KING'S College LONDON

University of London

A Dialogue Simulator for Evaluating Persuasion Strategies

Final Project Report

Author: Sapir Gal

Supervisor: Dr Elizabeth Black

Student ID: 1524327

April 16, 2018

Abstract

Argumentation dialogues are increasingly becoming a popular method for formalising communications in multi-agent systems. Some recent studies have attempted to find a way to generate an optimal strategy for persuasion dialogues, which are one type of dialogues where one party is trying to persuade another to do something or believe in some proposition. This project aims to provide means to evaluate further one particular approach that uses planning, by creating a system that can run real dialogues between different types of agents and employing various strategies, including strategies generated by the method of interest. Furthermore, the project provides experiment results obtained by dialogue simulations, showing the effectiveness of the planning approach in practice, how it is affected by different factors, and how it is compared to other strategies.

Originality Avowal

Acknowledgements

I would like to thank to my supervisor, Dr. Elizabeth Black, for her advice, support and guidance throughout the year, while working on this project. I would also like to thank Dr. Amanda Coles, who took time to attend the project meetings, contributed ideas, and provided guidance and support.

Contents

Chapter 1

Introduction

Intelligent agents in multi-agent systems often need to communicate with other agents to achieve their individual or joint goals [21]. The use of argumentation theory in artificial intelligence research has become a growing field of interest in last few decades [24], and has played an important role in formalising communication between agents [11]. Dialogue is one form of argumentation in which more than one agent is involved [13], and it can be classified into different types according to the participants' initial state of belief and what they intend to accomplish [11]. This project is concerned with one kind of dialogue, persuasion, wherein, initially, the participating agents have some conflicting beliefs, and one agent seeks to persuade another to believe the truth of some argument [13]. For example, persuasion can be applied in systems where agents need to cooperate and agree on which action to take in pursuance of their goal [4]; to alleviate conflicts with the intention to achieve an individual goal in non-cooperative systems [15]; or even used by systems to persuade human users to change their behaviour for their benefit (e.g., exercise more) [10].

Typically, in argumentation dialogues, the participating agents exchange arguments in turns. A strategy in an argumentation dialogue defines which arguments the agent should use in order to reach a successful outcome [6]. However, an agent success is also depended on the arguments the other parties in the dialogue choose to use [6]. In recent years, various studies have attempted to propose methods that can be used to determine a strategy for a persuasion dialogue, i.e., a strategy for the Persuader. For example, some methods use planners [6], others use decision trees [8], and some use Markov decision models [7]. This project will focus on one of these methods, Planning for Persuasion [6], that determines a strategy it defines as the ”Optimal Simple Strategy”. As no known work has been done to simulate agents that employ the optimal simple strategy in dialogues, and test them against agents with different strategies. Data on such dialogues can provide insight on the merits and limitations of said strategies, and could be used to assess and analyse the behaviour of strategic agents under different conditions. An understanding of the behaviour of strategic agents is essential to future research in this area, and for the development and improvement of new and existing strategies, methods and techniques.

1.1 Aim and Objectives

This project aims to provide an environment where different types of agents with different strategies can engage in a persuasion dialogue with other agents of the same or another kind, and that records the details of every simulated dialogue so that related information can be easily accessed, read and analysed. The environment will be built around an agent that devises a simple strategy as proposed by Black, Coles and Hampson [6], and will be extended to model additional agents with strategies of varying degrees of sophistication, ranging from stochastic to planned. Having an array of agents is essential to the understanding of how effective is the simple strategy against other strategies, as well as to benchmark the performance of simple strategy agents against other agents.

Moreover, the project aims to use the simulator run experiments, collect simulations data, and analyse the results to answer questions about the effectiveness of the simple strategy in different problems and against different agents, investigate the behaviour of the simple strategy agent in each case, and compare the performance of different strategies.

Chapter 2

Background

This section will start by giving an overview of the field of argumentation and argumentation theory, and discuss their applications in artificial intelligence research and multi-agent systems. It will then continue with an explanation of the persuasion problem and a review of recent work to develop approaches for finding optimal strategies for persuasion dialogues, with a focus on the Planning for Persuasion approach [6], which is the heart of this project.

2.1 Argumentation

A multi-agent system is not merely a system with multiple unrelated agents, but a system in which the participating agents can communicate with other agents in the system. Without the ability to communicate, agents will not be able to solve a conflict of interests, reach agreements, distribute resources, share information or work together to solve problems. A communication process that has the purpose of attaining any of these objectives will likely require an exchange of multiple messages, or in other words, engagement in a dialogue [1]. In order to design and understand such systems, it is necessary to have some form of mechanism to formalise unambiguously the process that is dialogue. One way of achieving this is by modelling agent communication using argumentation theory.

2.1.1 Argumentation Theory

Argumentation theory is a diverse field of research, spanning the disciplines of philosophy, linguistics, psychology [17], logic, and law theory [24], that has been studied from the days of ancient Greek philosophers [5]. In recent years, it has found applications in various areas, including artificial intelligence and multi-agent systems research [17]. One of the primary uses of argumentation in multi-agent systems is agent communication, but it is also used by autonomous agents for reasoning and belief revision [11], which are actions the agent takes to decide how to behave. The use of argumentation theory in artificial intelligence research originated from the failure of classical propositional logic to handle reasoning with partial and uncertain information [24], and has laid the basis for formalising agent communication [11].

Argumentation in multi-agent systems can be defined as a principled process in which different arguments, possibly conflicting, are communicated in order to reach an agreed conclusion [11]. One of the most common approaches to modelling this kind of processes is by using an abstract argumentation framework which can be reasoned about, as proposed by [19].

