Arguing Using Opponent Models

Nir Oren[1] and Timothy J. Norman[2]

[1] Dept. of Computer Science
King's College London
Strand, London
WC2R 2LS, United Kingdom
nir.oren@kcl.ac.uk
[2] Dept. of Computing Science
University of Aberdeen
Aberdeen
AB24 3UE
Scotland
t.j.norman@abdn.ac.uk

Abstract. While researchers have looked at many aspects of argumentation, an area often neglected is that of argumentation strategies. That is, given multiple possible arguments that an agent can put forth, which should be selected in what circumstances. In this paper we propose a heuristic that implements one such strategy. The heuristic is built around opponent modelling, and operates by selecting the line of argument that yields maximal utility, based on the opponent's expected response, as computed by the opponent model. An opponent model may be recursive, with the opponent modelling of the agent captured by the original agent's opponent model. Computing the utility for each possible line of argument is thus done using a variant of M* search, which in itself is an enhancement of min-max search. After describing the M* algorithm we show how it may be adapted to the argumentation domain, and then study what enhancements are possible for more specific types of dialogue. Finally, we discuss how this heuristic may be extended in future work, and its relevance to argumentation theory in general.

1 Introduction

Argumentation has emerged as a powerful reasoning mechanism in many domains. Applications tend to revolve around either the logical form of argument, identifying when an argument is, in some sense, acceptable. Some extension of Dung's seminal argument framework [8] is typically used in such applications, and domains have included negotiation [3] and normative conflict detection [12]. Another use of argumentation makes use of the dialogue level, and focuses on the interaction between different parties as the dialogue progresses. It is important to note that these two approaches are not mutually exclusive, but rather complementary. In describing argumentation systems, Prakken [18] identified four layers with which an argument framework must concern itself. These are the

logical, dialectic, procedural, and heuristic layers. The first strand of research primarily concerns itself with the logical and dialectic layers, while the second focuses on the procedural level. As Prakken notes, little work has dealt with the heuristic layer, which, among other things, deals with how agents should decide which argument to advance at different points in the dialogue.

In this paper, we focus on the heuristic layer, examining argument strategies. In other words, we attempt to determine what argument an agent should advance in order to achieve a certain goal. Previous work on the topic includes a utility based approach with one step lookahead [14], while another tack has focused on game theory and mechanism design [19, 20] to ensure that dialogues are strategy free. Here, we look at how an agent may decide what utterance to advance by making use of opponent modelling. That is, given that agent α has a belief about agent β's knowledge and goals, we determine what argument the agent should advance to achieve its aims. Informally, we examine possible utterances from each position in the dialogue, creating a tree of possible dialogues. We then make use of techniques adapted from computer game playing, namely opponent model search and the M* algorithm [5], which is itself an extension of standard min-max search [22].

As an example of such a search, consider two agents (α and β) participating in a dialogue regarding whether to invade a small country. Agent α would like to persuade β that this is a prudent course of action. One possible line of argument is that the country has weapons of mass destruction, and that invading it will prevent the use of these weapons. If α knows that β cannot determine whether the country has such weapons, and that β may not ask α where it obtained its information from, then α will use this line of argument in the dialogue. However, if it knows that β knows that the country has no WMDs, it will not attempt to use this line of argument. Thus, α can make use of its model of β's knowledge in deciding what arguments to advance as part of a dialogue. The use of opponent modelling as an argument strategy has clear applications to many different scenarios, including persuasion type dialogues, and the domain of automated negotiation.

This paper's main contribution is a description of how a dialogue participant may decide what arguments to advance given that it has a model of the other dialogue party. We also show how the M* algorithm may be adapted for the argumentation domain, and provide some domain dependant enhancements to the search process, which results in a pruning of the dialogue tree.

After describing the M* algorithm in more detail in the next section, we show how M* can be adapted for the argumentation domain. Our initial effort, described in Section 3.1 deals with very general situations, and we show how our approach can be specialised for more specific types of dialogues and situations in Sections 3.2 and 3.3. Finally, we discuss a number of possible enhancements to our approach, and describe additional related work.

