GANGSTER: an Automated Negotiator Applying Genetic Algorithms

Dave de Jonge and Carles Sierra

Abstract Negotiation is an essential skill for agents in a multiagent system. Much work has been published on this subject, but traditional approaches assume negotiators are able to evaluate all possible deals and pick the one that is best according to some negotiation strategy. Such an approach fails when the set of possible deals is too large to analyze exhaustively. For this reason the Annual Negotiating Agents Competition of 2014 has focused on negotiations over very large agreement spaces. In this paper we present a negotiating agent that explores the search space by means of a Genetic Algorithm. It has participated in the competition successfully and finished in 2nd and 3rd place in the two categories of the competition respectively.

1 Introduction

In this paper we present the agent that we have developed for the Annual Negotiating Agents Competition 2014 (ANAC'14). Our agent is called GANGSTER, which stands for Genetic Algorithm NeGotiator Subject To alternating offERs.

Theoretical work on negotiations has been published as early as 1950 [9]. The study of negotiations from an algorithmic point of view however is a much more recent topic. In [3, 4] the authors propose a strategy that amounts to determining for each time t which utility value should be demanded from the opponent (the *aspiration level*). However, they do not take into account that one first needs to find a deal that indeed yields that aspired utility level. They simply assume that such a deal always exists, and that the negotiator can find it without any effort.

To overcome that shortcoming, work on negotiations on large spaces was done in [7, 8]. They chose a model in which utility functions are nonlinear over an abstract

Dave de Jonge
IIIA-CSIC, Campus de la UAB s/n, Bellaterra, Catalonia, Spain, e-mail: davedejonge@iiia.csic.es

Carles Sierra
IIIA-CSIC, Campus de la UAB s/n, Bellaterra, Catalonia, Spain, e-mail: sierra@eiiia.csic.es

vector space that represents the set of possible deals, but such that for any given deal x its utility value can still be calculated by solving a linear equation. A similar model is also used for the ANAC'14 competition, as we will see in Section 2.2.

The idea of large spaces with non-linear functions was carried even further by ourselves in [5, 6], where we introduced the Negotiating Salesmen Problem: a negotiation scenario in which not only the number of possible deals is very large, but, given a deal it is also very hard to determine its utility value as it requires solving a Traveling Salesman Problem. In order to tackle that domain we applied a Branch & Bound algorithm adapted to negotiations.

In this paper however, we apply Genetic Algorithms to perform the search. Furthermore, we present a new acceptance strategy that introduces the concept of reproposing, and we present a new bargaining strategy that not only depends on an aspiration level, but also on the distance between a new proposal and the proposals that have been made before.

2 Setup of the Competition

Before explaining the algorithm, we first explain how the ANAC'14 competition was set up. We use the notation $[a,b]$, where a and b are integers, to denote the set of integers z such that $a \leq z \leq b$. We use α_1 to denote our agent and α_2 to denote its opponent. Furthermore, we use $\mathscr{H}_{1\to 2}(t)$ to denote the set of proposals made by α_1 until time t, and $\mathscr{H}_{2\to 1}(t)$ to denote the set of proposals made by α_2 until time t.

2.1 The Protocol

In the competition each agent engaged in several negotiation sessions. In each session two agents were paired to negotiate against each other. Each session would finish as soon as the agents made an agreement, or when the deadline of 180 seconds had passed. The agents had to negotiate according to the alternating offers protocol [10]. One of the two agents begins. That agent may pick one deal x from the agreement space Agr_n (see Sect. 2.2) and propose it to the other. The second agent may then either accept that proposal, in which case the session finishes, or may pick another deal from the agreement space and propose it to the first agent. This then continues: agents alternately take turns, and in each turn the agent whose turn it is may make a new proposal or accept the last proposal made by the other agent. When it is agent α_1's turn agent α_2 cannot do anything, and vice versa. Furthermore, an agent may take as much time as he likes before making the next proposal (or accepting the previous proposal). Therefore, the agent's decision is not only *what* deal to propose (or accept) but also *when* to propose. More precisely: when an agent finds a potential deal to propose it should determine whether it will propose that deal or whether it should continue searching for a better deal.