2.1.2 Abstract Argumentation Frameworks

Definition 1: Dung [19] defines an *argumentation framework* as a pair

$$\mathcal{AF} = \langle \mathcal{AR}, attacks \rangle$$

where $\mathcal{AR}$ is a set of arguments, and attacks is a binary relation on $\mathcal{AR}$, such that

$$attacks \subseteq \mathcal{AR} \times \mathcal{AR}$$

If $attacks(a, b)$ holds, then a attacks b, i.e., argument a rebuts argument b.

Each abstract argumentation framework can be represented as a graph, such that each argument $a, b \in \mathcal{AR}$ is a node, and each attack $(a, b) \subseteq attacks$ is a directed edge from a to b.

This basic definition can be used to determine the acceptability of arguments, thus allowing to determine whether a proponent was successful achieving its goal. The acceptability of an argument can be established by different notions, one of which is the grounded extension defined by Dung [13]. A grounded extension of an argumentation system contains all the arguments that are guaranteed to be accepted. For example, arguments that have no attackers must be in the grounded extension; whereas, for the same reason, arguments that are attacked by these cannot be in the grounded extension. Iterating this process of identifying arguments that are in the extension, and eliminating those that are out until there are no further changes, will result in a new set of arguments that is the grounded extension [13]. To formalise the notion of grounded extension, it is necessary first to define some key properties that may hold for members of the domain.

It is reasonable to assume that if the set of arguments accepted by Ag can defend an argument a against all attacks, then a is acceptable for Ag, and Ag accepts a only if it is acceptable. In other words, the set of arguments accepted by Ag is a set that defends all its members [19]. This leads to the next definition by Dung [19].

Definition 2: An argument $a \in AR$ is acceptable with respect to a set of arguments $S \subseteq AR$ if and only if for every $b \in AR$ such that $attacks(b, a)$ holds, S attacks b.

This definition of the acceptability of an argument can be used to define the characteristic function of an argumentation framework [19].

Definition 3: The characteristic function of an argumentation framework AR is

$$F_{AF} : 2^{AR} \Longrightarrow 2^{AR}$$

such that

$$F_{AF}(S) = \{a \,|\, a \text{ is acceptable wrt } S\}$$

Then, intuitively, a grounded extension as described above cannot contain any arguments that attack one another. Such set is said to be conflict-free [19].

Definition 4: A set of arguments S is conflict-free if for every $a, b \in S,\ (a, b) \notin attacks$ holds.

From definition 3 and definition 4, an admissible extension can be defined as below [19].

Definition 5: A set of arguments S is admissible if and only if S is conflict free and $S \subseteq F_{AF}(S)$

Dung also defines a variant of the admissible extension called complete extension [19].

Definition 6: An admissible set of arguments S is a complete extension if and only if S contains all the arguments it defends, i.e., $S = F_{AF}(S)$

Finally, a grounded extension can be defined as a set of arguments that contains all the arguments that are non-defeated, that is, arguments that are not attacked by any argument, as well as all the arguments that are defended by them, directly or indirectly. The formal definition is given below [12].

Definition 7: A grounded extension is the minimal complete extension (with respect to set inclusion).

For each argumentation framework, there always exists a unique grounded extension [12]. The proponent will be considered successful in the dialogue if all its goal arguments are in the grounded extension.

It is important to note that when using the grounded extension notion to determine the acceptability of arguments, it is done under the assumption that all arguments are equally weighted, and therefore that all attacks carry an equal impact, which is rarely the case in reality. For example, in a court of law arguments based on forensic evidence tend to have more weight over arguments based on hearsay [13]. However, the strategies explored in this project use similar semantics and are not concerned with the relative strength of arguments, and therefore it has no effect on the work in this project.

2.2 Persuasion Dialogues

Walton [5] describes six types of basic dialogues, defined by the initial state of the dialogue, the participant's goal, and the goal of the dialogue: persuasion, inquiry, negotiation, information seeking, deliberation, and eristic. Persuasion dialogue is a dialogue in which the initial state is such that there is a conflict of opinions between the participants, each participants' goal is to persuade the other parties to accept their views, and the purpose of the dialogue is to resolve or clarify an issue. When considering a simplified version of a dialogue with only two parties, a persuasion dialogue is a dialogue in which the persuader has a goal to convince the persuadee of the truth of some proposition. Walton further describes the persuader as the participant who has the burden to prove some thesis, while the persuadee has the role of preventing the proponent from achieving such proof.

In more formal terms, a persuasion dialogue, $PD = [a_1, a_2, \cdots, a_m]$, is a sequence of arguments asserted by both participants alternately, such that a_1 is the thesis the persuader tries to prove. Let $\mathcal{AF_D} = \langle \mathcal{AR_D}, attacks_{\mathcal{D}} \rangle$ be an argumentation framework, such that $\{a_1, a_2, \cdots, a_m\} = \mathcal{AR_D}$, and for every other $a_k \in PD, (a_{k+1}, a_1) \in attacks_{\mathcal{D}}$. This is a further simplified description of a dialogue, in which the persuader's thesis is a singleton and agents may only assert one argument at a time alternately. A persuader is considered to have proven their thesis if and only if at the end of the dialogue a_1 is in the grounded extension of $\mathcal{AR_D}$.

2.3 Strategies

A strategy in argumentation dialogue specifies what arguments should the proponent utter in order to prove its thesis. While many studies have explored the evaluation and generation of arguments, there have been only a few studies focusing on determining strategies. This section will discuss in detail the work of Black, Coles, and Hampson [6], that developed a method to find an optimal strategy using planning, and will briefly discuss other approaches that use decision trees and Markov decision models. However, before proceeding to examine the research undertaken in this area of work, it is important to define some terms that will be used in the following sections and chapters. As the work reviewed in this project attempt to finds strategies make the persuader successful; therefore, in the next sections, the persuader will be referred to as the proponent, and the persuadee as the opponent.

