results of this experiment. Even though the proposed method (PA) is run completely offline and the agent does not change its behavior during the negotiation in any way, its score is on bar with the most advanced negotiation heuristics evaluated. Applying factorial t-test with multiple-comparisons correction, we found that the differences between the proposed agent and Caudaseaus and ParsCat

are not statistically significant while the differences between it and PonPokoAgent and YXAgent are at a significance level less than 0.01.

The results of this experiment should be taken with a grain of salt for three reasons: Firstly, PA had access to the acceptance model which was not available to other agents. Secondly, the short negotiation time (10 rounds), made it harder for other agents to adapt their strategies to the behavior of the opponent. Finally, all agents except PA had access to the structure of the issue space (how many issues) and knew that the utility function was a linear combination of value functions. PA is incapable of exploiting this information in its current form.

The results are, nevertheless, instructive as they show that effective use of an empirically learned acceptance models can lead to improvements in the expected utility which is the main message of this paper. In the following experiments we evaluate the proposed approach in more realistic situations.

7.4 Generalization Against State-of-the-art Agents

For the fourth experiment, we evaluate the generalization ability of the proposed method to new unseen opponents. The experiment was conducted using all negotiation domains with a cardinality not greater than 100 in the GENIUS [14] package which contains all the domains used for the International Automated Negotiation Agents Competition (ANAC) since 2010 [28]. For multilateral scenarios of N agents, $N-1$ different bilateral scenarios were constructed by selecting consecutive pairs of utility functions (leading to 70 different domains).

The state-of-the-art (SOTA) competitors selected were all winners of the ANAC competition between 2010 and 2017[2]. Namely: AgentK (2010), HardHeaded (2011), CUHKAgent (2012), TheFawkes (2014), AgentM (2014) [29], Atlas3 (2015), Caduceus (2016), PonPokoAgent (2017) as well as NiceTitForTat which represents a behavior-based algorithm with opponent modeling[21]. The opponents consisted of all SOTA competitors plus two baseline strategies (time based Boulware and Naive Tit for Tat) as well as RandomDance (2015) representing a probabilistic SOTA agent.

The experiment was conducted in two phases. In the training phase, the agent acted randomly to learn the acceptance models using the simple approach highlighted in Section 6.3.2. Different TDAM and SAM were learned for each opponent at every domain. Moreover, a single agent-utility based TDAM and SAM (Section 6.3.1) was learned for every opponent by combining the logs of all negotiations against that opponent during training. 36 training sessions were conducted against each opponent in each scenario.

In the testing phase, each of the state of the art agents (SOTA) as well as PA and its domain-independent extension (GPA) were tested. For the two proposed variations, the acceptance-model used against each opponent was the average of

[2] The ANAC competition after 2018 used uncertainty in the agent's own utility function which gives the proposed method an unfair advantage against them. Moreover, negotiation under uncertainty is not compatible with the NegMAS-Genius bridge used to run the experiments. The original plan was to use AgreeableAgent as well (winner of ANAC 2018) but it was raising exceptions most of the time specially against SOTA agents leading to a bias toward the proposed method. To avoid this bias, data from all sessions with AgreeableAgent were removed.

models learned during training against all *other* opponents. This ensured that the agent was using a policy that was learned against opponents different from the one it is encountering during testing. Each of the 70 domains was run six times per combination of agents (three times per ufun assignment) to reduce the effect of randomization in agent behavior.

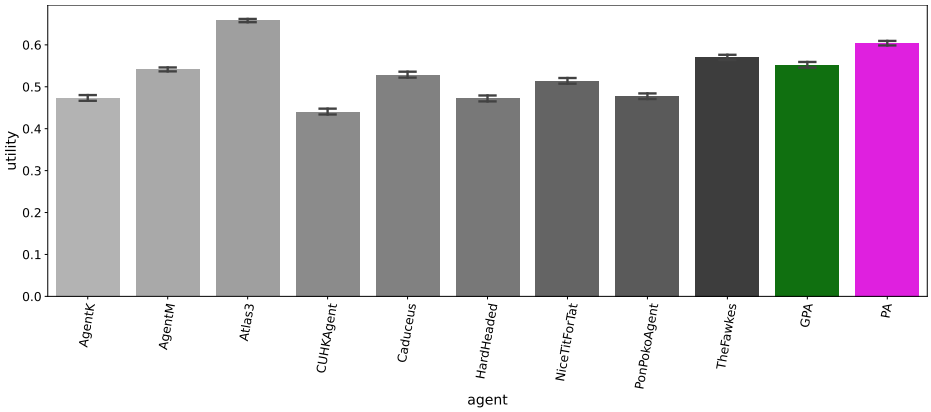


Fig. 6: Evaluation of Generalization against top ANAC agents. ANAC winner agents (SOTA) are shown in grayscale. All 10 to 100 outcomes GENIUS domains were used and the number of rounds was fixed at 10. Error bars represent the standard error.

Table 3: The proposed method against state-of-the-art negotiators on Genius domains.

agent	mean	std	min	25%	50%	75%	max
Atlas3	0.658	0.292	0.000	0.452	0.689	0.933	1.000
PA	0.604	0.386	0.000	0.277	0.701	1.000	1.000
TheFawkes	0.570	0.457	0.000	0.000	0.840	1.000	1.000
GPA	0.552	0.472	0.000	0.000	0.890	1.000	1.000
AgentM	0.541	0.329	0.000	0.299	0.586	0.836	1.000
Caduceus	0.529	0.486	0.000	0.000	0.834	1.000	1.000
NiceTitForTat	0.514	0.477	0.000	0.000	0.676	1.000	1.000
PonPokoAgent	0.478	0.464	0.000	0.000	0.714	0.942	1.000
AgentK	0.473	0.486	0.000	0.000	0.000	1.000	1.000
HardHeaded	0.473	0.497	0.000	0.000	0.000	1.000	1.000
CUHKAgent	0.441	0.497	0.000	0.000	0.000	1.000	1.000

Fig. 6 and Table 3 show a summary of the results. We applied factorial t-test comparing the proposed two algorithms PA, GPA with state-of-the-art algorithms which showed that all differences beteween the propsoed methods and SOTA agents in this table were statistically significant. Table 4 shows the performance of PA and GPA compared with the *average* performance of SOTA agents. PA achieves the highest mean utility of 0.604 which is 9.4% higher than SOTA performance while

Table 4: The proposed method against state-of-the-art negotiators on Genius domains (combined).

Agent Type	mean	std	min	25%	50%	75%	max
PA	0.604	0.386	0.000	0.277	0.701	1.000	1.000
GPA	0.552	0.472	0.000	0.000	0.890	1.000	1.000
SOTA	0.520	0.453	0.000	0.000	0.661	1.000	1.000

GPAreceives the second mean utility of 0.52 which is still 6% higher than SOTA agents. This shows the price of domain independence.

