win.win | a negotiating agent for mutual benefit

JOCHEM BROUWER, SAUL GEBHARDT, WOUTER VAN HARTSKAMP, and OTTO MÄTTAS, Utrecht University, The Netherlands

ACM Reference Format:
Jochem Brouwer, Saul Gebhardt, Wouter van Hartskamp, and Otto Mättas. 2021. win.win | a negotiating agent for mutual benefit. 1, 1 (April 2021), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BOA IMPLEMENTATION

The implementation of the BOA agent follows the description of the components in report 2. On a high level, this means that win.win is a cooperative agent, who strives to reach the best solution possible for both agents.

The base of the code written for win.win stems from the default BOA components that can be found in the folder `bilateralexamples/boacomponents` that can be found in Genius. The first iteration of win.win took advantage of these components. Afterwards, the strategies were altered completely to make sure that the components matched the design of the BOA agent from the previously mentioned report 2.

At the start of a negotiation session, win.win makes a random bid, which lies between 70% and 80% of the maximum utility it can expect from this negotiation. Afterwards, win.win starts making some random bids, slowly raising both lower and upper bounds that the bid should be between. This process follows the time of the session in a linear fashion.

In short, the bids are between 70% and 80% of the maximum expected utility at the start, then rise slowly to bids that are between 75% and 85% of the maximum utility. The process of raising bid bounds persists until a random bid is generated between 90% and 100% of the expected maximum utility. This informs the agent to switch tactics. win.win then goes from “opening phase” tactics to the “main phase” tactics.

This process of linearly increasing the maximum expected utility in the "opening phase" is to convince the opponent into believing win.win is conceding. This is actually not the case. The "opening phase" of the negotiation is defined in this way, since a somewhat well defined opponent model is necessary for the “main” bidding strategy of win.win.

Win.win uses the opponent model to calculate an estimation of the Pareto frontier in the negotiation. At this point, win.win goes into the second phase of the negotiation. This second part starts once win.win makes a bid with a utility of 1 for itself. This bid is part of the Pareto frontier. After this, win.win ’walks’ down the Pareto frontier, making bids that are part of the Pareto frontier. Note that the Pareto frontier is approximated and is very likely to be imperfect. Pulling bids from the approximated Pareto frontier ensures that the bids that are being sent out by win.win are still good, but that win.win is also conceding and actively looking towards a solution of the negotiation.

The acceptance strategy is just a direct implementation of the acceptance strategy as discussed in the previous report. There are three different acceptance conditions, and the acceptance strategy is based on a combination of those conditions. The first acceptance condition checks if the received utility is higher than or equal to the utility of the next bid that win.win is going to send out next, which is linearly scaled by two factors: multiplying this utility by some percentage and adding a constant amount to enforce a “gap”. The second acceptance condition checks if the utility of the received bid is higher than or equal to some constant and the final acceptance condition checks whether a certain time instance has passed and the received bid is higher than or equal to the average over a certain percentage of past bids. The idea behind the acceptance strategy is to accept if we are about to send out a bid which is lower than the bid the opponent has just sent to us, assuming that a certain time instance has passed. The other acceptance conditions are meant to ensure that win.win does not use this acceptance condition when win.win is still attempting to form an opponent model, and is effectively, sending out quasi-random bids.

Below is a list which describes the different phases of the agent:

- Bidding strategy initial phase: this phase is used to collect bids in order to approximate the Pareto frontier. The value is 0.2
- Bidding strategy main phase: this phase is used to walk over the approximated Pareto frontier and starts by sending out the highest utility bid from our side, before linearly conceding to the minimum utility (0.5). This starts at $t = 0.2$.
- Acceptance strategy initial phase: in this phase, bids are accepted if they have an utility of 0.675 or higher. This phase is only during $t < 0.05$.
- Acceptance strategy default phase: in this phase, bids are accepted if the received bid is 2% higher than the bid we are about to send. This phase is within $t \in [0.05, 0.95)$
- Acceptance strategy conceding phase: in this phase, the agent tries to find a bid which is reasonably well so it can accept. The agent really strives to accept a bid during the negotation. In this phase, a bid is accepted if it has an utility higher than or equal to the average utility of the last 20% of the bids sent right before entering this phase. The acceptance strategy default phase rules hold as well. This happens during $t >= 0.95$.

Authors’ address: Jochem Brouwer, j.brouwer8@students.uu.nl; Saul Gebhardt, s.a.gebhardt@students.uu.nl; Wouter van Hartskamp, w.vanhartskamp@students.uu.nl; Otto Mättas, o.mattas@students.uu.nl, Utrecht University, P.O. Box 80125, Utrecht, Utrecht, The Netherlands, 3508 TC.