2.4 Planning for Persuation

Black, Coles, and Hampson [6] proposed a method to represent and solve the strategic argumentation problem as a planning problem, expressed in the PDDL2.1 planning language, and using an automated planner to find strategies. The search for strategies was narrowed to consider only simple strategies, which the paper defines as a strategy that determines a finite sequence of arguments for the proponent to assert without regard to previous moves. This method aims to find an optimal simple strategy, that is, a simple strategy that can guarantee success with a certain probability.

The method described in this study is designed to find an effective strategy for a proponent, given that it has some uncertain model of its opponent, for any strategy the opponent may employ and without assuming the goal of the opponent. An opponent model represents what the proponent believes to be the possible sets of beliefs the opponent might hold, and the probability they attach to each possible set of beliefs to be the actual set of arguments of the opponent. For example, $OpponentModel = (\{a\} : 0.75), (\{b, c\} : 0.25)$ means that the proponent believes that there is a probability of 0.75 that the opponent knows only a, and a probability of 0.25 that it knows only b and c. In addition, it considers a case where the opponent may derive new arguments from arguments asserted by the proponent, which they can later use in the dialogue.

Finding a strategy that will be effective regardless of the arguments the opponent might assert is a PSpace-complete problem [14]. However, modelling it as a planning problem and restricting the search space to simple strategies only, provides a solution with a certain probability to be successful no matter what strategy the opponent employs. Planners assume deterministic actions and full knowledge, while the persuasion problem is non-deterministic (there is no mapping between each move of the proponent to a specific response by the opponent), and the proponent does not have complete knowledge on the opponent. Such planning problems can be solved by compiling away the uncertainty [20], which was achieved by considering all possible dialogues against each possible opponent according to the opponent model. A plan is guaranteed to be successful against a particular possible opponent only if it is effective for every possible dialogue against that opponent. The probability of guaranteed success of a plan denoted by λ is the sum of probabilities of the possible opponents against which the plan was found to be effective. To find an optimal solution, the planner searches for a plan that maximises λ, by first finding a simple strategy with $\lambda > 0$. Then, it will try to search for a strategy that yields a greater λ than that given by the previous solution. It will repeat the last step until it cannot find any more solutions; the optimal simple strategy will be the last solution found.

The performance of the planning approach was evaluated by considering the probability of guaranteed success it can produce, and for the time it took to find an optimal strategy. The search time was benchmarked against the performance of a depth-first search based brute-force algorithm, referred to as the naive algorithm, which explores every possible simple strategy to find an optimal one. The evaluation involved argumentation frameworks of various structures and different level of difficulties, including bipartite, cycle and ladder graphs. Bipartite structures are assumed to be less challenging, as the proponent cannot contradict its own arguments, and so it does not need to worry about undermining itself. However, arguments in cycle and ladder frameworks cannot be classified as helpful or harmful conclusively and may have a different effect depending on the arguments asserted by the opponent.

An argumentation framework that conforms to the $cycle_n$ structure has arguments $\mathcal{AR} = \{a\} \cup \{b_i, c_i : i < n\}$, where a is the goal argument (singleton), and $attacks = \{(b_i, a), (c_i, b_i) : i < n\} \cup \{(b_{i-1}, b_i), (c_{i-1}, c_i) : 0 < i < n\} \cup \{(b_{n-1}, b_0), (c_{n-1}, c_0)\}$; the proponent is associated with the arguments $\{a\} \cup \{c_i : i < n\}$, and the opponent associated with the arguments $\{b_i : i < n\}$ [6]. Figure 2.4 illustrates a $cycle_3$ graph, where the white node represents the proponent's arguments, and the greyed nodes are arguments that may be used by the opponent.

As can be seen in the example, if the opponent knows only b_2, for instance, argument c_2 can helpful for the proponent . Nevertheless, it the opponent also knows b_0, then c_2 can be

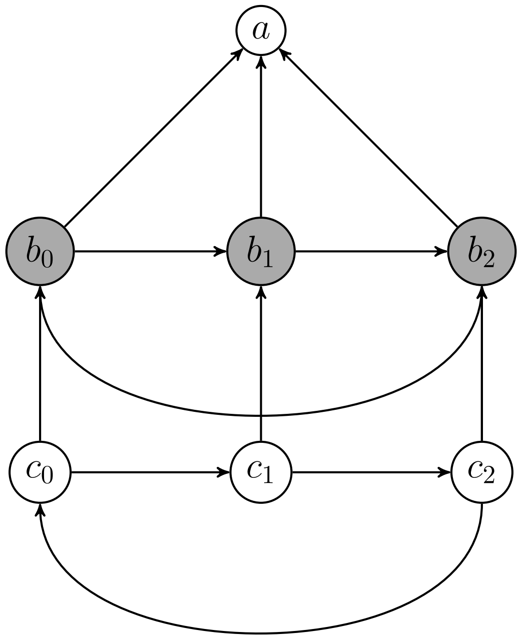


Figure 2.1: Example of $cycle_3$

harmful for the proponent as it also attacks c_0, which is the only argument the proponent can use to attack b_0. Therefore, the proponent has to be careful not to compromise their goal when deciding which arguments to assert, and take into account all the possible sets of arguments that the opponent might have [6].