Atlas3 wins the competition with a mean score of 0.658 (std. dev. 0.292). The proposed PA algorithm achieved second place with a 8% lower mean score of 0.605 ($t = -15.055$, $p < 1e-10$). Moreover, PA outperformed the third contender (TheFawkes) by 6% ($t = 20.219$, $p < 1e-10$), GPA by 9.4%, AgentM by 11.6%, Caduceus by 14.2%, NiceTitForTat by 17.5%, PonPokoAgent by 26.4%, AgentK by 27.7%, HadrdHeaded by 27.7%, and CUHKAgent by 37%.

The proposed domain-independent extension introduced in Section 6.3.1 (GPA) received fourth place with a mean score of 0.552 after being outperformed by Atlas3 ($t = -30.613$, $p < 1e-10$), PA ($t = -24.811$, $p < 1e-10$) and TheFawkes ($t = -5.841$, $p = 1e-6$). GPA outperformed AgentM by 2%, Caduceus by 4.3%, NiceTitForTat by 7.4%, PonPokoAgent by 15.5%, AgentK by 16.7%, HadrdHeaded by 16.7%, and CUHKAgent by 25.2%.

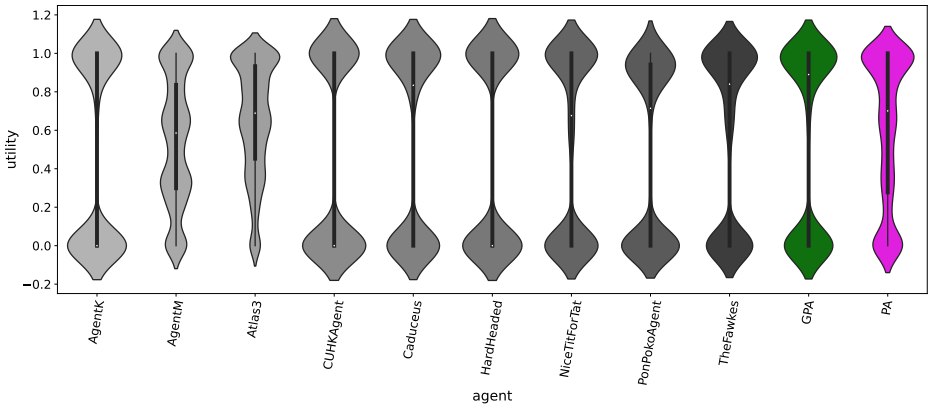


Fig. 7: Violin Plot of the results shown in Fig. 6.

To understand the reason for this difference in behavior between different algorithms, Fig. 7 shows a violin plot representing the distribution of different scores for different algorithms. It is apparently clear that AgentK, HardHeaded, CUHK, PonPokoAgent, and Caudasaus receive either very high or very low scores while the agents with best results (Atlas3, PA, AgentM and to a lower degree GPA, NiceTitForTat, and TheFawkes) have a more balanced distributions. The reason Agent3 won this competition is apparently that it could get higher fraction of very high (near

one) scores as evident from the shape of the distribution. Moreover, Compared PA was able to achieve higher fraction of mid-range scores compared with GPA because it was optimized for each domain independently.

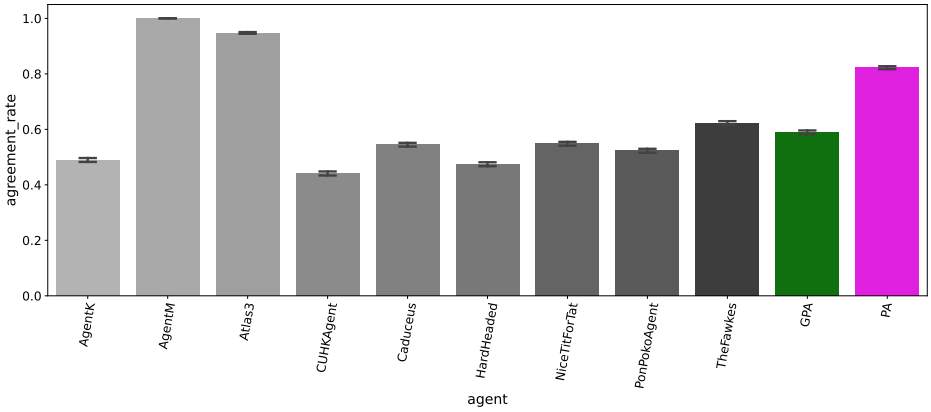


Fig. 8: Agreement rates against top ANAC agents. ANAC winner agents (SOTA) are shown in grayscale. All 10 to 100 outcomes GENIUS domains were used and the number of rounds was fixed at 10. Error bars represent the standard error.

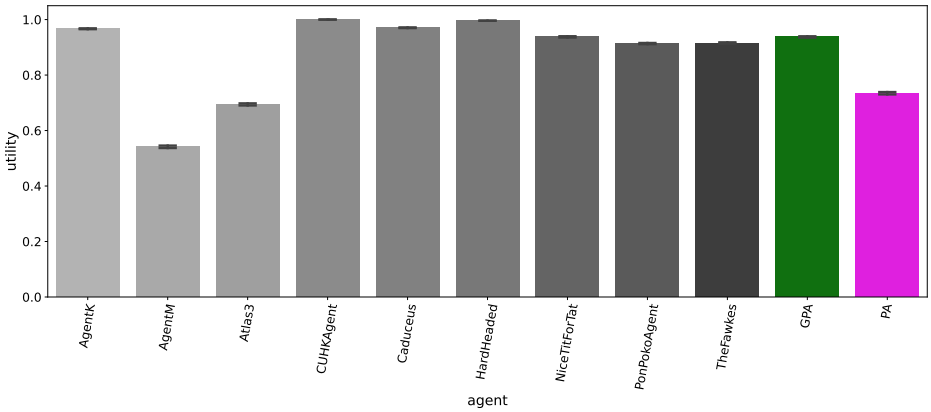


Fig. 9: Utility in case of agreement against top ANAC agents. ANAC winner agents (SOTA) are shown in grayscale. All 10 to 100 outcomes GENIUS domains were used and the number of rounds was fixed at 10. Error bars represent the standard error.

Fig 8 shows the agreement rates achieved by different algorithms and Fig. 9 shows the utility received by each algorithm when an agreement is reached. PA achieves higher agreement rates than SOTA agents (except for Atlas3 and AgentM) but on the expense of receiving a modest average utility in cases of agreement. GPA has almost the opposite behavior with a much lower agreement rate and higher utility in cases

of agreement. This shows the importance of achieving a balance between concession (propensity to reach agreements) and toughness (bias toward high utilities in cases of agreements).