1.1 Opponent Model

Significant efforts were made to develop a component which would incorporate Gaussian distributions with a Bayesian estimation algorithm.

To properly express the opponent's bid and its importance, the agent was designed to keep track of the opponent's bid minimum and maximum threshold values in real time. The agent would store and update mapped values as time progresses through Bayesian learning. Additionally, Gaussian probability distributions would be used to inform and improve the learning process.

Unfortunately, the lack of expertise with Java made the team concede to a simpler approach and to reconsider the efforts. As explained by [Baarslag et al. 2014], opponent models make a surprisingly small contribution to the performance of a negotiation strategy. There is little difference between not using a model and using any of the state of the art models. As a caveat, it is also established that having an opponent model can still bring significant difference in the context of a tournament where every small contribution can determine the final ranking of the agent. To conclude, once an agent employs a reasonable opponent model, there is not much more to gain after that.

For win.win, the opponent model was chosen from amongst the available example models within Genius. Eventually, the Hard-Headed opponent model was chosen as it is in principle not very different from the initially envisioned design.

As explained by [van Krimpen et al. 2013], the learning function is a greedy reinforcement learning function, which updates the issue weights and value utilities of the preference profile after each bid. The algorithm is split into two parts, one for the weights the other for the issue value utilities.

```
if $T == 1$ then
    Initialize preference profile
else
    set index to where $V(T) == V(T_1)$
    $W(index) = W(index) + \epsilon$
    $W = W / \sum_{n=1}^{N} w_n$
    $V(index) = V(index) + 1$
end
```

The algorithm checks for issue values that the opponent has kept unchanged over all its offered bids. It then adds a value ε to each of those issue weights and the weights are normalized every round. The values get their utilities incremented each time they remain unchanged. Since these utilities are not required to be percentages, they only normalized to their maximum value per issue at the time the utility is calculated. As also noted by the authors, the function always manages to identify the most valuable bid, and the least valuable bid for the opponent.

1.2 Opponent Model Strategy

The opponent model strategy has not been changed from the default `BestBid.java` which is found in the Genius installation folder. The Opponent Model Strategy (OMS) has retained every function that was already present in the initial OMS, to ensure that the other components function like other BOA components, in the sense that they can be mixed and matched with other BOA components as one sees fit. Due to time constraints, little to no experimentation was done on the OMS and methods that are available in the OMS. In the current state of win.win, the OMS is not necessary for any component to function correctly. However, it might be the case that large improvements on negotiations under certain conditions can be made by utilizing the OMS.

1.3 Bidding Strategy

The agent is divided up in two phases: the "opening" phase and the "main" phase. In the opening phase, the agent tries to fool the other agent by sending sub-optimal bids to the agent. As mentioned before, the bids start within an utility range between 0.7 and 0.8. A random bid is picked from any possible bids within this range. This opening phase takes until the time reaches `switchTime`. During this time, the range linearly converges with respect to the time to the range 0.9 - 1. If the other agent assumes that our first bid is a bid which we valuate at utility 1, then the agent has succesfully fooled the opponent. Note that the utility of the bids the agent sends is raised on average. If the opponent assumes that the agent first tries high bids and then slowly starts to concede in this initial phase, then we have also fooled the opponent. The idea is to disrupt the opponent model of the opponent, such that they cannot rely upon their model, since it is simply wrong.

It is clear that the agent initially offers bids which increase in utility for the agent itself. The reason for this is thus to fool the other agent.

The opening bid uses the same method as getting a new bid within some range: therefore, the opening bid returns a random bid with an utility between 0.7 and 0.8.

During the initial phase, bids are collected from the opponent. The opponent model is also trained. The agent tries to use the past bids in combination with the opponent model to approximate the Pareto frontier. If, in the end, an agreement is reached which does not lay on the actual Pareto frontier, then this is suboptimal: at least one of the agent could have sent a bid with a higher utility for themselves, which the other agent would still accept.

When entering the "main" phase, the agent would have sent a bids with an utility between 0.9 and 1.0 during the past few bids. This phase walks down the (approximated) Pareto frontier from our viewpoint. Initially, it sends a bid which has utility 1. When time passes, it linearly chooses bids from the Pareto frontier based upon the current time and the time left, where this time is scaled between 1 and `switchTime`. The agent thus slowly concedes over the approximated Pareto frontier.

1.4 Acceptance Strategy

The acceptance strategy originates from the `AC_Uncertain.java` component found in the Genius software. This acceptance strategy was chosen over the other default acceptance strategy present in the Genius software, because this acceptance strategy already considers preference uncertainty, which is something that win.win should be able to deal with.