An argumentation framework that conforms to the $ladder_n$ structure has arguments $\mathcal{AR} = \{a\} \cup \{b_i, c_i : i < n\}$, where a is the goal argument (singleton), and $attacks = \{(b_0, a), (c_0, a)\} \cup \{(b_i, b_{i-1}), (c_i, c_{i-1}) : 0 < i < n\} \cup \{(b_i, c_i) : i < n\}$; the proponent is associated with the arguments $\{a\} \cup \{b_i, c_i : i = 1, 3, \cdots\}$, and the opponent is associated with the arguments $\{b_i, c_i : i = 0, 2, \cdots\}$ [6]. Figure 2.4 illustrates an example for a $ladder_4$ argumentation framework, where asserting $b3$ is beneficial to the proponent only if the the opponent knows b_2 and b_0, but nothing else.

In addition to different frameworks, the planning approach was also evaluated for randomly selected opponent models with different sizes to examine the effect change of opponent model size may have.

The results [6] showed that the planner outperformed the naive algorithm, and had better search times for all the $ladder_n$ examples. For the other examples, the naive algorithm performed better for smaller instances. However, the planner was significantly faster as n increased. In terms of effectiveness, solutions with a good probability of guaranteed success ($\lambda > 0.7$) were found for five out 11 examples, while for the $cycle_n$ examples lower probabilities were observed.

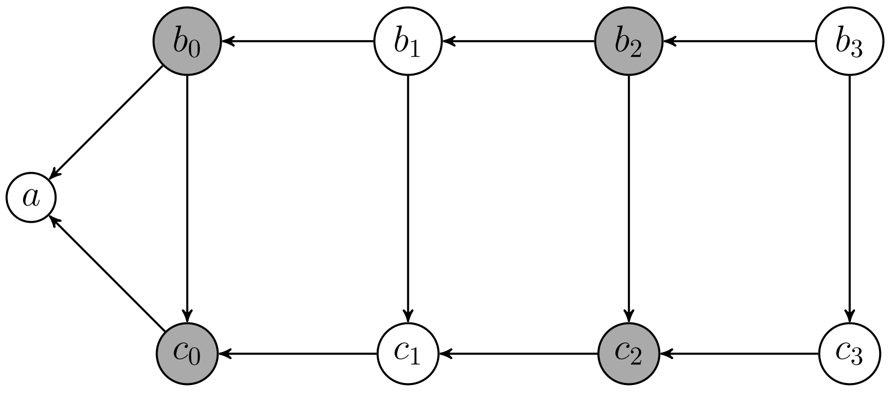


Figure 2.2: Example of $ladder_4$

While the structure of the framework seemed to affect λ, the size of opponent models was found to have no meaningful impact on the results (see [6] for detailed results).

This project will evaluate the performance of the optimal simple strategy in practice, by simulating dialogues for different argumentation framework structures, opponent model sizes, and proponent and opponent behaviours. As a probability of success can only be guaranteed if the opponent asserts arguments that are in the proponent's opponent model, this project will explore how the optimal simple strategy performs in practice against unpredictable opponents. Furthermore, it will examine if the optimal simple strategy can be used to create a more powerful proponent that takes into account what it learns about the opponent by the arguments the latter asserts.

2.5 Strategic Sequences of Arguments for Persuasion Using Decision Trees

Hadoux and Hunter [8] proposed using state-of-the-art decision theory methods to find a policy. A policy is a mapping between every possible state of a particular dialogue, to the best possible argument to assert in that state.

Similarly to Planning for Persuasion [6] approach, this method deals with two agents and makes no assumptions on the behaviour of the opponent. However, it cannot be used with cyclic problems. Moreover, instead of an opponent model, the proponent holds a belief model of the opponent, which is initially empty. Whenever the opponent asserts an argument, the proponent updates the belief model accordingly and uses the updated model to make decisions on which arguments to assert. Formally, given an argumentation framework $\mathcal{AF} = \langle \mathcal{AR}, attacks \rangle$, a belief model is a mass distribution over all possible subsets of AR, such that $\sum_{X \subseteq \mathcal{AR}} P(X) = 1$.

Then, for an $a \in \mathcal{AR}$, $P(a)$ is the degree to which the opponent believes in a, such that $P(a) > 0$ represents a degree of belief, and $P(a) \leq 0$ represents a degree of disbelief.

This approach also takes into account a parameter called *horizon*, which represents the maximum number of turns that could be played. The rationale behind this parameter is that the opponent may lose interest in the dialogue after some time, and this parameter allows the proponent to consider this in the decision process.

To find the optimal policy, the dialogue is represented as a decision tree, in which a path is a possible permutation of arguments, and all possible permutations are paths, i.e., paths are possible dialogues. The tree has two types of nodes: decision nodes, which are associated with the proponent and are square, and chance nodes, which are associated with the opponent and are round. An asserted argument is an edge between two nodes. The central square node is the root, and $v_1, \cdots, v_i$ are the leaves, where each leaf is a valuation given to one particular dialogue to describe how desirable is that execution with regards to achieving the dialogue goal of the proponent. Figure 2.5 shows an example of such a decision tree. The problem is solved by using decision rules, such as the MaxiMax and MaxiMin rules. A discussion o this methods falls outside the scope of this report.