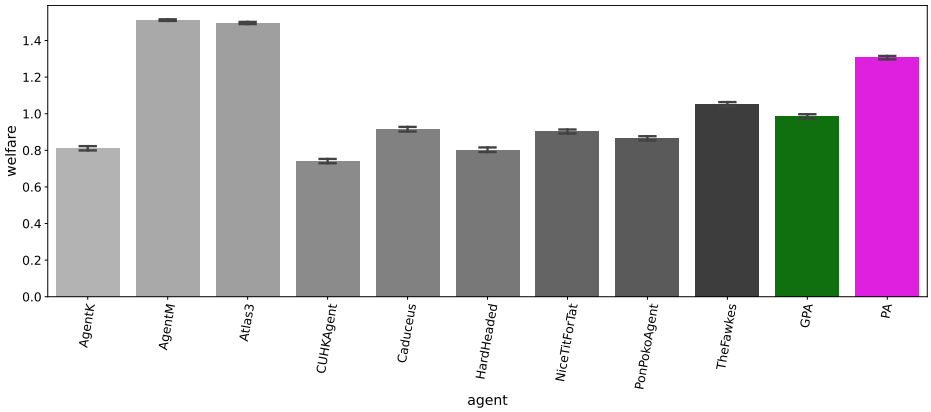


Fig. 10: Total welfare against top ANAC agents. ANAC winner agents (SOTA) are shown in grayscale. All 10 to 100 outcomes GENIUS domains were used and the number of rounds was fixed at 10. Error bars represent the standard error.

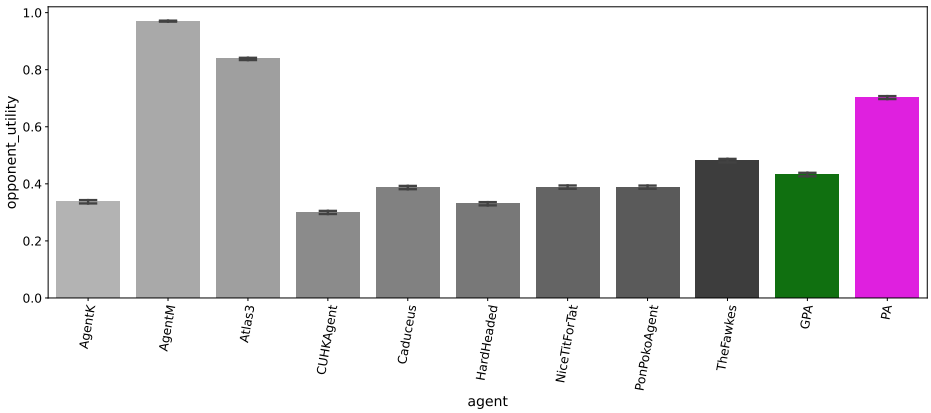


Fig. 11: Opponent utility against top ANAC agents. ANAC winner agents (SOTA) are shown in grayscale. All 10 to 100 outcomes GENIUS domains were used and the number of rounds was fixed at 10. Error bars represent the standard error.

Fig 10 shows the welfare (defined as sum of utilities for both partners) achieved by different algorithms and Fig. 11 shows the utility achieved by the partner (opponent) in each case. In business environment where negotiations are repeated, allowing the partner to achieve higher utility is an advantage to enhance the long-term rela-

tionship with the partner. PA outperfomrs TheFawkes in both measures while still being outperformed by Atlas3.

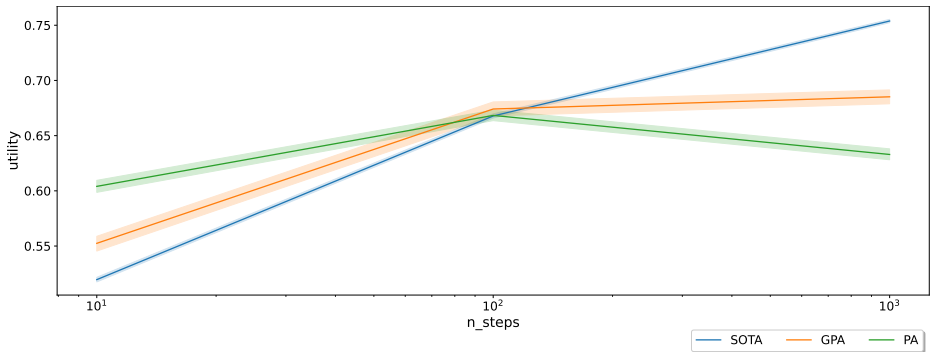


Fig. 12: Effect of the number of negotiation rounds (steps) for different algorithms.

Fig 12 shows the effect of the number of negotiation rounds allowed on different algorithms with all SOTA algorithms combined together to reduce clutter. The proposed method clearly outperforms SOTA algorithms at short negotiations ($T = 10$), achieves indistinguishable performnace from them for medium negotiatiosn ($T = 100$) and is clearly outperformed by SOTA agents for long negotiations ($T = 1000$). This highlights one of the main limitaions of the proposed method (being a completely offline method) as longer negotiatiosn provide the agent with an opportunity to improve the acceptance model (during the negotiation) which is utilized effectively by SOTA agents but not by the pure form of the proposed approach. Extending the proposed method to an online version that updates the acceptance model at every round could alleviate this problem but it not pursued in this work.

Fig 13 shows the performance of all algorithms for each domain considered. It is clear that the advantage of PA over GPA is consistent for all domains (same is true for the advantage of Atlas3 over other algorithms). As expected, GPA almost never outperformed PA for any domains.

Fig. 14 shows the performance of the proposed method as a function of the number of outcomes for different negotiation lengths. Again, the performance of the proposed approach seems best at short negotiations. Moreover, it is clear that on average (Fig. 14d), the proposed method achieves its best performance compared with SOTA agents with small outcome-spaces and its worst performance with mid-sized outcome spaces.