The standard method of the acceptance strategy is called `determineAcceptability()`. This method is responsible for the returning of an `Actions.Accept` (which would conclude the negotiation) or `Actions.Reject` (which would mean that win.win sends its created bid to the opponent).

The first thing that this method does is to save which is the last bid received by the opponent and the bid that win.win is going to send out next. If no bid has been received or no bid has been sent yet, win.win sends out a bid regardless. This entails that it is impossible for win.win to accept a bid in the very first round.

The next thing that the method checks is whether there is preference uncertainty or if there is a normal utilityspace. If there is preference uncertainty, the acceptance strategy enters the 'default acceptance strategy' under uncertainty which was already present in the original code.

If it is the case that there is preference uncertainty, the acceptance strategy attempts to get the bid order of win.win's own preference profile. Win.win then checks if this bid is present in the bid order. If this is the case, win.win checks if the received bid is in the first 10% of the bid order. If this is the case, win.win accepts the bid. If this is not the case, win.win rejects and sends back a bid to the opponent. This is not new logic, but instead inherited from the parent logic the acceptance strategy extends.

If a normal utility space is present, the 'normal' acceptance strategy will be consulted. This acceptance strategy contains three different acceptance conditions that will first be discussed before discussing the overarching strategy.

The first acceptance condition is `Acceptance_Next()`. This acceptance condition uses two variables, α and β. α indicates a percentage, i.e. 1.02 indicates 102%, or 2% higher than the default value, and β is a constant. This acceptance condition calculates the utility of the bid that was just received and calculates the utility of the bid win.win is about to send out. It then is calculated whether $\alpha \times \texttt{ReceivedUtil} + \beta \geq \texttt{utilToSend}$. This function returns `true` if and only if the received utility times some percentage is at least β higher than the utility of the bid that is about to be sent out. If this is the case, true is returned, otherwise this acceptance condition returns false.

The second acceptance condition is `Acceptance_Const(double alpha)`. This function checks if the utility of the opponents' bid is higher than or equal to some constant α. This α is configured at a value of 0.675, which was deemed a respectable utility to receive, since it was the higher than the average utility which was received in some ANAC competitions.[Baarslag et al. 2013]

The final acceptance condition is `Acceptance_Time(double T)`. This acceptance condition checks if the negotiation time is equal or greater than T and returns true if this is the case. There are two different static values of T, `time_init` which is 0.05 and `time_final`, which is set to 0.95. With these variables, the negotiation phase is divided up into three seperate stages for the acceptance strategy. The first phase is the first 5% of the time, the second phase is the last 5% of the time and the middle phase is everything in between these two phases.

With these three acceptance conditions, the acceptance strategy is formed. Whenever the acceptance phase of a round is started and there is a normal utilityspace, win.win checks first if `Acceptance_Const(this.alpha_init)` and if the current time $t \leq 0.05$. If this is the case, win.win accepts. (This only happens if a bid is received with a higher utility than 0.675 in the first 5% of the negotiation time).

After this has been checked, the following is checked; first of all it must be the case that `Acceptance_Next()` holds or that the negotiation has reached the final phase and the bid has a utility equal or higher than the average utility of all the bids that win.win has received in the last 20% of the negotiation. If either of these conditions holds, it must also be the case that the current time is greater or equal than 0.05, in other words, that at least 5% of the negotiation has concluded. If this `if`-statement returns true, the bid is accepted and win.win returns `Actions.Accept`. Otherwise, the acceptance strategy of win.win returns `Actions.Reject` and the bid that has been created in the bidding strategy is sent to the opponent. The 20% average is only calculated once and is thus not updated dynamically. It is possible to change this: updating the average can be done in a very efficient way by removing the utility of the bid(s) which was/were in the average list, but is not anymore, and add the new bid to this sum before dividing by the total amount of items.

1.5 Preference Uncertainty

Due to time constraints the agent does not incorporate any extra logic to handle uncertain preferences. Most of the time was spend on designing the bidding strategy. The agent could be improved in the future for situations were the uncertain profiles arise.

2 PERFORMANCE EVALUATION

Significant effort was made to create a baseline measurement of the performance of the agent. This would allow the components to be improved on, effectively giving a performance standard to measure against.

Additionally, the agent was not considered against an opponent with an identical preference profile. First, tournaments were ran with such scenarios. This proved to provide noisy results, often in the form of both agents receiving their expected utility. As the agent is designed to operate as close to the real world as possible, and such scenarios do not occur in real world, the scenarios were removed from the analysis.