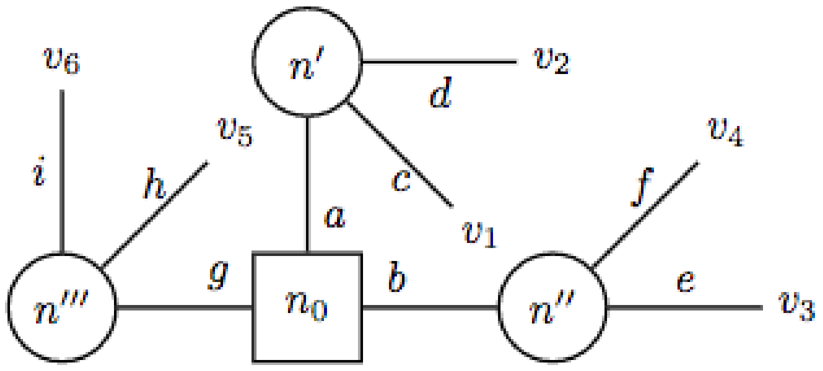


Figure 2.3: An example of an argumentation decision tree [8]

2.6 Optimization of Probabilistic Argumentation with Markov Decision Models

Another approach for finding a policy [7], proposes to model the persuasion problem as a sequential decision problem, that is, a problem in which the desirability of being in a particular state depends on previous decisions, and to use Markov Decision Processes (MDP) to solve it. An MDP is a framework that allows solving sequential decision problems under uncertainty by considering:

- All possible states
- All possible actions that can be taken
- The probability to transition to a state $s\prime$ after making a certain move, a, from a state s, for each possible state s.
- A reward function that represents the utility, or the preference, of being in one state over another, or of taking a particular action over another.

MDPs provide a solution in the form of optimal policy, that specifies which action the agent needs to take at each state to maximise the expected utility.

MDPs as described above work only in fully observable environments, which in the case of dialogue means that the proponent must always know in which state it is. However, this method [7] assumes an opponent with a stochastic behaviour, which means its actions are not determined by the current state or by arguments asserted previously by themselves or the proponent. In addition, the opponent's initial belief state is unknown, hence the necessity for a solution that is able to deal with partial observability. The fact that the exchanged arguments in the dialogue are known to the proponent enabled to solve the problem using an extended MDP framework called Mixed Observability Markov Decision Processes (MOMDP) [18], that can deal with partial observability.

Chapter 3

2. Requirements

The system is intended to be used by a user who wishes to simulate dialogues between different types of agents with different strategies, using different kinds of problems, and to generate simulations data that will allow them to analyse the behaviour and performance of agents and strategies under different conditions.

The requirements for the project can be divided into two sets: user requirements and software requirements. The first set of requirements relates to the basic, high-level functionality that will allow the user to use and interact with the software; the second set details what the underlying software should deliver for the user, that is, the low-level capabilities of the software.

3.1 User Requirements

Interface

1. The user should be able to interact with the simulator using a basic graphic interface and do the following:
   (a) The user must be able to run and stop dialogue simulations.
   (b) The user must be able to set the properties for each dialogue, namely: proponent type, opponent type, number of runs, and a problem file; a problem file contains the argumentation framework to be used in the dialogue, the arguments that the proponent may use, and the opponent model of the proponent, equivalent to that in Planning for Persuasion [6]
   (c) The user must be able to save and load dialogue settings to and from a json file.

(d) The user must be able to create random problem files that can be used in dialogues.

(e) The use must be able to select random opponent model for an existing problem file.

2. The user should be able to run dialogues form the command line without having to use the graphic interface. The settings from the run should be provided as a json file, that can be easily generated using the graphic interface.

3.2 System Requirements

Agents

1. There must be proponent agents and opponent agents, with at least two different strategies for each.

2. There must be at least one type of opponent agent that can derive arguments given a closure function. A closure function is a relation from a set of arguments asserted by the agents to a subset of the arguments in the argumentation framework that have not been asserted yet. It means that if there is a closure function from a to b, once a is asserted, the opponent may be able to derive b, depending on their type.

3. Agents must be able to participate in dialogues and assert arguments when needed.

4. Opponent agents must be able to reset their state so they can compete against several proponents. As the state of some agents is randomly initialised, having the ability to reset their state is essential to ensure a fair comparison how do different proponents types perform against different opponent types, as it guarantees they all competed against the same agents.

Simulator

1. The simulator must be able to receive as input a problem file in a certain format that it will be able to parse, and extract an argumentation framework (arguments and attacks), a set of arguments for the proponent, opponent models and their probabilities, dialogue goal, a and a closure operation if exists.

2. The simulator must be able to run a planner as a sub-process, provide it with the required input to generate an optimal simple strategy, and parse the results.

3. The simulator must be able to edit the planner input files for two different cases:

   (a) To generate a new strategy that takes into account arguments that have already been asserted, by both the proponent and the opponent, at a certain state in a dialogue
   (b) To change the dialogue goal and allow to generate contradicting strategies. In other words, to generate a strategy to contradict a certain argument, rather than to prove it.

4. The simulator must run dialogues according to a given input, indicate the progress in the graphical interface, and save the results to a report. The results must be clear and capture all information that can be used for analysis.
5. For each dialogue, the report must record:
   (a) The proponent and opponent types.
   (b) The arguments asserted by the proponent.
   (c) The arguments asserted by the opponent.
   (d) The strategy or strategies of the proponent if applicable, i.e., the list of arguments found by the planner to be the optimal simple plan.
   (e) The search time from the planner.
   (f) The probability for guaranteed success if applicable
   (g) Whether or not the proponent has achieved their dialogue goal.
6. The simulator must be able to use a planner to search for strategies, and as such, must be able to create and edit problems files that can be used as input to the planner.

Chapter 4

Design & Implementation

The simulator provides an environment in which two agents, a proponent, and an opponent, can engage in persuasion dialogues and exchange arguments. Therefore, designing an agent that is able to use the optimal simple strategy [6] will allow evaluating the planning for persuasion method [6] in real dialogues. This chapter details the components that comprise a simulator and discusses their roles, designs, and implementations and how they work as a whole.

4.1 Overview

The simulator has three main components: an agent, an argumentation framework, and a dialogue, as illustrated in figure 4.1.