If the deadline passes without any agreement having been made each agent receives a certain number of points, called its *reservation value*. Otherwise, each agent receives the amount of points equal to its utility value $f_i(x)$ (see Sect. 2.2) for the deal x they agreed upon.

2.2 The Agreement Space

In each session the space of possible deals that can be made by the agents (the *agreement space*) Agr_n is represented by an n-dimensional vector space, where n varies per session and can be as high as 50. For each vector entry there are 10 possible values.

$$Agr_n = [0,9]^n$$

For a vector $x \in Agr_n$ we use the notation x_j to denote its j-th entry.

$$x = (x_1, x_2, ...x_n)$$

Definition 1. A **rectangular subspace** s is a subset of Agr_n such that for each $j \in [1,n]$ there are two integers $a_j, b_j \in [0,9]$ with:

$$x \in s \quad \text{iff} \quad \forall j \in [1,n] : x_j \in [a_j, b_j]$$

Definition 2. A **constraint** is c a pair (s_c, v_c) where s_c is a rectangular subset of Agr_n and v_c is a real number. We say a deal $x \in Agr_n$ **satisfies** a constraint c, iff $x \in s_c$. The **characteristic function** f^c of a constraint c is defined as:

$$f^c(x) = \begin{cases} v_c & \text{if } x \in s_c \\ 0 & \text{if } x \notin s_c \end{cases}$$

For each agent $\alpha_i \in \{\alpha_1, \alpha_2\}$ there is a set of constraints C_i, that determine the agent's utility function f_i according to:

$$f_i(x) = \sum_{c \in C_i} f^c(x)$$

Both sets of constraints however, remain hidden for both agents, so an agent cannot directly calculate its own utility values. Instead, each agent α_i has access to an 'oracle' that, for any given deal $x \in Agr_n$ returns its corresponding utility value $f_i(x)$. The agents cannot know anything about their opponents' utility functions (i.e. agent α_1 can request $f_1(x)$ from its oracle, but not $f_2(x)$). Note that, in principle, an agent can request the value of each deal in the agreement space. However, since the agreement spaces are extremely large (consisting of up to 10^{50} deals) this is obviously infeasible, so the agents can only explore a tiny fraction of the agreement space. In the rest of this paper, whenever we use the term 'utility' without specifying an agent, we mean the utility function f_1 of our agent α_1.

3 Overview of the Algorithm

Let us first give a global overview of the algorithm before we go into more detail on each of its steps. Each turn the algorithm takes the following steps:

1. Calculate the *aspiration value* and *max distance* (Alg. 1, lines 1-2).
2. Decide whether to accept the previous offer made by the opponent (Alg. 1, lines 3-11).
3. Apply a *Global* genetic algorithm to sample the agreement space, and store the 10 proposals with highest utility (Alg. 1, lines 12-13).
4. Apply a *Local* genetic algorithm to sample the agreement space, and store the 10 proposals with highest utility (Alg. 1, lines 14-15).
5. Apply the offer strategy to pick the "best" proposal found by the GAs in the current round or any of the previous rounds (Alg. 1, line 16).

Algorithm 1 chooseAction(t, x, z)

Require: m_1, m_2
1: $m_1 \leftarrow$ calculateAspirationValue(t)
2: $m_2 \leftarrow$ calculateMaxDistance(t)
3: **if** $f_1(x) \geq m_1$ **then**
4: accept(x)
5: **return**
6: **else**
7: **if** $f_1(z) \geq m_1$ **then**
8: propose(z)
9: **return**
10: **end if**
11: **end if**
12: newFound $\leftarrow$ globalGeneticAlgorithm()
13: found $\leftarrow$ found $\cup$ newFound
14: newFound $\leftarrow$ localGeneticAlgorithm(x)
15: found $\leftarrow$ found $\cup$ newFound
16: proposeBest(found, m_1, m_2)

4 Acceptance Strategy

The acceptance strategy of GANGSTER is given in lines 3-11 of Algorithm 1. It depends on a function of time $m_1(t)$ that we call our *aspiration level*. We do not have space here to explain how it is calculated, but the only important thing to know is that it is a decreasing function of time that represents our agent's willingness to concede.