2.1 Party Domain

The first tournament to test the performance of the Win.Win agent was performed in the party domain. This domain encompasses the planning for a party including the types of food, drinks, location, invitations, music, and the way the cleanup should be done.

In this domain there are 3072 unique bids possible, meaning the party domain is a moderately large field. In this tournament the opposing agents were decided to be the `HardHeaded`, `Gahboninho`, and `IAMhaggler2011` as provided by the GENIUS package. This line-up is completed by our own Win.Win agent, so the performance against the same agent can be tested.

Over a gruelling 76 negotiations against each opponent, win.win was determined to be the winner in one case. This was against itself. Average utilities against the other opponents was 0.2 to even 0.3

lower than the other agent. The only reason the Win.Win agent seemed to fare well against itself was due to the rounds being only 3.875 bids long, placing the agreements that were reached square in the semi-randomized bidding based `openingPhase()`. There are some metrics in which the agent did quite well, as it was designed to do so. The average distance to Pareto optimal solutions against the other agents is 0.008562176 and reached an efficient outcome 161 times out of the 304 negotiations, meaning that the efficiency of the outcomes was top of the line. The agents even managed to reach a Nash solution 22 times out of the 304 negotiations, although the average distance to the Nash point was only 0.191319444. The average social welfare was also quite high, 1.684141481, which can be fully attributed to the Pareto optimal solutions being reached often.

2.1.1 Under Certainty. Utilising the same party domain as before, a new factor was introduced: Certainty. GENIUS includes a feature to evaluate how a given agent performs when the agent has imperfect knowledge of the entire utility space. This is simulated by replacing the output of getSortedOutcomeSpace with a ranking of a selection of bids lacking any utility scores for the agent. Only the ranking is left at this point. The same agents were used for this analysis, these being the `HardHeaded`, `Gahboninho`, `IAMhaggler2011`, and the Win.Win agent. For this part a selection was made from the previous preference profiles, now only containing `party1_utility` and `party2_utility` as this will be the basis for the preference profiles with uncertainties used in subsection 2.1.2. This significantly decreases the number of total negotiations from 304 to only 8. The trend set in subsection 2.1 continues here. Average utility against other opponents is again trailing by 0.2 compared to the other agents. Pareto distance is again excellent with an average value of 0.003968333, with Nash distance 0.213921667 on average. The same can be said for social welfare, being 1.620728333 on average. Again the same conclusion can be made for the Win.Win agent against itself, the speed at which the agreement is made indicates random bids being accepted.

2.1.2 Under Uncertainty. The same setup as in the previous subsection applies, with the difference being the preference profiles having uncertainty in some cases. Three distinct cases will be discussed: Win.Win having certainty and the opponent not having certainty, vice versa, and lastly everyone having uncertainty. These test have been done with multiple levels of uncertainty, ranked from u50 to u10. This means that in the u50 scenario 50 random bids are in the ranking while at u10, 10 random bids are in the ranking.

- Certain vs Uncertain
  Having certainty while the opponent has to deal with uncertainty is a distinct advantage from a instinctual perspective on a human level. This is doubly so for artificial agents as there is no gut feeling involved. This is clearly visible in the data as the average utility gap between the other agents and Win.Win is reduced to only being 0.06 and Win.Win actually coming out ahead not only against itself but `IAMhaggler2011` as well. Pareto distance however has suffered significantly, dropping to 0.146139444 on average. This can be explained by the opponent not knowing exactly what it prefers and as such throwing off the Pareto front calculator in the Win.Win Bidding Strategy. Nash distance and social welfare both did not change much, being 0.20724 and 1.61298 respectively.

- Uncertain vs Certain
  As stated in the point above, certainty is a big advantage and it is clear that our opponents took advantage of this as their average utility sky-rocketed to a whopping 0.9469 on average in u10. Their average utility lead however did diminish somewhat compared to the baseline scenario from a comfortable 0.2 to 0.14 in the u10 scenario. The average utility for the opponents is clearly lower in the u50 scenario 0.78377. The lead however is almost gone in this scenario with Win.Win closing in to an average 0.75495 utility. The opposite can be seen in the other metrics, where the Pareto distance dropped from a 0.04 in the u10 scenario to 0.19 in the u50 scenario. The same goes for Nash distance dropping from 0.14 to 0.23 and again for social welfare dropping from 1.75 to 1.54.
  All this is quite counter-intuitive but can be explained by the combination of the Win.Win Acceptance and Bidding strategies. The Acceptance strategy gets rather bullish when it detects that it is in a uncertainty scenario and only accepts top 10% bids, which means higher than the highest bid in the u10 scenario. As such not a lot of bids get accepted and this requires the hardliner agents to start conceding for a deal to be made. This means that the Win.Win agent gets more time to gather information on the hardliners and as such the Pareto frontier calculation gets quite accurate around the opponent even though our side is still fluctuating quite severely. This advantage diminishes with higher numbers of bids in the ranking. The acceptance bar of 10% is reached more easily, the agent only bids in the part of the Pareto frontier that is less well-defined and gives more up dropping all other metrics. However since the bids are less well-defined the utilities for the opponent and Win.Win reach parity.