Agent	an agent in the system represents a party in a dialogue that embodies some method to generate a strategy, such as the planning approach [6], and can be either a proponent or an opponent. As already described multiple times in previous chapters, the proponent has the aim to persuade the opponent to believe in or to accept some argument(s). However, the intentions of the opponent have not been addressed yet. There are two objectives the opponent may try to achieve: it will either have its own goal argument(s) to convince the proponent to accept; or, it will aim to contradict the proponent's goal argument(s).
Argumentation Framework	an argumentation framework can be thought of as the domain of the topic of the dialogue. It specifies which arguments are relevant

	to the topic, and also the contradictions between them.
Dialogue	a dialogue has a proponent agent, an opponent agent and an argumentation framework. It is a process in which the proponent and the opponent exchange arguments related to the dialogue topic, with the intention to achieve their dialogue goal. A dialogue is said to have a successful result if the proponent has won, or unsuccessful if the proponent lost. It is important to note that having said that the proponent won does not imply necessarily that the opponent lost, as the opponent may achieve its goal argument(s) without contradicting the proponent. However, as this project focuses on evaluating the performance of the proponent, the opponent only serves as a tool for testing, and therefore, it is of no interest whether they win or not.

All the components described above are constructed according to dialogue properties given as input. The input defines for each dialogue what strategies the proponent and the opponent will employ, as well as the argumentation framework that will be used. Moreover, it will decide which arguments can be used by the proponent, which arguments can be used by the opponent and the goal argument(s) of the proponent, as will be described in the next sections in detail. The simulator will execute dialogues according to the input, and will output the results of all executions.

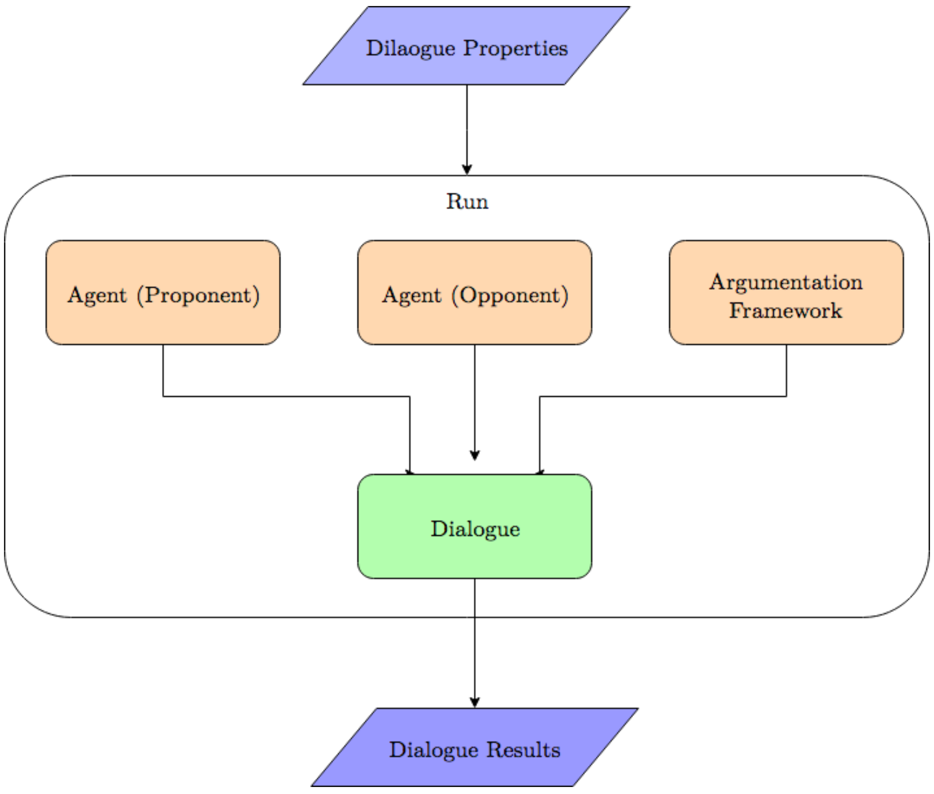


Figure 4.1: A high level overview of the system

4.2 Agent

Given an argumentation framework $\mathcal{AF} = \langle \mathcal{AR}, attacks \rangle$, as defined in section 2.1.2, an agent in its most general form is a quadruple $\mathcal{AG} = \langle Args, strategy, Goal, \mathcal{M}, \mu \rangle$, where:

Args is a non-empty finite set of beliefs, $Args \subseteq \mathcal{AR}$. These are the arguments available for $\mathcal{AG}$ to assert in the course of a dialogue. $\mathcal{AG}$ will not assert any other argument a, such that $a \in Args^C$.

strategy a label that determines which method the agent will use to generate its strategy. There are four different methods: planning, re-planning, stochastic and random. Each method produces a strategy by deciding a set of arguments $SArgs \subseteq Args$ for $\mathcal{AG}$ to assert in a dialogue, and the order in which they should be affirmed. For all agent types, $SArgs$ is established prior to the commencement of any dialogue in which they may take part. However, some agent types can replace their strategy with another once or more in the course of a dialogue, where a new strategy may have different arguments.