2 Background

In this section, we examine work in the field of opponent modelling. When participating in a competitive event in which strategy may affect the outcome, human players typically adjust their strategy in response to the opponent(s) they face. On the other hand, artificial agents typically construct plans in which they assume that their opponent will play in the way most damaging to their goals, resulting in algorithms such as min-max search, and many of the results of game theory. Such approaches typically also assume that the opponent will use the same strategy as the player.

By making use of opponent modelling, it is possible to represent opponents with different goals and strategies to the agent doing the modelling. Such an approach raises a number of interesting possibilities. First, it allows for situations such as swindles and traps. The former occurs when a player may, due to an opponent's weakness, play a non-optimal move and still win, while the latter allows a player to play a weak move under the assumption that the opponent will view it as a strong move. Second, by explicitly modelling the opponent's goals, it is easier to deal with non-zero sum games. Other advantages gained by making use of strategies built around opponent modelling are described in [5].

The M* algorithm is similar to the min-max search algorithm, but makes use of an explicit opponent model to more accurately calculate the utility of a position. An opponent model (described as a player in [5]) is a pair $\langle f, O \rangle$ where f is an evaluation function, and O is an opponent model. The latter may also take on the special value of *NIL*, representing the case where no opponent model exists.

In this paper, we make use of a slightly modified form of this algorithm, but before describing this modified algorithm, we examine the original, which is shown in Algorithm 1.

Informally, given a board position, a search depth, and an opponent model (consisting of the opponent's evaluation function and another opponent model), the algorithm returns the best move (and its value) by recursively computing what move the best move for the opponent would be for the given opponent model (line 11), based on what the opponent thinks the player's best move would be (line 10), and so on, until the maximal depth of search is reached. At this point, the utility of the current move can be computed (lines 2 and 8). The *MoveGen* function in line 5 is dependant on the rules of the game, and generates all possible moves from the current board position *pos*. It should be noted that it is easy to represent standard min-max search using the M* algorithm by using an opponent model of the form $(f, (-f, (f, \ldots)))$ to the depth of the search. It should also be noted that the algorithm assumes that the depth to which the opponent is modelled is greater than (or, more usually equal to) the depth to which the search takes place.

Algorithm 1 $M^*(pos, depth, f_{pl}, oppModel)$

Require: A board position pos
Require: A search depth $depth$
Require: A position evaluation function f_{pl}
Require: An opponent model $oppModel = \{f^o_{pl}, oppModel^o\}$

if depth=0 **then**
  return $(NIL, f_{pl}(pos))$
else
  $maxUtil = -\infty$
  $PossibleMoves = MoveGen(pos)$
  for all $move \in PossibleMoves$ **do**
    if depth=1 **then**
      $playUtil = f_{pl}(move)$
    else
      $\langle oppMove, oppUtil \rangle = M^*(move, depth - 1, f^o_{pl}, oppModel^o)$
      $\langle playMove, playUtil \rangle = M^*(oppMove, depth - 2, \langle f_{pl}, oppModel \rangle)$
    end if
    if $playUtil > maxUtil$ **then**
      $maxUtil = playUtil$
      $maxMove = move$
    end if
  end for
end if
return $(maxMove, maxUtil)$

3 Approach

Argument strategies are employed by agents taking part in a dialogue. Arguments are thus added over time, and the goal of an agent's strategy involves deciding which utterance to make. For dialogue to take place, participants must agree on the rules governing the dialogue. These include not only the rules of the dialogue game itself [10], such as whether turn taking exists, what utterances may be made at which point, and the like, but also the rules governing the interactions between arguments; for example stating that *modus ponens* is valid.

3.1 Opponent Modelling for General Dialogues

We begin by assuming a very general type of dialogue game similar in spirit to the one proposed in [17]. This game is represented by agents advancing arguments. Arguments may take the form of more specific utterances such as "assert x", but such utterances are captured in abstract form, by being represented as arguments themselves. Thus, for example, an agent may make an argument a, and if another agent questions this argument (for example, by asking "why a?"), this could be viewed as an argument b which attacks a. A response could then be an argument c, attacking b, etc. We denote this set of abstract arguments $Args$.

A dialogue represents the utterances made by an agent. We define it very simply as follows:

Definition 1. ***(Dialogue)*** *A dialogue is an ordered set of arguments*

$$Dialogue = \{a, b, \ldots\}$$

such that $Dialogue \subseteq Args$.