- Uncertain vs Uncertain
  In the scenario of full uncertainty the same pattern is seen as the Uncertain vs Certain part applies. Lower uncertainties for Win.Win lead to a lower gap in average utilities between the other agents and Win.Win. Higher certainties lead to a higher gap in average utilities between the other agents and Win.Win. However the other metrics stay practically the same. This is due to the Pareto calculator losing accuracy due to having two agents without a clear preference.

2.2 Component Evaluation

2.2.1 Acceptance Strategy. A tournament was ran between the agent and all possible combinations of an opponent where only the acceptance strategy component differed from the agent. Therefore, the agent runs using its own acceptance strategy against the same agent, except that now the acceptance strategy is changed. The results show that almost all negotiations the agent with its own strategy ends up with a higher average utility than the agent with a different agent. However, there are some exceptions. The agents with acceptance strategies `AC_TheNegotiatorReloaded_bd_c_b_a_t_ad`, `AC_Next_b_a`, `AC_InoxAgent` score better. The biggest difference is in `AC_InoxAgent`: here the Inox agent scores appoximately 0.751 on average, while the agent using its own strategy only scores approximately 0.655 on average. Against `AC_Gahboninho`, the agent implementing the Gahboninho acceptance strategy only scored approximately 0.326 on average, while the agent using its own acceptance strategy scored approximately 0.814 on average, which is a huge distance. It should also be noted that the agent scores better against the `AC_HardHeaded` strategy, approximately 0.735 against 0.669.

2.2.2 Opponent Model. The agent did not use a custom opponent model. There is little to be gained by replacing the opponent model. That said, a tournament was ran between the agent and all possible combinations of an opponent where only the opponent model component differed from the agent.

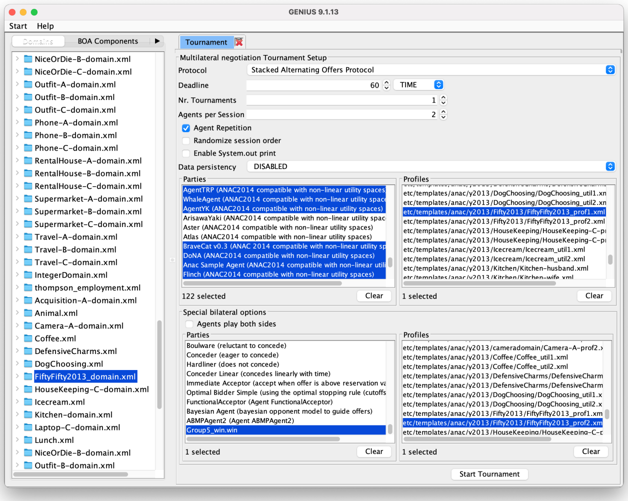
2.3 Generality

To test the generality of win.win, a total of 16 tournaments were run in the Genius GUI. These tournaments were run on 4 different domains, with 4 tournaments that were run on each domain. Each domain was both tested in a round based negotiation as well as a time based negotiation, and also with preference certainty and preference uncertainty. Time-based tournaments had 60 second rounds, while round-based tournaments consisted of 60 rounds.

The performance in these tournaments is quite important, since the domain that win.win will be negotiating in during the real tournament is unknown. Therefore, win.win should be able to perform well in domains of any size. Analysis was done by combining the results of win.win in every tournament in the same conditions. 4 Analyses have been conducted; the performance of win.win in time-based negotiations with preference certainty, the performance of win.win in time-based negotiations with preference uncertainty, the performance of win.win in round-based negotiations with preference certainty, and the performance of win.win in round-based negotiations with preference uncertainty.

The tournament were set up as can be seen in 1.