Goal is the dialogue goal of $\mathcal{AG}$. Let $GE_{\mathcal{AF}}$ be the grounded extension of $\mathcal{AF}$, and $acc : \mathcal{AR} \to \{true, false\}$ a total function, such that

$$acc(x) = \begin{cases} true, & \text{if } x \in GE_{\mathcal{AF}} \\ false, & \text{otherwise} \end{cases}$$

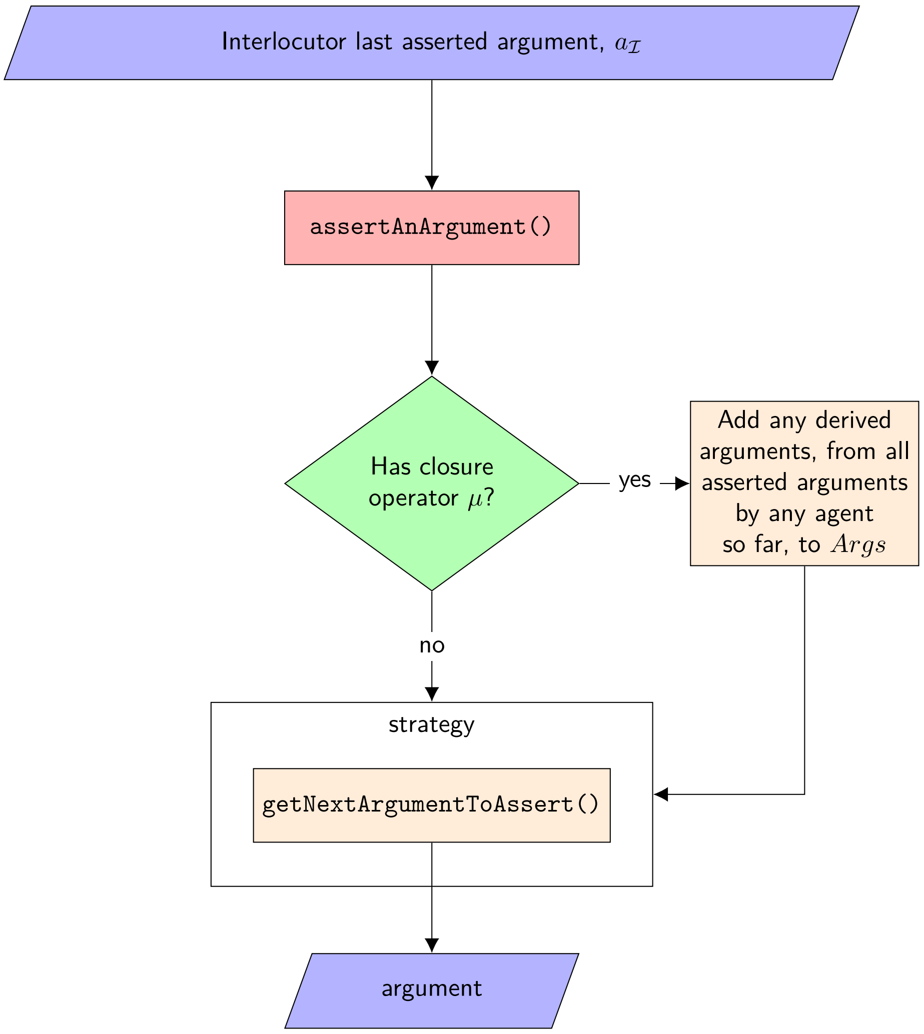
Then, $Goal = \langle \mathcal{G}, acceptance \rangle$ is a pair, where $\mathcal{G} \subseteq Args$ is a non-empty, finite and conflict free set of goal arguments, and $acceptance \in \{true, false\}$ is an indicator of the intention $\mathcal{AG}$ has with regards to $\mathcal{G}$, such that the dialogue goal of $\mathcal{AG}$ is that $acc(g) = acceptance$ holds for every $g \in \mathcal{G}$.

$\mathcal{M}$ is a representation an agent $\mathcal{AG}$ may have of some other interlocutor agent, $\mathcal{AG}_{\mathcal{O}}$, denoted by a pair $\mathcal{M} = \langle \mathcal{E}, \Pr \rangle$, where $\mathcal{E}$ is a finite set of possible sets of arguments $\mathcal{AG}_{\mathcal{O}}$ might have, and $\Pr : \mathcal{E} \to \mathbb{Q}$ is a probability distribution over $\mathcal{E}$ that denotes what $\mathcal{AG}$ believes to be the likelihood that $\mathcal{O}_i = Args_{\mathcal{O}}$, for every $\mathcal{O}_i \in \mathcal{E}$, where $Args_{\mathcal{O}}$ is the actual set of arguments of $\mathcal{AG}_{\mathcal{O}}$, and $\Pr(\mathcal{O}_i) > 0$ and $\sum_{i=1}^{n} \Pr(\mathcal{O}_i) = 1$. This is equivalent to an opponent model in Planning for Persuasion [6].

μ is a closure function $\mu : \mathcal{P}(\mathcal{AR}) \to \mathcal{P}(\mathcal{AR})$, such that if $\mu(\mathcal{B}) = \mathcal{C}$, and $\mathcal{C} \notin$

$Args$, and $\mathcal{B}$ was asserted, then $Args \leftarrow Args \cup \mathcal{C}$. The closure function allows an agent to derive new arguments from arguments the interlocutor asserts.

An agent must also be able to assert an argument from $SArgs$ when it is asked to do so, which could be the empty argument if $SArgs = \emptyset$, $null \in SArgs$ or if required to assert the empty argument according to the rules of the dialogue. The input for the process of selecting which argument to assert is the last argument asserted by the interlocutor. Then, the agent needs to reply with its own argument. An agent with a closure operator will first update $Args$ if required. Otherwise, it will proceed as any other agent to use its strategy to select an argument and return it, as illustrated in figure 4.2.



Interlocutor last asserted argument, $a_{\mathcal{I}}$
assertAnArgument()
Has closure operator μ?
yes
Add any derived arguments, from all asserted arguments by any agent so far, to $Args$
no
strategy
getNextArgumentToAssert()
argument

Figure 4.2: The process of asserting an argument for an agent $\mathcal{AG}$

There are different types of agents, which vary in the components they are required to have and in their behaviour. However, all types of agents adhere to the dialogue rules described in section 4.3.