For convenience, and without loss of generality, we assume that only two dialogue participants exist, and that they take turns when making utterances. We do not therefore need to associate a dialogue participant with an utterance in our representation of a dialogue.

In order to abstract the rules governing a dialogue, we assume the existence of a generator function allowing us to compute the set of legal possible moves in a dialogue. This function is equivalent to the *MoveGen* function in Algorithm 1.

Definition 2. ***(Move Generation)*** *The function*

$$legalMoves : 2^{Dialogue} \rightarrow 2^{Args}$$

takes in a dialogue and returns the set of arguments that may be uttered by an agent at that point in the dialogue.

Note that only a single argument may be advanced by an agent during its turn (as this argument may actually encapsulate other arguments). Also, note that the legal moves function does not explicitly depend on the player making an utterance (though such information may be implicitly found by examining the dialogue's current length). The $legalMoves$ function identifies *legal moves* rather than *possible moves* for a dialogue. That is, moves that are sanctioned by the rules of the game, rather than those moves that an agent may actually make based on factors such as its knowledge and goals.

At this point, we must define the structure of dialogue participants (also referred to as agents). An agent participating in a dialogue has a knowledge base identifying the arguments it is aware of, some goals it is trying to achieve, and an opponent model.

Definition 3. ***(Agent)*** *An agent is a tuple*

$$\langle KB, Goals, Opp \rangle$$

where $KB \subseteq Args$ *is a knowledge base containing those arguments known by an agent,* $Goals$ *is the agent's goal function, and is described in Definition 4, while* Opp *is an opponent model as detailed in Definition 5.*

An agent participates in a dialogue to achieve certain goals. Different dialogues may meet these goals to a greater or lesser extent. An agent may thus assign a utility to a dialogue, and this is modelled by the $Goals$ function. This utility may depend on many factors, including the arguments introduced within the dialogue, the agent introducing these arguments, the arguments deemed admissible by some argumentation semantics at some stage of the dialogue, and the like.

Definition 4. ***(Goals)*** *A Goals function takes in a dialogue, and returns its utility.*

$$Goals : 2^{Dialogue} \rightarrow \mathbb{R}$$

An opponent model is meant to represent an agent, and thus looks very similar to Definition 3. However, we must also handle the situation where an agent has no recursive model of its opponent. We start by representing this situation, and then recursively defining more complex, nested opponent models.

Definition 5. ***(Opponent Model)*** *An opponent model is incrementally defined as follows:*

- $\langle KB, Goals, \emptyset \rangle$ *is an opponent model*
- $\langle KB, Goals, Opp \rangle$*, where Opp is an opponent model.*

Here, $KB \subseteq Args$ *and* $Goals : 2^{Dialogue} \rightarrow \mathbb{R}$ *are a knowledge base and goal function respectively.*

Given an agent with some knowledge base, goal function and opponent model, together with a $legalMoves$ function representing the dialogue game's rules, an agent may decide what utterance to make (i.e. what argument to advance) by following the slightly modified version of M*, called M^*_{gd} shown in Algorithm 2.

Apart from the different specification of an agent, line 5 ensures that an agent not only uses moves generated by the $legalMoves$ function, but filters these to ensure that it only uses moves that it believes are possible (according to its knowledge base). Also, note that on line 10, the opponent model is treated as an agent when passed as a parameter to the algorithm.

The M^*_{gd} algorithm is very general, in the sense that it does not make use of any of the properties of argument that would differentiate it from other types of search. In the remainder of this section, we take a closer look at the $legalMoves$ and $Goals$ functions, showing how they may be specialised to represent more specific dialogues, and how these specialisations may aid in pruning the possible dialogue tree.

3.2 Pruning by Legal Moves

One of the focuses of argumentation research examines how groups of arguments interact. Typically, this interaction is built around the notion of attack, and more controversially support, between arguments. The arguments advanced by (rational) agents in the course of a dialogue are aimed at achieving their goals, either by building up support for, or attacking other arguments [2].