For each domain, the first preference profile was used by the opposing agents, and the second preference profile was used by win.win. The four domains that was tested on were: FiftyFifty2013, Domain8, Ultimatum and Animal. These domains were chosen for the following reasons: FiftyFifty2013 is a domain with a small amount of issues (only 1) and a large amount of values in this domain (10). Domain8 is a domain with 8 different issues, but each issue only contains 3 values. The result of this is that there are a lot of different bids possible, way more than in FiftyFifty2013. The Ultimatum is a crossbreed between FiftyFifty2013 and Domain8; there are only 2 different issues, and each issue has 3 different values. As a result, there are even less unique bids in the Ultimatum domain than there were in the FiftyFifty2013 domain. The final domain is the animal domain. The created bids in this domain are animals from


```
GENIUS 9.1.13
Start   Help
Domains | BOA Components
  NiceOrDie-B-domain.xml
  NiceOrDie-C-domain.xml
  Outfit-A-domain.xml
  Outfit-B-domain.xml
  Outfit-C-domain.xml
  Phone-A-domain.xml
  Phone-B-domain.xml
  Phone-C-domain.xml
  RentalHouse-A-domain.xml
  RentalHouse-B-domain.xml
  RentalHouse-C-domain.xml
  Supermarket-A-domain.xml
  Supermarket-B-domain.xml
  Supermarket-C-domain.xml
  Travel-A-domain.xml
  Travel-B-domain.xml
  Travel-C-domain.xml
  IntegerDomain.xml
  thompson_employment.xml
  Acquisition-A-domain.xml
  Animal.xml
  Camera-A-domain.xml
  Coffee.xml
  DefensiveCharms.xml
  DogChoosing.xml
  FiftyFifty2013_domain.xml
  HouseKeeping-C-domain.xml
  Icecream.xml
  Kitchen-domain.xml
  Laptop-C-domain.xml
  Lunch.xml
  NiceOrDie-B-domain.xml
  Outfit-B-domain.xml

Tournament
Multilateral negotiation Tournament Setup
Protocol             Stacked Alternating Offers Protocol
Deadline             60   TIME
Nr. Tournaments      1
Agents per Session   2
[✓] Agent Repetition
[ ] Randomize session order
[ ] Enable System.out print
Data persistency     DISABLED

Parties
AgentTRP (ANAC2014 compatible with non-linear utility spaces)
WhaleAgent (ANAC2014 compatible with non-linear utility space
AgentYK (ANAC2014 compatible with non-linear utility spaces)
ArisawaYaki (ANAC2014 compatible with non-linear utility spac
Aster (ANAC2014 compatible with non-linear utility spaces)
Atlas (ANAC2014 compatible with non-linear utility spaces)
BraveCat v0.3 (ANAC 2014 compatible with non-linear utility sp
DoNA (ANAC2014 compatible with non-linear utility spaces)
Anac Sample Agent (ANAC2014 compatible with non-linear utili
Flinch (ANAC2014 compatible with non-linear utility spaces)
122 selected                                     Clear

Profiles
etc/templates/anac/y2013/DogChoosing/DogChoosing_util1.xn
etc/templates/anac/y2013/DogChoosing/DogChoosing_util2.xn
etc/templates/anac/y2013/Fifty2013/FiftyFifty2013_prof1.xml
etc/templates/anac/y2013/Fifty2013/FiftyFifty2013_prof2.xml
etc/templates/anac/y2013/HouseKeeping/HouseKeeping-C-pı
etc/templates/anac/y2013/HouseKeeping/HouseKeeping-C-pı
etc/templates/anac/y2013/Icecream/Icecream_util1.xml
etc/templates/anac/y2013/Icecream/Icecream_util2.xml
etc/templates/anac/y2013/Kitchen/Kitchen-husband.xml
etc/templates/anac/y2013/Kitchen/Kitchen-wife.xml
1 selected                                       Clear

Special bilateral options
[ ] Agents play both sides
Parties
Boulware (reluctant to concede)
Conceder (eager to concede)
Hardliner (does not concede)
Conceder Linear (concedes linearly with time)
Immediate Acceptor (accept when offer is above reservation va
Optimal Bidder Simple (using the optimal stopping rule (cutoffs
FunctionalAcceptor (Agent FunctionalAcceptor)
Bayesian Agent (bayesian opponent model to guide offers)
ABMPAgent2 (Agent ABMPAgent2)
Group5_win.win
1 selected                                       Clear

Profiles
etc/templates/anac/y2013/cameradomain/Camera-A-prof2.x
etc/templates/anac/y2013/Coffee/Coffee_util1.xml
etc/templates/anac/y2013/Coffee/Coffee_util2.xml
etc/templates/anac/y2013/DefensiveCharms/DefensiveCharm
etc/templates/anac/y2013/DefensiveCharms/DefensiveCharm
etc/templates/anac/y2013/DogChoosing/DogChoosing_util1.x
etc/templates/anac/y2013/DogChoosing/DogChoosing_util2.x
etc/templates/anac/y2013/Fifty2013/FiftyFifty2013_prof1.xm
etc/templates/anac/y2013/Fifty2013/FiftyFifty2013_prof2.xm
etc/templates/anac/y2013/HouseKeeping/HouseKeeping-C-p
1 selected                                       Clear

Start Tournament
```