7.5 Real-time limit

In the final experiment, we relax the assumption that the number of rounds is known in advance used until now using the approach described in Section 6.3.3. The dataset and procedure are exactly the same as the previous experiment with the only exception of using real-time limit of 40 seconds instead of a predefined number of rounds during testing. The proposed PA and GPA algorithms were trained with 100 rounds

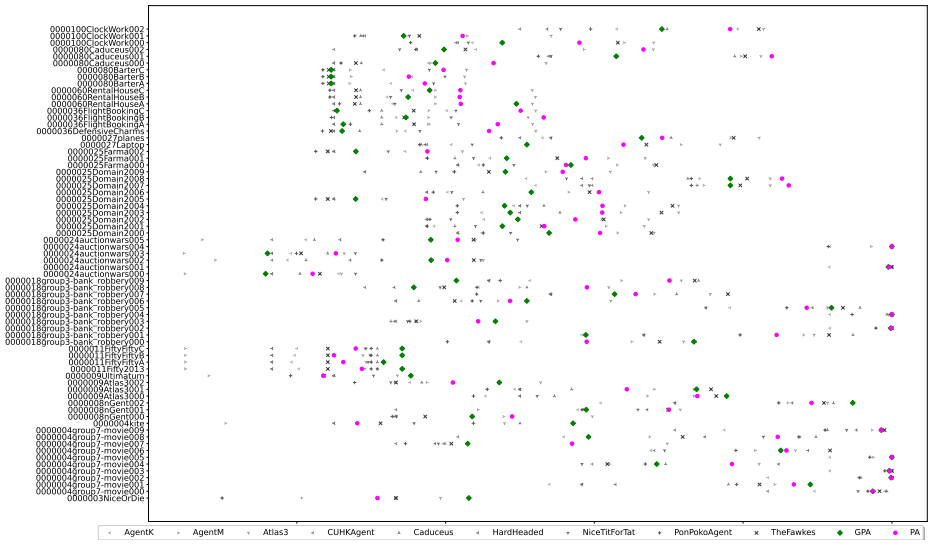
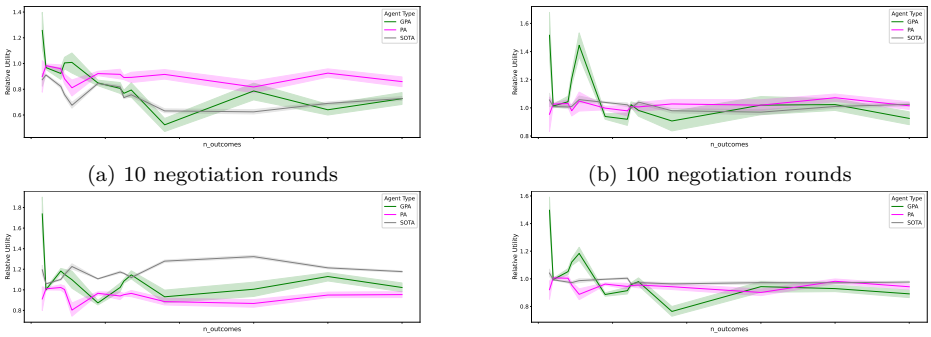


Fig. 13: Performance of all algorithms in each individual domain.



(a) 10 negotiation rounds

(b) 100 negotiation rounds

(c) 1000 negotiation round

(d) Average

Fig. 14: Comparing outcome-space scalability of the proposed method to State-of-the-art (SOTA) agents

negotiations and tested with the same real-time limits mentioned above. Note that this forces the proposed method to extrapolate over negotiation length as well as opponent type (and domain in case of GPA).

Fig. 15 and Table 5 show a summary of the results. We applied factorial t-test comparing the proposed two algorithms PA, GPA with state-of-the-art algorithms which showed that all differences beteween the propsoed methods and SOTA agents in this table were statistically significant except for the ones shown in Table 6. This time, PA outperforms all other agents in terms of mean utility achieving statistically *insignificant* improvements over HardHeaded (0.1%), AgentK (1.2%) Caduceus (1.8%) and NiceTitForTat (2.6%). PAachieved statistically significant improvements over TheFawkes (5.6%), GPA (7.8%), Atlas3 (9.1%), CUHKAgent (11.3%), PonPokoAgent (11.6%), and AgentM (34.6%).

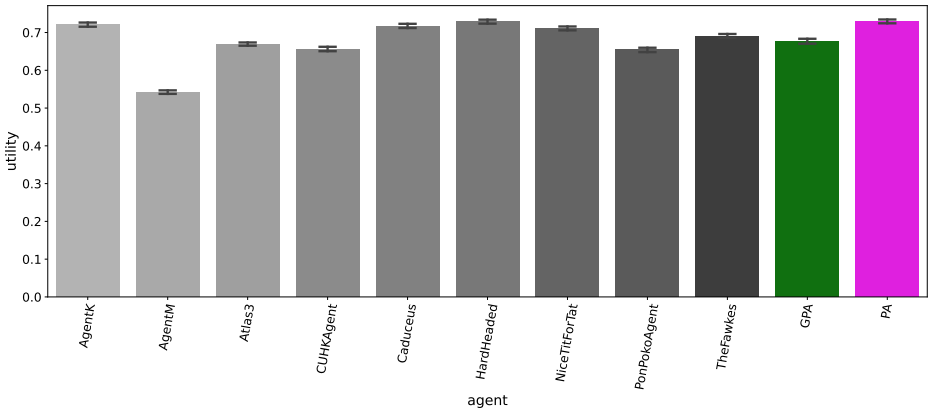


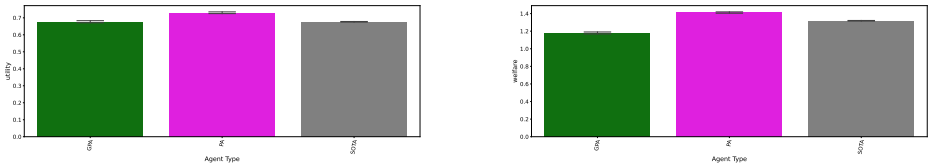
Fig. 15: Evaluation of Generalization against top ANAC agents with real-time limit of 40 seconds. ANAC winner agents are shown in grayscale. Error bars represent the standard error.

Table 5: The proposed method against state-of-the-art negotiators on Genius domains (real-time timeout condition with timeout 40 seconds).

agent	mean	std	min	25%	50%	75%	max
PA	0.730	0.353	0.000	0.537	0.914	1.000	1.000
HardHeaded	0.729	0.387	0.000	0.687	0.925	1.000	1.000
AgentK	0.721	0.391	0.000	0.652	0.919	1.000	1.000
Caduceus	0.717	0.387	0.000	0.702	0.900	1.000	1.000
NiceTitForTat	0.711	0.368	0.000	0.527	0.900	1.000	1.000
TheFawkes	0.691	0.388	0.000	0.537	0.896	1.000	1.000
GPA	0.677	0.433	0.000	0.000	0.943	1.000	1.000
Atlas3	0.669	0.300	0.000	0.422	0.703	0.953	1.000
CUHKAgent	0.656	0.433	0.000	0.000	0.987	1.000	1.000
PonPokoAgent	0.654	0.412	0.000	0.000	0.900	0.947	1.000
AgentM	0.542	0.331	0.000	0.299	0.595	0.840	1.000

Table 6: Differences between agents that were found *not* to be statistically significant (after Bonferroni Correction) in the real-time evaluation experiment.

a	b	t	p
PA	HardHeaded	0.009	0.993
PA	AgentK	1.376	0.169
PA	Caduceus	1.919	0.055
GPA	PonPokoAgent	1.639	0.101
GPA	CUHKAgent	1.034	0.301
GPA	Atlas3	0.417	0.676



(a) Utility (b) Welfare

Fig. 16: Comparing the proposed method to State-of-the-art (SOTA) agents

Table 4 shows the performance of PA and GPA compared with the *average* performance of SOTA agents. PA achieves the highest mean utility of 0.730 which is 7.8% higher than SOTA performance while GPA ties for the second mean utility of 0.677 with SOTA agents. Fig. 16 shows the performance of PA and GPA compared with SOTA agents in terms of utility received and total-welfare. PA outperforms SOTA in both while GPA ties with them for utility and is outperformed by SATA in terms of total welfare. Again, showing the price of domain independence.