Various semantics for argument frameworks have been proposed [8, 13]. These semantics, when given a set of arguments and the interactions between them, identify which sets of arguments may be viewed as, in some sense, consistent with each other. In most cases, it makes little sense for an agent to introduce an argument which would not be deemed consistent, and so we begin by showing how this notion may be added to the agent's reasoning process.

Filtering the set of arguments that may be advanced is the task of the $legalMoves$ function. We must, therefore, specialise this function as follows:

Algorithm 2 $M^*_{gd}(dia, depth, agent, legalMoves)$

Require: A dialogue dia
Require: A search depth $depth$
Require: An agent $agent = \langle KB, Goals, oppModel \rangle$
Require: A move generation function $legalMoves$
1: **if** depth=0 **then**
2: **return** $(NIL, Goals(dia))$
3: **else**
4: $maxUtil = -\infty$
5: $PossibleMoves = legalMoves(dia) \cap KB$
6: **for all** $move \in PossibleMoves$ **do**
7: **if** depth=1 **then**
8: $playUtil = Goals(dia \cup \{move\})$
9: **else**
10: $\langle oppMove, oppUtil \rangle = M^*_{gd}(dia \cup \{move\}, depth-1, oppModel, legalMoves)$
11: $\langle playMove, playUtil \rangle = M^*_{gd}(dia \cup \{move, oppMove\}, depth - 2, agent, legalMoves)$
12: **end if**
13: **if** $playUtil > maxUtil$ **then**
14: $maxUtil = playUtil$
15: $maxMove = move$
16: **end if**
17: **end for**
18: **end if**
19: **return** $(maxMove, maxUtil)$

– Introduce the notion of an argument system; rather than having the arguments $Args$ in isolation, we define an argument system consisting of $Args$ and a set of relations between subsets (or elements) of $Args$. For example, a Dung-style argument system is a tuple $\langle Args, Attacks \rangle$ where $Attacks \subseteq Args \times Args$.
– Add a semantics under which the concept of "additional information" can be judged. In the case of a Dung-style argument system, these may be preferred, grounded, stable, or some other semantics.

Definition 6. ***(The Legal Moves Function for Argument Systems)*** *The $legalMoves$ function for an argument system AS is a function*

$$legalMoves : AS \times Semantics \times Dialogue \rightarrow 2^{Args}$$

This function takes in an argument system AS, together with an associated semantics $Semantics$, and a dialogue, and returns a set of possible arguments.

For example, in a Dung-style argument system, the $legalMoves$ function could be defined as follows[3]:

[3] Note that this definition means that only an argument relevant (in the sense of Definition 7) to some argument may be introduced in this dialogue game. However,

$$legalMoves(AS, DSemantics, Dialogue) = \{\textbf{pass}\} \cup \{a | a \in Args \text{ and } DSemantics(\langle Dialogue \cup \{a\}, Attacks\rangle) \neq DSemantics(\langle Dialogue, Attacks\rangle)\}$$

Where **pass** is a move meaning no utterance is made[4], $AS = \langle Args, Attacks\rangle$ and $DSemantics : 2^{AS} \rightarrow 2^{Args}$ captures the appropriate Dung-style preferred, grounded or stable semantics.

This $legalMoves$ function requires that a legal move be one that changes the conclusions that can be drawn from the dialogue if it were to terminate at this point. In other words, a legal move is one that changes the set of arguments that are contained in the grounded, preferred or whatever extension that is required by the semantics. As a consequence of this, the dialogue tree of moves excludes those that do not alter the accepted set of arguments, and so this function ensures that a dialogue will terminate (assuming a finite set of arguments).

It is important to note that the $legalMoves$ function captures both the rules of the dialogue game's locutions, and the contents of these locutions, mixing the dialogue game's syntax with its semantics.

3.3 Filtering Using Relevance

Often, a dialogue participant's goals revolve around having the other parties accept, or reject a single argument. In such a situation, arguments that are not directly relevant to this single argument may be ignored when searching the possible move tree. This notion is related, but different to the idea presented in the previous section. The $legalMoves$ function captures all legal moves according to a dialogue. A relevant move is a legal move that also affects the agent's goals. We define one argument as relevant to another if it can affect its status in some way.