Fig. 1. Tournament setup

the following class: there is one mammal, one bird, one reptile, one amphibian and one primate. This domain feels the most 'normal' compared to the other previously mentioned domains and looks a lot more like the party domain than the other previously mentioned domains. This domain is a fair amount smaller than Domain8, but is a lot bigger than FiftyFifty2013 and the Ultimatum domain. The tournament was set up in such a way that win.win was always the second agent in the negotiation, and the first agent was the agent win.win was negotiating with. The agents that win.win negotiated with were part of the Genius GUI. Win.win negotiated with every agent that was in the list of parties in Genius by default, but after some tries, some agents were found to be unusable. The following agents were excluded from participating in the tournaments:

- UIAgent
- UIAgentExtended
- CounterOfferHumanNegotiationParty
- Exp_Rubick
- GroupY
- Libra
- Sontag
- CaduceusDC16
- ShahAgent
- Mamenchis
- AgentKN
- Rubick
- ArisawaYaki
- Aster
- Atlas

2.3.1 Analysis of time-based negotiations under preference certainty. As you can see in Table 1, the performance of user 2 (win.win) is on average a bit better than the performance of other agents. What also stands out is the high amount of crashes of the negotiation. Upon further inspection, it seems to be that within Domain8, there

Agreements	Disagreements	Crashes of negotiation
276	68	144

User 1 average utility	User 2 average utility
0.37	0.40

Avg. dist. to Pareto	Avg. dist to Nash	Avg. of Social Welfare
0.22	0.22	0.89

Average run time (s)	Average rounds	Average t-value
19.78	21828.34	0.33

Table 1. Analysis of time-based negotiations with preference certainty

Agreements	Disagreements	Crashes of negotiation
195	242	51

User 1 average utility	User 2 average utility
0.33	0.35

Avg. dist. to Pareto	Avg. dist to Nash	Avg. of Social Welfare
0.34	0.38	0.99

Average run time (s)	Average rounds	Average t-value
39.55	41555.90	0.66

Table 2. Analysis of time-based negotiations with preference uncertainty

were only 37 agreements, and a staggering amount of 85 failed negotiations, with the most of those originating from crashes of one of the agents.
We can conclude from this that Domain8 might not have been the best domain to test on, since either win.win is unable to do negotiations in this domain, or there is a large amount of agents in the Genius GUI that cannot do negotiations in this domain. Whenever agreements were made, the negotiation concluded very quickly. The average utility that win.win got in the tournaments on the Ultimatum and FiftyFifty domain was very low, especially compared to the average utilities received in the other two domains. However, it should be noted that the utility that win.win received was a bit higher in every domain than the average utility of its opponents.
When looking at the time the negotiation took on average, it becomes clear that there was a lot more time that the negotiation could have taken. The t-value states how much percentage of the total time was utilized on average for each negotiation. A t-value of 0.33 is quite low, it was possible for win.win to use more time in each negotiation to find better bids.

2.3.2 Analysis of time-based negotiations under preference uncertainty. Whenever win.win negotiates in time-based negotiations with preference uncertainty, the failure rate is even higher than in Table 1. As you can see in Table 2, almost half of the negotiations failed due to no party accepting at the end of the negotiation. Moreover, average utility was lower across the board, and the distance to both Pareto and Nash was higher on average. The one positive point to take away was that the social welfare was a bit higher than under preference certainty.

Finally, on average, negotiations took about twice the time. This is also already deducible from the amount of disagreements after time ran out, which would be the cause that the average run time also goes up drastically.

From these two analyses, it is clear that win.win performs better under preference certainty than under preference uncertainty, but this is a common property on many agents. For generality's sake, it seems like win.win does not perform that well in longer, time-based negotiations. The high amount of crashes and number of disagreements can indicate that win.win is either: unable to generate a good bid that the opponent wants to accept or the acceptance strategy of win.win could use some tweaking.

After these tournaments were run, win.win was put in tournaments on the same domains, but now on round-based negotiations.