Let $x \in Agr_n$ denote the last offer proposed by the opponent, and let $z \in Agr_n$ denote the offer with highest utility among all deals made by the opponent in the

earlier rounds:

$$\forall z' \in \mathscr{H}_{2\to 1}(t) \ : \ f_1(z) \geq f_1(z')$$

If the utility $f_1(x) \geq m_1$, then our agent immediately accept the proposal made by the opponent. If not, then our agent compares its aspiration level with the highest utility offered by the opponent so far, $f_1(z)$. If $f_1(z) \geq m_1$ then α_1 *reproposes* that deal to the opponent. Note that this means that z is a deal that was earlier rejected by our agent (because at that time its aspiration level was higher), but now that the deadline has come closer it has lowered its standards since the risk of failure has become bigger and is now willing to accept it after all. Unfortunately, the alternating offers protocol does not allow an agent to accept a proposal from an earlier round, so it needs to be proposed again. If, on the other hand $f_1(z) < m_1$ it means that it does not consider z good enough (yet), so it will apply the search strategy (Sec. 5) and offer strategy (Sec. 6) to determine a new proposal to make.

Let us now compare this strategy with existing acceptance strategies. In [1] a study was made of several acceptance strategies. The most common acceptance strategy they identified is named $AC_{prev}(1,0)$. In that strategy the agent α_1 compares the utility of the opponent's offer $f_1(x)$ with the utility $f_1(w)$ of the proposal w that agent α_1 would make if it would not accept x.

Our strategy is a variation of that strategy. However, instead of comparing the utilities of two proposals, our agent compares the utility of the opponent's proposal $f_1(x)$ with its aspiration level $m_1(t)$. The reason for applying this strategy is that α_1 can already determine whether or not to accept the opponent's offer x before it has determined its own proposal w. This has two advantages: firstly, this may save some time because determining the next proposal w can be time consuming. Secondly, it may not always be possible to find a deal w for which the utility is higher than the aspiration level. For example, the agent may only be able to find a deal w with utility $f_1(w) = m_1 - 0.1$. If it would apply $AC_{prev}(1,0)$ it would not accept the opponent's proposal x, even though x yields a utility value higher than our agent's aspiration level. That would be suboptimal, since the aspiration level is by definition the amount of utility that the agent considers high enough to accept.

Another improvement with respect to the $AC_{prev}(1,0)$ strategy is that we introduce the concept of *reproposing* (Alg. 1, lines 6-11). After all, the fact that our agent rejected an earlier proposal from the opponent does not have to mean it would never accept it. The great advantage of reproposing, is that we know that the opponent has already proposed it, and therefore it is very likely that he will accept it. Moreover, the fact that it was already proposed to α_1 means that our agent does not have to search for a new proposal, it already has z readily available in its memory.

5 Search Strategy

We will now explain the Genetic Algorithms (GA) that our agent applies to find good deals to propose. For more background information about Genetic Algorithms

we refer to, for example, [11] and [2]. To apply a Genetic Algorithm one needs to model the elements of the search space as vectors, called *chromosomes*. Luckily, in the ANAC-domain the possible deals are already given as vectors, so we do not have to put any effort in this modeling. The chromosomes are simply the vectors x as defined in Section 2.2. Our GA consists of the following steps:

1. **Initial population:** Randomly pick 120 vectors from Agr_n. This is the initial population.
2. **Selection:** Pick the 10 vectors with highest utility from the population. These are the 'survivors'.
3. **Mutation:** Pick another random vector from Agr_n and add it to the survivors.
4. **Cross-over:** For each pair (v,w) of these 11 survivors, create two new vectors v' and w' (so in total we create 110 new vectors).
5. **New population:** The new population now consists of the 110 new vectors from step 4, plus the 10 survivors from step 2. Go back to step 2, and repeat until convergence, or until we have iterated 10 times.