Definition 7. ***(Relevance)*** *Given*

- *Two arguments* a, b
- *The set of all arguments* $Args$
- *A semantics for argument, which is able to determine the status of an argument given a set of arguments using the function* $Semantics(c, X)$ *where* $c \in Args$ *and* $X \subseteq Args$

We say that a *is relevant to* b *(in the context of some argument system) if* $\exists X \subseteq Args$ *such that* $a \notin X$ *and* $Semantics(b, X) \neq Semantics(b, X \cup \{a\})$. *We may write* $relevant(a, b)$ *to indicate that* a *is relevant to* b.

---

a relevance-aware agent (Definition 8) considers arguments relevant to its goals, maintaining the distinction between legal moves and moves relevant to it. Note also that this $legalMoves$ function allows an agent to introduce arguments that attack its own arguments, which may not be allowed in some dialogues.

[4] A dialogue game typically ends after all agents consecutively pass.

*In dialogues where an agent may pass to end the game, a **pass** move is also considered relevant. In such games, relevant*(**pass**, A) *for any* $A \in Args$ *also holds.*

While the $legalMoves$ function *may* make use of the underlying argument system's semantics, the determination of relevance of an argument *must* employ the system's semantics.

Within a Dung-style argument framework, an argument is relevant if there is a path within the argument graph between it and some other argument. This makes sense intuitively, as the argument has the potential to affect the other argument's status (by directly, or indirectly attacking, or reinstating it).

Support is another way of having one argument affect another. In many dialogue types, an argument's premises have to be introduced before the argument itself may be used. A number of semantics have been proposed to deal with support [13, 1, 9], and any are appropriate for the purposes of this algorithm[5]

As mentioned above, the notion of relevance requires that an agent be able to specifically identify its goals. We extend the notion of an agent to capture this as follows:

Definition 8. ***(Relevance-Aware Agent)*** *A relevance-aware agent is a tuple*

$$\langle KB, Goals, GoalArguments, Opp \rangle$$

where $KB \subseteq Args$, $Goals : 2^{Dialogue} \rightarrow \mathbb{R}$ *and Opp is an opponent model as in Definition 3, and* $GoalArguments \subseteq Args$.

As discussed below, a relevance-aware agent utilises Algorithm 3 to consider only arguments which are relevant to its $GoalArguments$ in its search. If the $GoalArguments$ set is small (in comparison to the arguments returned by the $legalMoves$ function), this may lead to a considerable pruning of the search tree. If we assume a single argument within $GoalArguments$, then the notion of relevance presented here is equivalent to R1 relevance in the work of Parsons et al. [16].

As the number of $GoalArguments$ increase, the usefulness of the relevance optimization decreases, as more arguments typically become relevant. In order for the relevance based approach to function, the $Goals$ function must also be constrained.

Since the utility of a dialogue is only evaluated once the tree is expanded to the maximum search depth, we cannot simply alter the agent's $Goal$ function to

[5] It should be noted that the dialogues resulting from a framework such as [13] are very different to those obtained from a Dung style framework. In the latter, the dialogue would begin with the goal arguments being introduced, and attacks on those arguments (and attacks on those attacks) introduced in subsequent moves. However, the former approach requires that all arguments be supported, meaning that the dialogue would progress by first introducing arguments as premises, and then build on these premises with additional arguments, until the goal arguments are introduced. Attacks on introduced arguments may still occur at any time.

include the notion of relevance. Instead, an agent must filter the set of possible moves to be evaluated based on its goals. To do this, a relevance-aware agent invokes Algorithm 3 using

$$M^*_{rel}(dia, depth, \langle KB, Goals, Opp\rangle, GoalArguments, legalMoves)$$

Where $dia, depth$ and $legalMoves$ are dependent on the dialogue and the agent's capabilities.

It is clear that the relevant moves form a subset of the legal moves for a dialogue. Thus, for all $g \in GoalArguments$

$$\bigcup_g \{a | a \in Args \text{ and } relevant(a, g)\} \subseteq legalMoves(AS, Semantics, Dialogue)$$

If this subset relation is strict, the notion of relevance is useful in pruning the search tree. However, even in this case, we still filter by using the $legalMoves$ function (line 5) before filtering by relevance, as we assume that this operation is computationally cheaper than relevance filtering.