Agreements	Disagreements	Crashes of negotiation
368	98	22

User 1 average utility	User 2 average utility
0.47	0.44

Avg. dist. to Pareto	Avg. dist to Nash	Avg. of Social Welfare
0.22	0.22	1.07

Average run time (s)	Average rounds	Average t-value
0.63	34.58	0.58

Table 3. Analysis of round-based negotiations with preference certainty

2.3.3 Analysis of round-based negotiations under preference certainty. In round based negotiations under preference certainty, the percentage of agreements is way higher than in time-based negotiations, even if the preference profiles in both cases do not have any preference uncertainty. Comparing Table 3 to Table 1, there are way more successful bids, which mostly has to do with the agent crashing way less often. This can indicate that the strategy that win.win employs is very computationally demanding over long

negotiating sessions, which might cause crashes. Another thing to note is that the average utility of the opponents of win.win went up considerably more than the average utility of win.win itself, to the point where win.win on average performs worse.
However, win.win is supposed to be a cooperative agent, and the social aspect of win.win really starts to show in these tournaments; the social welfare is considerably higher and the average distance to both Pareto and Nash is way lower. Since it is a round-based negotiation, sessions go super quickly time-wise, but also it seems to be the case that again not all of the time is utilized.

2.3.4 Analysis of round-based negotiations under preference uncertainty. Following the results of the previous Tables 1,2 and 3, it was to be expected that the performance in this condition would be worse than the one in Table 3, but by comparing Table 1 with Table 3, it was to be expected that performance in Table 4 would be better than the performance of Table 2. This does not seem to be the case, however. On every condition, the round-based negotiation performs worse than the time-based negotiation whenever we're dealing with preference uncertainty, making the performance in 4 the worst of all the conditions.
Win.win was tested on a lot of different domains against a lot of different agents that were delivered with a new installation of Genius. A lot of these agents have performed in ANAC competitions, which are agents that could be considered to be well-performing agents. The agent performed on average a bit better when looking at the average utility when looking at every condition except the round-based negotiations with preference certainty. The domains that the tournaments were run on were varying in size, and should be representative for any random domain being thrown at win.win.

Agreements	Disagreements	Crashes of negotiation
168	280	40

User 1 average utility	User 2 average utility
0.31	0.32

Avg. dist. to Pareto	Avg. dist to Nash	Avg. of Social Welfare
0.38	0.40	0.98

Average run time (s)	Average rounds	Average t-value
0.54	49.22	0.82

Table 4. Analysis of round-based negotiations with preference uncertainty

2.4 Future

There are a lot of changes that can be made to win.win to ensure that it will be able to perform really well in a lot of different preference profiles. First of all, the opponent modelling strategy is the default hardheaded one that comes with Genius. An easy improvement that could be made with this is to apply Gaussian learning to the opponent model, as was mentioned before in report 2. At the very least, this would have left win.win with more data and research already conducted on the opponent model, and could be used to make comments on the quality of the opponent model that is currently being used. The opponent model strategy is also a component that was not used by win.win. Win.win did not use any elements of the opponent model strategy, since it was the most unknown component of the BOA component, which was not mentioned in the literature that was read before the strategy of win.win was designed. The researchers were made aware of the fact that the opponent model strategy was an element that needed to be implemented extensively when they received the feedback of the second report just before the deadline of the agent itself.
Furthermore, win.win can improve a lot if it's better equipped to deal with preference uncertainty. In the current state, the acceptance strategy (which utilizes the default acceptance strategy under uncertainty) is the only part to really implement an acceptance strategy under uncertainty.
Under uncertainty there are in general some huge improvements to be made. The bidding strategy is not able to calculate the Pareto frontier successfully, and therefore is unable to create good bids that are part of the (perceived) Pareto frontier.
Considering the troubles Win.Win has against Hardliner opponents a Tit-for-Tat strategy was experimented with after the results of this report. Some test were run but since time limited our opportunities to test the agent thoroughly, it was not submitted as the main agent.

3 ACKNOWLEDGMENTS

The team wishes to extend gratitude to professor Dr. Tim Baarslag and instructor Sorin Dragan for creating an inspiring challenge and helping us tackle it.

REFERENCES

Tim Baarslag, Alexander Dirkzwager, Koen Hindriks, and Catholijn Jonker. 2014. The Significance of Bidding, Accepting and Opponent Modeling in Automated Negotiation. *Frontiers in Artificial Intelligence and Applications* 263. https://doi.org/10.3233/978-1-61499-419-0-27
Tim Baarslag, Koen Hindriks, and Catholijn Jonker. 2013. *Acceptance Conditions in Automated Negotiation.* Springer Berlin Heidelberg, Berlin, Heidelberg, 95–111. https://doi.org/10.1007/978-3-642-30737-9_6
Thijs van Krimpen, Daphne Looije, and Siamak Hajizadeh. 2013. *HardHeaded.* Springer Berlin Heidelberg, Berlin, Heidelberg, 223–227. https://doi.org/10.1007/978-3-642-30737-9_17