After a number of iterations the population may contain the same vector more than once. Therefore, when we pick the 10 vectors with highest utility in the selection step, we mean the 10 best *unique* vectors. In other words: we first remove any duplicates from the population and then pick the 10 best vectors. It may happen however that the population has evolved so quickly that no more new unique vectors are created by cross-over. In that case we say it has converged, and the GA is stopped.

5.1 Cross-over

A common way for a GA to apply cross-over, is to cut two vectors both in two halves, and then gluing the first half of one vector to the second half of the other vector and vice versa. We have however opted for a different kind of cross-over, in which random vector-entries are swapped.

Suppose we have two vectors $v,w \in Agr_n$. The cross-over mechanism will output two vectors v' and w' as follows. It first generates a random vector r of dimension n, where each entry r_i has the value 0 with probability 50% or the value 1 with probability 50%. Then, given the vectors v,w and r, the vectors v' and w' are defined according to:

$$\text{if } r_i = 0 \text{ then } v'_i = v_i \text{ and } w'_i = w_i$$

$$\text{if } r_i = 1 \text{ then } v'_i = w_i \text{ and } w'_i = v_i$$

The reason that we have chosen for this type of cross-over, is that (as far as the participants can know) there is no relation between the vector entries in the domain. A constraint may for example involve the 3rd and the 7th entry, and there is no reason to assume that consecutive entries are stronger related than non-consecutive entries. This is reflected in our cross-over mechanism by the fact that it is symmetric

under any permutation of the entries, whereas the regular cross-over mechanism has a strong bias towards the survival of consecutive sequences of values.

5.2 Global Search vs. Local Search

As we can see in Algorithm 1, in each turn our agent applies two GAs.The first is called the *global* GA, and the second one we call the *local* GA. The difference is that in the global GA it picks vectors randomly from anywhere in the agreement space, while in the local GA it only picks vectors that are close to the last proposal made by the opponent. Specifically: there is a decreasing time-dependent function $m_2(t)$ and our agent only picks vectors for which the Manhattan distance to the last proposal made by the opponent is smaller than $m_2(t)$. The idea is that on one hand α_1 wants to maximize its own utility f_1 and therefore searches for good deals anywhere in the space, but on the other hand also needs to find proposals that are good for the opponent so it applies a local GA to find good deals that are similar to the proposals made by the opponent.

6 Offer Strategy

In lines 12-15 of Alg. 1 we see that the vectors returned by the GAs are added to a set of potential proposals. After that, it is the task of the offer strategy to determine which of those potential proposals should be proposed to the opponent (Alg. 1, line 16). In this section we use the notation $d(x,y)$ to denote the Manhattan distance between vectors x and y:

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

For each vector x in the set of potential proposals the agent determines three properties, called *utility*, *distance*, and *diversity*, that will determine which deal is the best to propose. The first of these properties, utility, is the most obvious: the higher our agent's utility $f_1(x)$, the better the deal.

Definition 3. The **distance** $dist_t(x)$ of a vector $x \in Agr_n$ at time t, is the lowest Manhattan distance between x and any proposal previously made by the opponent:

$$dist_t(x) = \min\{d(x,y) \mid y \in \mathscr{H}_{2\to 1}(t)\}$$

The idea is that, since our agent cannot know $f_2(x)$ it uses $dist_t(x)$ as a measure for the opponent's utility instead. If $dist_t(x)$ is low, then there is a high probability that $f_2(x)$ is high. Therefore, our strategy prefers to propose deals with low distance.