Table 7: The proposed method against state-of-the-art negotiators on Genius domains (real-time timeout condition with timeout 40 seconds).

Agent Type	mean	std	min	25%	50%	75%	max
PA	0.730	0.353	0.000	0.537	0.914	1.000	1.000
GPA	0.677	0.433	0.000	0.000	0.943	1.000	1.000
SOTA	0.677	0.383	0.000	0.388	0.857	1.000	1.000

8 Limitations and Discussion

The main goal of this work is *not* to provide a new negotiation heuristic but to extend existing foundational research on time-based strategies in negotiation. One important result of this research is that concession – usually assumed to be the default behavior during negotiation is actually optimal *only* under the assumption of a static acceptance model. This is also clear from the fact that PA can improve upon the *optimal* concession strategy found by QGCA$^+$. It is easy to construct counter examples for both the concession lemma [9] and the optimality of greedy policy construction for general time-dependent acceptance models (See Appendix B for a proof by contradiction for monotonic acceptance models and hence TDAMs).

The proposed QGCA, PA, and GPA share the feature that they calculate the policy *before* even starting the negotiation. They do not adapt their behavior to the behavior of the agent during the negotiation. It is quiet easy to extend this approach *greedily* by re-evaluating the acceptance model and hence the optimal policy at every time-step based on the behavior of the agent. For this to be achievable, a method for adapting the initial acceptance model based on the offers exchanged during the negotiation is needed. In the future, we will consider this question.

Nevertheless, and despite this offline nature of the proposed methods, the experiments reported in Section 7 show that it can still behave on bar and sometimes outperform state-of-the-art automated negotiation agents even when they are trained and tested against different strategies. Moreover, the good performance of PA shown in the last experiment suggests that the proposed method generalizes well to negotiations much longer than the ones it was trained with. Generalization to different domains achieved by GPA is also shown to have a cost in terms of the mean utility attained in all conditions stressing the importance of taking the details of the negotiation scenario into account.

Even though we discussed a hierarchy of acceptance models in Section 3.4, existing methods including ours assumes either a SAM (GCA, QGCA) or the most general TDAM (PBS, PA). Realistic acceptance models are expected to fall in the range between these two extremes and designing optimal policy learners for each of these intermediate acceptance model types that can be more efficient than the proposed TDAM approach is another direction of future research. It is instructive to note that the counter examples we could find did not have any *homogenous* acceptance models, nor did it contain any *monotonically decreasing* acceptance models. It may still be possible to use QGCA+ directly in these special cases but no proof of optimality could be devised for either case. The proosed method — even with its online extension — is designed for TDAMs. Extension to more general acceptance models is a direction of future research.

The proposed method is $O(K^2T)$ which is polynomial in both the number of rounds and outcomes. Nevertheless, for multi-issue negotiations, the number of outcomes grows *exponentially* with the number of issues. This limits the applicability of the proposed method to outcome-spaces with a small number of issues. One way to alleviate this limitation is to model both the utility function and acceptance models as functions of issue values instead of complete outcomes and extend the process of finding the optimal policy to take into account this structure. This is another direction we plan to pursue in the near future.

Another obvious limitation of the proposed method is that it does not take into account the opponent offers. The three proposed algorithms generate the expected utility of following the *optimal* policy found while ignore the opponent. As we discussed earlier, it is possible to use that to accept offers from the opponent that have utilities above this threshold. This *opportunistic* acceptance policy is, nevertheless, just a heuristic. Taking opponent offers seriously, would require a modification of the way the expected utility is calculated and is another direction of future research.

Finally, the heuristic extension of QGCA+ to TDAMs using PBS and PA proposed in this work is but one possible approach. In the future we will pursue this direction by comparing different optimization approach starting from the initial greedy concession policy found by QGCA+.

9 Conclusions

Finding an optimal time-based policy against an opponent given a probabilistic model of its acceptance strategy is an important direction of research in automated negotiation. Until now, an optimal solution called the Greedy Concession Algorithm (GCA) was only known for the case of a static acceptance model and with a non-

repeating policy. GCA is linear in the policy length but quadratic in the outcome space size.

This work introduces QGCA (Quick GCA) which extends the applicability of GCA to repeating and probabilistic policies while speeding up the process of finding the optimal policy to be linear in the size of both policy length and outcome space size. This is achieved without changing the space complexity of the algorithm.

A three-steps extension to general time-dependent acceptance models (QGCA$^+$, PBS, and PA) is also introduced that progressively modifies an approximate policy found using QGCA applied to an approximate static version of the acceptance model.

The proposed method was evaluated against opponents that use the assumed acceptance model type and against state-of-the-art agents after learning an approximation of their behavior as a time-dependent acceptance model based on negotiation records. The results of this evaluation show that the proposed method is capable of providing performance comparable to state-of-the-art negotiation heuristics even though it does not react to the opponent's behavior during the negotiation.

The main limitations of the proposed approach are being an offline method (not reacting to the opponent's behavior) and not utilizing the internal structure of the outcome space which is itself exponential in the number of negotiation issues which limits its applicability to small to medium sized negotiations. Both of these limitations may be can be alleviated by extending the proposed method to dynamic acceptance models that utilize the structure of the issue space. This is our main direction of future research.

A Proofs for Operations on Policies

Here we provide a simple proof for the results found in Table 1 and Table 2. In the following proofs, there are several cases in which a division by $1 - a_k^k$ is needed. This seems to pose a difficulty when $A_k^k = 1$ but in all of these cases, it is possible to rewrite the expression avoiding this division using P and S values albeit with a much longer expression and cumbersome notation. We will avoid this complexity here and keep the version using the devision.