Lines 6–10 filter out irrelevant moves. It should be noted that the relevance-aware agent's $GoalArguments$ are used at all depths of this algorithm, as any move that does not affect these goals can be ignored by the agent. This follows from the idea that if the other party makes (what are considered) irrelevant arguments, these have no effects on the arguments introduced by the agent.

As a simple example, consider the relevance-aware agent

$$\langle \{a, c\}, \{a\}, \{a\}, \langle \{b\}, \{\neg a\}, \langle \{d\}, \{a\}, \{\}\rangle\rangle\rangle$$

Here, $\neg a$ represents a goal that a is not deemed acceptable according to the game's semantics. We may assume that only a single argument may be introduced by an agent during its turn (or that it may pass), and assume Dung-style argument system with grounded semantics.

$$AS = (\{a, b, c, d\}, \{(b, a), (d, b)\})$$

For simplicity, we represent goals as single arguments rather than dialogues; we assume that all dialogues containing the goal argument are worth 10 utility, less one utility for every argument the agent advances.

Then at the first level of argument, arguments a, c, and passing are all legal moves according to the agent's knowledge base and semantics. However, a and pass are the only relevant moves, and the agent must then determine what the opponent will play in response to a. According to its opponent model, the opponent could play b. However, argument c is not relevant, and the agent will thus not consider whether the opponent will play it or not. The algorithm shows that if the opponent plays b, it believes that the agent will play d (even though it does not actually know d). Since playing a move will cost the opponent utility, the agent believes that the opponent will pass.

Algorithm 3 $M^*_{rel}(dia, depth, agent, goalArguments, legalMoves)$

Require: A dialogue dia
Require: A search depth $depth$
Require: An agent $agent = \langle KB, Goals, oppModel \rangle$
Require: A set of arguments $goalArguments$
Require: A move generation function $legalMoves$
if $depth = 0$ **then**
  return $(NIL, Goals(dia))$
else
  $maxUtil = -\infty$
  $PossibleMoves = legalMoves(dia) \cap KB$
  for all $move \in PossibleMoves$ **do**
    if $\nexists a \in goalArguments$ such that $relevant(move, a)$ **then**
      $PossibleMoves = PossibleMoves \backslash move$
    end if
  end for
  for all $move \in PossibleMoves$ **do**
    if $depth = 1$ **then**
      $playUtil = Goals(dia \cup \{move\})$
    else
      $\langle oppMove, oppUtil \rangle = M^*_{rel}(dia \cup \{move\}, depth - 1,$
        $oppModel, goalArguments, legalMoves)$
      $\langle playMove, playUtil \rangle = M^*_{rel}(dia \cup \{move, oppMove\}, depth - 2,$
        $agent, goalArguments, legalMoves)$
    end if
    if $playUtil > maxUtil$ **then**
      $maxUtil = playUtil$
      $maxMove = move$
    end if
  end for
end if
return $(maxMove, maxUtil)$

If instead, the agent passes, it will gain no utility, while the opponent would also pass, and would gain 10 utility. Thus, the agent will utter a, and if its opponent model is accurate, will win the game when the opponent passes.

Given perfect information, the agent should not have won this game. Furthermore, the agent gains 10 utility, while the opponent gains 0 utility.

4 Discussion

This paper is built around the idea of an agent being able to assign utility to a dialogue. While the assignment of positive utility to a dialogue represents the agent meeting some of its goals, negative utility can arise from a number of dialogue-dependent situations. These obviously include the agent not meeting its goals, and, more interestingly, may occur when some other argument is introduced (c.f. [15]), or when the agent actually makes use of an argument (for

example, as a premise to one of its own arguments). Work such as [14] examines some of these utility assignment approaches in more detail, but performs only one step lookahead when selecting an utterance.

As discussed in Section 2, it is possible to model a min-max opponent by making use of an opponent model of the form $(f, (-f, (f, \ldots)))$ to the depth of the search. Constructing a min-max opponent model for the argumentation domain requires incorporating a knowledge base into the opponent model. Two natural choices for an opponent's knowledge base arise. First, it could be identical to the original agent's knowledge base. Second, it could contain all arguments in the system. Since min-max is inherently pessimistic, the latter approach makes sense for the opponent's model, while the former knowledge base would represent the opponent's model of the agent. It has been shown that an agent making use of the M^* strategy will perform no worse than an agent utilising min-max search [5], and this result can be trivially mapped to our extensions of the strategy.