Definition 4. The **diversity** $div_t(x)$ of a potential proposal $x \in Agr_n$ at time t is the shortest Manhattan distance between x and any of the proposals previously made by

our agent:

$$div_t(x) = \min\{d(x,y) \mid y \in \mathscr{H}_{1\to 2}(t)\}$$

Our offer strategy prefers proposing deals with high diversity because this has two advantages:

- If α_2 rejected proposal v, and the vector w is close to v (i.e. $div_t(w)$ is low), then it is likely that α_2 will also reject w. So our agent should avoid proposing deals that are similar to earlier rejected deals.
- By proposing more diverse offers, α_1 gives the opponent more information about its utility function f_1, making it more easy for α_2 to find proposals that are profitable to α_1.

Let us clarify this a bit more. Imagine that α_1's utility function has one very high peak. That is: there is a small area inside Agr_n where f_1 is very high. Then α_1 would be inclined to only make proposals from that area. However, if the opponent's utility f_2 is very low in that same area this will be an unsuccessful strategy. Now, suppose that there are a number of other areas of Agr_n where f_1 is less high, but still high enough to be proposed. Then by giving priority to deals with high diversity, we make sure that α_1 also makes proposals around those alternative peaks, hence increasing the probability that for some of these proposals the opponent utility f_2 will also be high. Secondly, in this way α_1 reveals to α_2 the locations of the alternative peaks, which also makes it easier for α_2 to find deals that are profitable to α_1.

We will now explain how utility, distance and diversity are used to determine which proposal to make. This strategy depends on two time-dependent functions: the aspiration level $m_1(t)$ and the maximum distance: $m_2(t)$. Let X denote the set of potential proposals found by the local and global GAs. Then we define the subset $Y \subseteq X$ as:

$$Y = \{y \in X \mid f_1(y) \geq m_1(t) \ \wedge \ dist_t(y) \leq \ m_2(t)\}$$

The deal that α_1 will propose next is then defined as the element $y^* \in Y$ with highest diversity:

$$\forall y \in Y : div_t(y^*) \geq div_t(y)$$

We see here that m_1 acts as a minimum amount of utility α_1 requires for itself, while m_2 acts as a minimum amount of utility that α_1 considers necessary to offer to α_2 (recall that low distance represents high opponent utility). We do not have space to explain how m_1 and m_2 are calculated, but the important thing to know is that both are decreasing functions of time. This means that as time passes, α_1 requires less and less utility for itself, while it forces itself to offer more and more utility to α_2. After all, the closer it gets to the deadline, the more desperate the agent will get to make a proposal that gets accepted by the opponent. If there is more than one potential proposal for which the utility is high enough and the distance is low enough then α_1 picks the one with highest diversity.

7 Motivation for Using Manhattan Distance

Let us now explain why we have chosen to use the Manhattan distance in our definitions. The idea is that we use distance to measure the difference between the utility values of two deals. The closer two deals x and y are, the more likely that $f_i(x)$ is close to $f_i(y)$. Indeed, the utility of a deal is determined by the constraints that it satisfies and if two deals are close to each other then they are likely to satisfy the same constraints. The question however, is which distance measure best reflects the similarity in utility values.

Let c be a constraint that is satisfied by x, that is: $x \in s_c$. Now for each entry x_j the constraint defines two integers a_j and b_j which are unknown. This means that if we increase or decrease x_j by 1, there is a probability that x will no longer satisfy the constraint as x_j may 'drop out' of the interval $[a_j, b_j]$. If x_j is in the interval $[a_j, b_j]$ then we denote the probability that $x_j + 1$ or $x_j - 1$ is not, by p:

$$P(x_j \pm 1 \notin [a_j, b_j] \mid x_j \in [a_j, b_j]) = p$$

Since we have no reason to assume that any entry $j \in [1, n]$ is different from any other entry, we can assume that p is equal for each entry j. Then the probability of dropping out of the constraint after making k steps is p^k, *independent of the directions* of these steps. Specifically: it does not matter whether we take two steps in the $j = 1$ direction or two steps in the $j = 2$ direction, or one step in the $j = 1$ direction and one step in the $j = 2$ direction. In other words, the probability that y satisfies c equals $p^{d(x,y)}$ where d is the Manhattan distance.