A.1 Proof for π_i^ω

$$\mathcal{EU}(\pi) = \sum_{j=1}^{T+1} u_j a_j^j P_j = \sum_{j=1}^{i-1} u_j a_j^j P_j + u_i a_i^i P_i + \sum_{j=i+1}^{T+1} u_j a_j^j P_j$$

Notice that, by definition, $a(\omega, T+1) = 1$ and $\pi_{T+1} = \phi$ (See Section 3.5) which takes into account nonzero reservation values.

$$\begin{aligned}\mathcal{EU}(\pi_i^\omega) = &\sum_{j=1}^{i-1} u_j a_j^j P_j + u(\omega) a(\omega, i) P_i \\ &+ \sum_{j=i+1}^{T+1} u_j a_j^j P_i \left(1 - a(\omega, i)\right) \prod_{l=i+1}^{j-1} 1 - a_l^l\end{aligned}$$

Subtracting and after some manipulations we get:

$$\begin{aligned}\mathcal{EU}(\pi_i^{\omega}) - \mathcal{EU}(\pi) &= P_i\left(u(\omega)a(\omega,i) - u_i a_i^i\right) - (S_{T+1} - S_i) + (S_{T+1} - S_i)\frac{1-a(\omega,i)}{1-a_i^i} \\ &= P_i\left(u(\omega)a(\omega,i) - u(\pi_i)a_i^i\right) + (S_{T+1} - S_i)\frac{a_i^i - a(\omega,i)}{1-a_i^i}\end{aligned}$$

This completes the proof for TDAM (Table 1). The proof for SAM (Table 2) follows from replacing $a(x,i)$ with $a(x)$. □

A.2 Proof for $\pi_{k\leftrightarrow s}$

$$\begin{aligned}\mathcal{EU}(\pi_{k\leftrightarrow s}) &= \sum_{i=1}^{k-1} u_i a_i^i P_i + u_s a_s^k P_k + \sum_{i=k+1}^{s-1} u_i a_i^i P_i \frac{1-a_s^k}{1-a_k^k} + u_k a_k^s P_s \frac{1-a_s^k}{1-a_k^k} + \sum_{i=k+1}^{T+1} u_i a_i^i P_i \\ &\quad - \sum_{i=1}^{T+1} u_i a_i^i P_i \\ &= P_k\left(u_s a_s^k - u_k a_k^k\right) + P_s\left(u_k a_k^s \frac{1-a_s^k}{1-a_k^k} - u_s a_s^s\right) + \left(\sum_{i=k+a}^{s-a} u_i a_i^i P_i\right)\frac{1-a_s^k}{1-a_k^k} \\ &= P_k\left(u_s a_s^k - u_k a_k^k\right) + P_s\left(u_k a_k^s \frac{1-a_s^k}{1-a_k^k} - u_s a_s^s\right) + (S_{s-1} - S_k)\frac{1-a_s^k}{a-a_k^k} \\ &= u_k\left(a_k^s P_s \frac{1-a_s^k}{a-a_k^k} - a_k^k P_k\right) + u_k\left(a_s^k P_k - a_s^s P_s\right) + \frac{a_k^k - a_s^k}{1-a_k^k}(S_{s-1} - S_{k+1})\end{aligned}$$

This completes the proof for TDAM (Table 1). The proof for SAM (Table 2) follows from replacing a_x^i with a_x. □

A.3 Proof for $\pi_{k\to 1}$

This is a special case of $\pi_{k\leftrightarrow s}$ with $s = k+1$.

$$\begin{aligned}\mathcal{EU}(\pi_{k\leftrightarrow s}) &= u_k\left(a_k^s P_s \frac{1-a_s^k}{a-a_k^k} - a_k^k P_k\right) + u_k\left(a_s^k P_k - a_s^s P_s\right) + \frac{a_k^k - a_s^k}{1-a_k^k}(S_{s-1} - S_{k+1}) \\ &= u_k\left(a_k^{k+1} P_{k+1} \frac{1-a_{k+1}^k}{a-a_k^k} - a_k^k P_k\right) + u_k\left(a_{k+1}^k P_k - a_{k+1}^{k+1} P_{k+1}\right) \\ &= u_k\left(a_k^{k+1} P_k\left(1-a_{k+1}^k\right) - a_k^k P_k\right) + u_k\left(a_{k+1}^k P_k - a_{k+1}^{k+1} P_{k+1}\right)\end{aligned}$$

This completes the proof for TDAM (Table 1). The proof for SAM (Table 2) follows from replacing a_x^i with a_x. □

Note that for SAM, this becomes:

$$\begin{aligned}\mathcal{EU}(\pi_{k\leftrightarrow s}) &= u_k\left(a_k^{k+1} P_k\left(1-a_{k+1}^k\right) - a_k^k P_k\right) + u_k\left(a_{k+1}^k P_k - a_{k+1}^{k+1} P_{k+1}\right) \\ &= u_k\left(a_k P_k\left(1-a_{k+1}\right) - a_k P_k\right) + u_k\left(a_{k+1} P_k - a_{k+1} P_{k+1}\right) \\ &= u_k\left(-a_k a_{k+1} P_k\right) + u_{k+1}\left(a_k a_{k+1} P_k\right) \\ &= P_k a_k a_{k+1}\left(u_{k+1} - u_k\right)\end{aligned}$$

This expression is always positive if $u_{k+1} > u_k$, which leads to the greedy concession lemma proven by Baarslag et al. [9].

A.4 Proof for $\pi_{\omega@i}$

$$\begin{aligned}
\mathcal{EU}(\pi_{\omega@i}) &= \sum_{i=1}^{k-1} u_i a_i^i P_i + a(\omega,k)u(\omega)P_k + \sum_{i=k}^{T+1} u_i a_i^{i+1} P_i \left(1 - a(\omega^k)\right) - \sum_{i}^{T+1} u_i a_i^i P_i \\
&= a(\omega,k)u(\omega)P_k + (1 - a(\omega,k)) \sum_{i=k}^{T+1} u_i a_i^{i+1} P_k \prod_{j=k+1}^{i-1} \left(1 - a_j^{j+1}\right) - \sum_{i=k}^{T} a_i^i u_i P_k \prod_{j=k+1}^{i-1} \left(1 - a_j^j\right) \\
&= P_k \left(u_k a(\omega,k) + (1 - a(\omega,k)) \sum_{i=k}^{T+1} u_i \left(a_i^{i+1} - a_i^i\right) \left(\prod_{j=k+1}^{i-1} \left(1 - a_j^{j+1}\right) - \prod_{j=k+1}^{i-1} \left(1 - a_j^j\right) \right) \right) \\
&= u(\omega)a(\omega,k)P_k + (1 - a(\omega,k)) \left(S_{T+1}^{+1} - S_{k-1}^{+1}\right) - (S_{T+1} - S_{k-1})
\end{aligned}$$

where we define $S_j^{+1} = \sum_{i=1}^{j} u_i a_i^{i+1} P_i^{+1}$, and $P_i^{+1} = \prod_{j=1}^{i-1} 1 - a_j^{j+1}$. These two new lists can be calculated incrementally the same way that S and P are calculated (See Equations 3 and 4).