The worst case computational complexity of the M^* algorithm, and thus our heuristic is bounded by $(b+1)^{d-1}$ where b represents the branching factor, and d the search depth. Clearly, computational techniques that prune the tree are important if the algorithm is to be used in real world situations. The techniques described in Section 3.2 and 3.3 reduce the branching factor, and perform pruning of the dialogue tree respectively. Furthermore, depending on the form of the utility function, an adaptation of $\alpha\beta$-pruning may be applied to M^*[5]. This adaptation is particularly applicable to persuasion dialogues, where an agent meeting its goals usually means that its opponent fails to meet their goals. Other techniques, such as transposition tables can also yield significant computational savings, even when the utility function is relatively complex.

Moore [11] suggested that any strategy for argument must meet three criteria, namely to maintain the focus of the dispute, build its point of view, or attack the opponent's one, and select an argument that fulfils the previous two objectives. The inclusion of relevance, and the filtering of the legal moves function allows an agent to include these criteria within its reasoning process. Clearly, any utterance made as a result of a reasoning process meeting Moore's criteria will itself meet the criteria.

Additional work dealing with argument strategy includes the work of Riveret [20, 21], who, like us, builds a tree of dialogues, and computes game theoretic equilibria to determine which move an agent should make. However, he assumes that the agent's knowledge and goals are perfectly known, an assumption which does not hold in many situations. Riveret's work is more general than ours in one sense, namely in that the dialogues they investigate do not assume that agents make a single utterance during their turn, or indeed, take turns in an orderly manner. However, our approach can easily be extended to include this generalisation. Rahwan [19] has also examined the notion of strategy within argument, but focused on showing how dialogues may be designed so as to make them strategy-proof.

We have begun implementing and evaluating the performance of agents making use of opponent modelling as part of their argument strategy. Our initial re-

sults indicate that this approach outperforms techniques such as min-max search, but still need to investigate issues such as the effects of errors in the opponent model on the quality of the strategy, and the tradeoffs between search time and search quality when increasing the search depth.

It is clear that the introduction of an argument by an opponent may cause an update in our knowledge base or opponent model (for example, if an argument a supports an argument b, which in turn supports argument c and we believe that our opponent's goal is argument c, the introduction of a should cause us to believe that our opponent knows b). Our algorithms cater for such updates implicitly; the agent's opponent model at the appropriate depth of the expanded dialogue tree *should* include the fact that the opponent knows b. In the future, we intend to provide more constraints on the form of the agent and its opponent model, capturing notions such as knowledge/belief revision.

Another area of future work involves extending our algorithms to make use of probabilistic opponent modelling [7]. In this paper, we assumed that an agent had a single model of its opponent. However, probabilistic opponent modelling would allow an agent to reason with imperfect information about its opponent, a situation that commonly arises in real world situations. The addition of probabilistic opponent modelling opens up many exciting avenues for research. For example, an agent may have two strategies open to it; one that will yield it a high utility, but assumes that there is a high chance that its opponent does not know a key fact, and the other yielding less utility, but being much safer. In such a situation, the agent must take into account the *risk* of making certain utterances, and we intend to investigate how such considerations can form part of an agent's argument strategy.

Finally, we have assumed throughout the paper that an agent has some opponent model, with an arbitrary evaluation function and knowledge base. However, we have not examined how an agent may learn an opponent model during the course of a dialogue, or over repeated interactions with an opponent. Some work on this topic exists in the context of game playing [6], and we intend to apply it to the argumentation domain.

5 Conclusions

In this paper, we showed how an agent may decide on which utterances to advance in the course of an argument by making use of an opponent model. By making use of an opponent model, the agent can discover lines of argument that may take advantage of its opponent's beliefs and goals. Such lines of argument would not be discovered by a more naïve strategy, such as min-max, which would assume that the opponent's goals are diametrically opposed to its own.