Note that this is a direct consequence of the fact that constraints are defined by *rectangular* subspaces. If they had been defined by spherical subspaces for example, then the same reasoning would have lead us to use Euclidean distance.

8 Conclusions

The ANAC'14 competition had two categories: the individual category, in which the agents where ranked according to the individual utility they obtained, and the social category in which agents were ranked by the social utility, which is the sum of the agent's own utility and its opponent's utility. Gangster ranked 3rd place int the individual category, and 2nd place in the social category, among more than 20 participants. We conclude that our agent is a good negotiator and that Genetic Algorithms are a good search technique for the given domain.

However, although the negotiation domains were very large, we think that this may have had only very little influence on the success of the negotiators. The reason for this belief is that when testing our GA on the largest test domain it was often able to find deals with $f_1(x) > 0.95$ in less than 70 ms. This is very short in comparison to the total amount of 180 seconds available. Therefore, we think the success of a participant depended more on its bargaining strategy than on its search algorithm.

This is a pity, because the search was supposed to be the distinguishing property of this year's competition with respect to other years. We would therefore be very interested to know what the results would have been if the deadlines had been much shorter.

Acknowledgments

This work was supported by the Agreement Technologies CONSOLIDER project, contract CSD2007-0022 and INGENIO 2010 and CHIST-ERA project ACE and EU project 318770 PRAISE.

References

1. Baarslag, T., Hindriks, K., Jonker, C.: Acceptance conditions in automated negotiation. In: Complex Automated Negotiations: Theories, Models, and Software Competitions, pp. 95–111. Springer Berlin Heidelberg (2013)
2. Falkenauer, E.: Genetic Algorithms and Grouping Problems. John Wiley & Sons, Inc., New York, NY, USA (1998)
3. Faratin, P., Sierra, C., Jennings, N.R.: Negotiation decision functions for autonomous agents. Robotics and Autonomous Systems **24**(3-4), 159 – 182 (1998). DOI 10.1016/S0921-8890(98)00029-3. Multi-Agent Rationality
4. Faratin, P., Sierra, C., Jennings, N.R.: Using similarity criteria to make negotiation trade-offs. In: International Conference on Multi-Agent Systems, ICMAS'00, pp. 119–126 (2000)
5. de Jonge, D., Sierra, C.: Automated negotiation for package delivery. In: Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on, pp. 83–88 (2012). DOI 10.1109/SASOW.2012.23
6. de Jonge, D., Sierra, C.: NB^3: a multilateral negotiation algorithm for large non-linear agreement spaces with limited time. Journal of Autonomous Agents and Multi-Agent Systems p. in press (2015)
7. Marsa-Maestre, I., Lopez-Carmona, M.A., Velasco, J.R., de la Hoz, E.: Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '09, pp. 1057–1064. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2009). URL http://dl.acm.org/citation.cfm?id=1558109.1558160
8. Marsa-Maestre, I., Lopez-Carmona, M.A., Velasco, J.R., Ito, T., Klein, M., Fujita, K.: Balancing utility and deal probability for auction-based negotiations in highly nonlinear utility spaces. In: Proceedings of the 21st International Jont Conference on Artifical Intelligence, IJCAI'09, pp. 214–219. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2009)
9. Nash, J.: The bargaining problem. "Econometrica" **"18"**, 155–162 (1950)
10. Rosenschein, J.S., Zlotkin, G.: Rules of Encounter. The MIT Press, Cambridge, USA (1994)
11. Schmitt, L.M.: Theory of genetic algorithms. Theoretical Computer Science **259**(12), 1 – 61 (2001). DOI http://dx.doi.org/10.1016/S0304-3975(00)00406-0