For the special case of a SAM, $S_j^{+1} = S_j$ and $P_i^{+1} = P_i$ which simplifies the expression greatly:

$$\begin{aligned}
\mathcal{EU}(\pi_{\omega@k}) &= u(\omega)a(\omega,k)P_k + (1 - a(\omega,k)) \left(S_{T+1}^{+1} - S_{k-1}^{+1}\right) - (S_{T+1} - S_{k-1}) \\
&= u(\omega)a(\omega)P_k - a(\omega)\left(S_{T+1} - S_{k-1}\right) \\
&= a(\omega)\left(u(\omega)P_k - (S_{T+1} - S_{k-1})\right) \\
&= a(\omega)\left(S_{k-1} + u(\omega)P_k - S_{T+1}\right)
\end{aligned}$$

□

B Counter example for the Greedy Concession Lemma

To prove that the Greedy Concession Lemma is false for a TDAM, we only need to find a single counter example. This is one such counter example:

Let $T = 2$, $\Omega = \{\omega_i | 0 \leq i \leq 4\}$, and let a, u be as follows:

$$a = \begin{pmatrix} t & \omega_0 & \omega_1 & \omega_2 & \omega_3 & \omega_4 \\ 0 & 0.0729 & 0.3047 & 0.2323 & 0.9274 & 0.7293 \\ 1 & 0.2280 & 0.9936 & 0.9323 & 0.2979 & 0.1223 \end{pmatrix}$$

$$u(\omega_i) = \begin{pmatrix} \omega_0 & \omega_1 & \omega_2 & \omega_3 & \omega_4 \\ 0.0723 & 0.1778 & 0.8086 & 0.8768 & 0.9446 \end{pmatrix}$$

By substituting into Equation 2, It is easy to verify that the optimal policies of length one and two are:

$$\pi_1^* = \langle \omega_3 \rangle$$

$$\pi_2^* = \langle \omega_4, \omega_2 \rangle$$

It is clear that the longer policy does not contain outcome ω_3 which is contained in the shorter policy. This shows that greedy construction of policies is not optimal for TDAMs.

For another example that directly contradicts the statement that the optimal policy is monotonically decreasing in terms of agent utility which is the core of the greedy concession

lemma, consider the following monotonic acceptance model which is a special case of a TDAM:

$$a = \begin{pmatrix} t & \omega_0 & \omega_1 & \omega_2 & \omega_3 & \omega_4 & \omega_5 & \omega_6 & \omega_7 & \omega_8 \\ 0 & 0.9087 & 0.9689 & 0.7115 & 0.7464 & 0.8324 & 0.8715 & 0.6600 & 0.7836 & 0.8372 \\ 1 & 0.8320 & 0.9460 & 0.6462 & 0.6810 & 0.8048 & 0.5904 & 0.5358 & 0.6562 & 0.7934 \\ 2 & 0.8061 & 0.9123 & 0.6030 & 0.4878 & 0.6479 & 0.5686 & 0.5130 & 0.6285 & 0.6937 \\ 3 & 0.6980 & 0.7857 & 0.5387 & 0.2988 & 0.4877 & 0.5280 & 0.4482 & 0.5021 & 0.5893 \\ 4 & 0.6034 & 0.6827 & 0.4343 & 0.2978 & 0.3306 & 0.5007 & 0.3056 & 0.5014 & 0.5025 \\ 5 & 0.4819 & 0.5891 & 0.3277 & 0.2167 & 0.3041 & 0.2851 & 0.2348 & 0.4322 & 0.3639 \\ 6 & 0.4026 & 0.5125 & 0.2302 & 0.1843 & 0.2877 & 0.2763 & 0.2285 & 0.3490 & 0.3554 \\ 7 & 0.3087 & 0.3689 & 0.1115 & 0.1464 & 0.2324 & 0.2715 & 0.0600 & 0.1836 & 0.2372 \end{pmatrix}$$

And let the utility function be:

$$u(\omega_i) = \begin{pmatrix} \omega_0 & \omega_1 & \omega_2 & \omega_3 & \omega_4 & \omega_5 & \omega_6 & \omega_7 & \omega_8 \\ 0.0065 & 0.0257 & 0.0372 & 0.0847 & 0.3885 & 0.4643 & 0.4896 & 0.6835 & 0.7590 \end{pmatrix}$$

By exhaustive search over all policies of length 7, it is straightforward to show that the optimal policy is:

$$pi^* : \langle \omega_8, \omega_8, \omega_8, \omega_8, \omega_8, \omega_7, \omega_8 \rangle$$

This policy is certainly not a concession policy because the outcome ω_7 appears both before and after ω_8 and they have different utility values. □

C Proofs for time complexities

This appendix provides proofs for all time-complexity results mentioned in the paper.

C.1 Time and space complexity of GCA

An efficient implementation of GCA will keep the policy in a sorted list by utility values allowing for insertion of new outcomes in $O(\log(K))$ operations. This means that $sort_{\mathcal{EU}}(\pi \circ \omega)$ takes $O(\log(K))$. For the K such policies, we also need to calculate the expected utility $\mathcal{EU}$ which is an order K operation. This means that a single loop of the algorithm takes $O(K^2)$ operations. This is repeated for each output of the final policy. Thus, the whole algorithm will still require $O(K^2T)$ steps.

The space complexity of the algorithm is $O(K)$ as the algorithm needs only to keep track of the optimal policy which cannot exceed K outcomes because of the assumption that offers are not repeated. □

The algorithm needs to keep the policy which needs $O(T)$ space. Becasue the $\arg\max$ operation does not require any extra storage, the space complexity of GCA is $O(T)$. □

C.2 Time Complexity of QGCA and QGCA$^+$

The time complexity of QGCA can be found by following Algorithm 2. Initialization lines (Line 1-3) can be done in $O(1)$. The inner most loop (Lines 5-10) is executed $O(K)$ times and each iteration of this loop can be executed in $O(1)$ steps leading to $O(K)$ for the loop. Outcome insertion (Line 11) can be done in $O(\log(T))$ steps by keeping an ordered list of outcomes by utility value (same as in GCA). This is dominated by the inner loop though. Updating S, P, and $\mathcal{L}$ is $O(K)$ operations. This implies that the each iteration of the outer loop (Lines 5-14c) takes $O(K)$ steps. This is repeated T times leading to an overall time complexity of $O(KT)$. □

Other than the policy, the algorithm needs to keep track of three arrays (S, P, and $\mathcal{L}$). S and P are $O(T)$ elements, and $\mathcal{L}$ is $O(outcomes)$ elements. This means that the space complexity of QGCA is $O(\max(K, T))$. □

As QGCA$^+$ is the same as QGCA computationally, it has the same time and space complexities.