We also showed how various aspects of the argumentation domain, namely the notion of relevance, and the interactions between arguments, may be used to reduce the computational complexity of searching for an argument while making use of opponent modelling.

To our knowledge, no work on argument strategies has yet utilised the notion of an opponent model to the same extent as we have. The introduction of opponent modelling opens up a number of avenues for future research, and also allows for the creation of new, and novel strategies for argument.

References

1. L. Amgoud, C. Cayrol, and M.-C. Lagasquie-Schiex. On the bipolarity in argumentation frameworks. In *Proceedings of the 10th International Workshop on Non-monotonic Reasoning*, pages 1–9, Whistler, Canada, 2004.
2. L. Amgoud and N. Maudet. Strategical considerations for argumentative agents (preliminary report). In *Proceedings of the 9th International Workshop on Non-monotonic Reasoning*, pages 399–407, 2002.
3. L. Amgoud and H. Prade. Generation and evaluation of different types of arguments in negotiation. In *Proceedings of the 10th International Workshop on Non-monotonic Reasoning*, 2004.
4. P. Besnard, S. Doutre, and A. Hunter, editors. *Computational Models of Argument: Proceedings of COMMA 2008, Toulouse, France, May 28-30, 2008*, volume 172 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.
5. D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 120–125. AAAI, 1996.
6. D. Carmel and S. Markovitch. Model-based learning of interaction strategies in multi-agent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):309–332, 1998.
7. H. H. L. M. Donkers, J. W. H. M. Uiterwijk, and H. J. van den Herik. Probabilistic opponent-model search. *Information Sciences*, 135(3-4):123–149, 2001.
8. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.
9. D. C. Martínez, A. J. García, and G. R. Simari. Progressive defeat paths in abstract argumentation frameworks. In L. Lamontagne and M. Marchand, editors, *Canadian Conference on AI*, volume 4013 of *Lecture Notes in Computer Science*, pages 242–253. Springer, 2006.
10. P. McBurney and S. Parsons. Dialogue games in multi-agent systems. *Informal Logic*, 22(3):257–274, 2002.
11. D. Moore. *Dialogue game theory for intelligent tutoring systems*. PhD thesis, Leeds Metropolitan University, 1993.
12. N. Oren, M. Luck, S. Miles, and T. J. Norman. An argumentation inspired heuristic for resolving normative conflict. In *Proceedings of The Fifth Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems (COIN@AAMAS-08)*, pages 41–56, Estoril, Portugal, 2008.
13. N. Oren and T. J. Norman. Semantics for evidence-based argumentation. In Besnard et al. [4], pages 276–284.
14. N. Oren, T. J. Norman, and A. Preece. Arguing with confidential information. In *Proceedings of the 18th European Conference on Artificial Intelligence*, pages 280–284, Riva del Garda, Italy, August 2006.
15. N. Oren, T. J. Norman, and A. Preece. Loose lips sink ships: a heuristic for argumentation. In *Proceedings of the Third International Workshop on Argumentation in Multi-Agent Systems*, pages 121–134, Hakodate, Japan, May 2006.

16. S. Parsons, P. McBurney, E. Sklar, and M. Wooldridge. On the relevance of utterances in formal inter-agent dialogues. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.
17. H. Prakken. Relating protocols for dynamic dispute with logics for defeasible argumentation. *Synthese*, 127:187–219, 2001.
18. H. Prakken and G. Sartor. *Computational Logic: Logic Programming and Beyond. Essays In Honour of Robert A. Kowalski, Part II*, volume 2048 of *LNCS*, pages 342–380. Springer-Verlag, 2002.
19. I. Rahwan and K. Larson. Mechanism design for abstract argumentation. In *Proceedings of AAMAS 2008*, 2008.
20. R. Riveret, H. Prakken, A. Rotolo, and G. Sartor. Heuristics in argumentation: A game theory investigation. In Besnard et al. [4], pages 324–335.
21. R. Riveret, N. Rotolo, S. G, H. Prakken, and B. Roth. Success chances in argument games:a probabilistic approach to legal disputes. In *Proceedings of the 20th Anniversary International Conference on Legal Knowledge and Information Systems (Jurix 2007)*, pages 99–108, Amsterdam, The Netherlands, 2007.
22. C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.