C.3 Time and space complexity of PBS

The inner loop of Algorithm 3 (Lines 4-6) runs T iterations each taking $O(1)$ operations while the outer loop (Lines 2-8) has T iterations. This leads to a time complexity of $O(T^2)$. □

The algorithm keeps no data strctures that are not kept by QGCA which leads to the same space complexity ($O(max(K,T))$). □

C.4 Time and space Complexity of PA

The inner loop of Algorithm 4 (Lines 7-8) is $O(T)$. This is repeated R times (Lines 2-9) leading to a time complexity of $O(TR)$. □

The algorithm keeps no data strctures that are not kept by QGCA which leads to the same space complexity ($O(max(K,T))$). □

References

1. Nash Jr, J.F.: The bargaining problem. Econometrica: Journal of the Econometric Society pp. 155–162 (1950)
2. Johnson, E., Gratch, J., DeVault, D.: Towards an autonomous agent that provides automated feedback on students' negotiation skills. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 410–418. International Foundation for Autonomous Agents and Multiagent Systems (2017)
3. De La Hoz, E., Marsa-Maestre, I., Gimenez-Guzman, J.M., Orden, D., Klein, M.: Multi-agent nonlinear negotiation for wi-fi channel assignment. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 1035–1043. International Foundation for Autonomous Agents and Multiagent Systems (2017)
4. Mohammad, Y., Fujita, K., Greenwald, A., Klein, M., Morinaga, S., Nakadai, S.: Anac 2019 scml (2019). URL http://tiny.cc/f8sv9y
5. Von Neumann, J., Morgenstern, O.: Theory of games and economic behavior, 2nd rev. Princeton University Press (1947)
6. Rubinstein, A.: Perfect equilibrium in a bargaining model. Econometrica: Journal of the Econometric Society pp. 97–109 (1982)
7. Aydoğan, R., Festen, D., Hindriks, K.V., Jonker, C.M.: Alternating offers protocols for multilateral negotiation. In: Modern Approaches to Agent-based Complex Automated Negotiation, pp. 153–167. Springer (2017)
8. Baarslag, T., Hindriks, K., Hendrikx, M., Dirkzwager, A., Jonker, C.: Decoupling negotiating agents to explore the space of negotiation strategies. In: Novel Insights in Agent-based Complex Automated Negotiation, pp. 61–83. Springer (2014)
9. Baarslag, T., Gerding, E.H., Aydogan, R., Schraefel, M.: Optimal negotiation decision functions in time-sensitive domains. In: 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), vol. 2, pp. 190–197. IEEE (2015)
10. Jonker, C., Aydogan, R., Baarslag, T., Fujita, K., Ito, T., Hindriks, K.: Automated negotiating agents competition (anac). In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2017)
11. Mohammad, Y.: Optimal deterministic time-based policy in automated negotiation. In: PRIMA 2020: Principles and Practice of Multi-Agent Systems: 23rd International Conference, p. 68. Springer Nature (2020)
12. Baarslag, T., Aydoğan, R., Hindriks, K.V., Fujita, K., Ito, T., Jonker, C.M.: The automated negotiating agents competition, 2010–2015. AI Magazine **36**(4), 115–118 (2015)
13. Aydoğan, R., Baarslag, T., Fujita, K., Mell, J., Gratch, J., de Jonge, D., Mohammad, Y., Nakadai, S., Morinaga, S., Osawa, H., et al.: Challenges and main results of the automated negotiating agents competition (anac) 2019. In: Multi-Agent Systems and Agreement Technologies, pp. 366–381. Springer (2020)
14. Lin, R., Kraus, S., Baarslag, T., Tykhonov, D., Hindriks, K., Jonker, C.M.: Genius: An integrated environment for supporting the design of generic automated negotiators. Computational Intelligence **30**(1), 48–70 (2014). DOI 10.1111/j.1467-8640.2012.00463.x. URL http://dx.doi.org/10.1111/j.1467-8640.2012.00463.x

15. de Jonge, D., Zhang, D.: Gdl as a unifying domain description language for declarative automated negotiation. Autonomous Agents and Multi-Agent Systems **35**(1), 1–48 (2021)
16. Chakraborty, S., Baarslag, T., Kaisers, M.: Automated peer-to-peer negotiation for energy contract settlements in residential cooperatives. Applied Energy **259**, 114,173 (2020)
17. Sengupta, A., Mohammad, Y., Nakadai, S.: An autonomous negotiating agent framework with reinforcement learning based strategies and adaptive strategy switching mechanism. In: Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS '21. International Foundation for Autonomous Agents and Multiagent Systems (2021)
18. Klein, M., Faratin, P., Sayama, H., Bar-Yam, Y.: Negotiating complex contracts. Group Decision and Negotiation **12**(2), 111–125 (2003)
19. Baarslag, T., Hindriks, K., Jonker, C.: Effective acceptance conditions in real-time automated negotiation. Decision Support Systems **60**, 68–77 (2014)
20. Baarslag, T., Hendrikx, M.J., Hindriks, K.V., Jonker, C.M.: Learning about the opponent in automated bilateral negotiation: a comprehensive survey of opponent modeling techniques. Autonomous Agents and Multi-Agent Systems **30**(5), 849–898 (2016)
21. Baarslag, T., Hindriks, K., Jonker, C.: A tit for tat negotiation strategy for real-time bilateral negotiations. In: Complex Automated Negotiations: Theories, Models, and Software Competitions, pp. 229–233. Springer (2013)
22. Kawaguchi, S., Fujita, K., Ito, T.: Agentk: Compromising strategy based on estimated maximum utility for automated negotiating agents. In: New Trends in Agent-Based Complex Automated Negotiations, pp. 137–144. Springer (2012)
23. Van Krimpen, T., Looije, D., Hajizadeh, S.: Hardheaded. In: Complex Automated Negotiations: Theories, Models, and Software Competitions, pp. 223–227. Springer (2013)
24. Chen, S., Ammar, H.B., Tuyls, K., Weiss, G.: Optimizing complex automated negotiation using sparse pseudo-input gaussian processes. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pp. 707–714 (2013)
25. Niimi, M., Ito, T.: Agentm. In: Recent advances in agent-based complex automated negotiation, pp. 235–240. Springer (2016)
26. Mori, A., Ito, T.: Atlas3: A negotiating agent based on expecting lower limit of concession function. In: Modern Approaches to Agent-based Complex Automated Negotiation, pp. 169–173. Springer (2017)
27. Mohammad, Y., Greenwald, A., Nakadai, S.: Negmas: A platform for situated negotiations. In: Twelfth International Workshop on Agent-based Complex Automated Negotiations (ACAN2019) in conjunction with IJCAI (2019)
28. Fujita, K., Aydogan, R., Baarslag, T., Hindriks, K., Ito, T., Jonker, C.: Anac 2016 (2016). URL http://web.tuat.ac.jp/~katfuji/ANAC2016/
29. Ilany, L., et al.: The fourth automated negotiation competition. In: Next Frontier in Agent-Based Complex Automated Negotiation, pp. 129–136. Springer (2015)