4.2.1 Stochastic Agent

Stochastic agents manifest the simplest form of an argumentative agent, as they assert arguments randomly from $SArgs$, where $SArgs$ is decided according to the role of the agent in the dialogue (i.e., proponent or opponent), and its $strategy \in \{stochastic, random\}$ label.

Whenever required to assert an argument, the agent will sample an argument from a discrete uniform distribution over $SArgs$, where for every $a \in SArgs, P(a) = \frac{1}{n}$, and $n = |SArgs|$. Although a Stochastic Agent may have some belief on some other agent $\mathcal{AG}_{\mathcal{O}}$, i.e., $\mathcal{M} \neq \varnothing$, it will not use it to determine $SArgs$, its moves, or to make any other decision.

There are two types of stochastic agents, rational and naive. A stochastic agent is rational if and only if its set of arguments $SArgs$ is conflict-free; a stochastic agent that is not rational is said to be naive.

```
input  : Set of newly derived arguments derivedArguments
output: An argument $a \in SArgs \cup$ null
if derivedArguments $\neq \emptyset$ then
    SArgs $\leftarrow SArgs \cup$ derivedArguments
end
if $SArgs = \emptyset$ then
    return null
end
dist $\leftarrow$ new DiscreteUniformDistribution($SArgs$)
arg $\leftarrow$ dist.Sample
SArgs $\leftarrow SArgs \setminus \{$arg$\}$
return arg
```

Algorithm 1: Selection of the next argument to assert for a Stochastic agent

4.2.2 Planning Agents

Unlike stochastic agents, planning agents do not make random decisions. They use a planner to generate a strategy that not only dictates which arguments they should assert, but also the order to assert them. A strategy is a pair $\mathcal{S} = \langle \bar{s}, p(success|\bar{s}, \mathcal{M}) \rangle$, where $\bar{s} \in Args$ is an ordered list of n pairwise distinct elements $\bar{s} = (s_1, \dots s_n,)$, and $p(success|\bar{s}, \mathcal{M})$ is the probability of guaranteed success of an agent $\mathcal{AG}$, using $\bar{s}$ to determine its arguments, against some agent $\mathcal{AG}_{\mathcal{O}}$, of which $\mathcal{AG}$ has representation $\mathcal{M}$.

There are two types of planning agents: simple planning, which embodies the proponent described in Planning for Persuasion[6], and re-planning, which uses the optimal simple strategy to demonstrate a new type of behaviour. A simple planning agent uses the same strategy from the start of a dialogue to its end unless it has a closure function μ, which in that case, it may update its strategy if a new argument was derived. A re-planning agent, on the other hand, may update its strategy multiple times in a dialogue when it gains more knowledge about the real beliefs of $\mathcal{AG}_{\mathcal{O}}$, from the arguments $\mathcal{AG}_{\mathcal{O}}$ asserts. This will occur if and only if the agent is asked to assert an argument after the interlocutor has asserted a non-empty argument. The process of updating a strategy for a simple planning agent and a re-planning agent, as well as where it takes place in the process of argument assertion, is illustrated in figures 4.3 and 4.4 respectively.

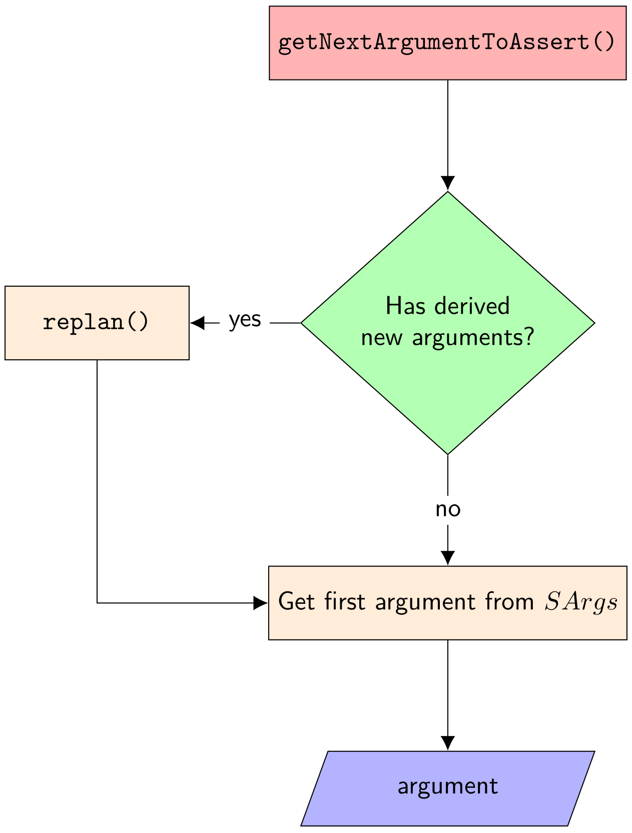

getNextArgumentToAssert()
Has derived new arguments?
yes
replan()
no
Get first argument from $SArgs$
argument

Figure 4.3: The process of updating a strategy for a Simple planning agent with μ

Figure 4.4 is simplified and does not explicitly take into account the closure operator μ, as derived arguments necessarily mean the interlocutor has asserted a non-empty argument.

There is a special case in which a re-planning agent will need to update its opponent model before it can re-plan: it is when the opponent might assert an argument from $\mathcal{AR}$ that is not in the opponent models. When it happens, the proponent will add the unpredicted argument to every set in the opponent model, and will also create a new set of arguments that will

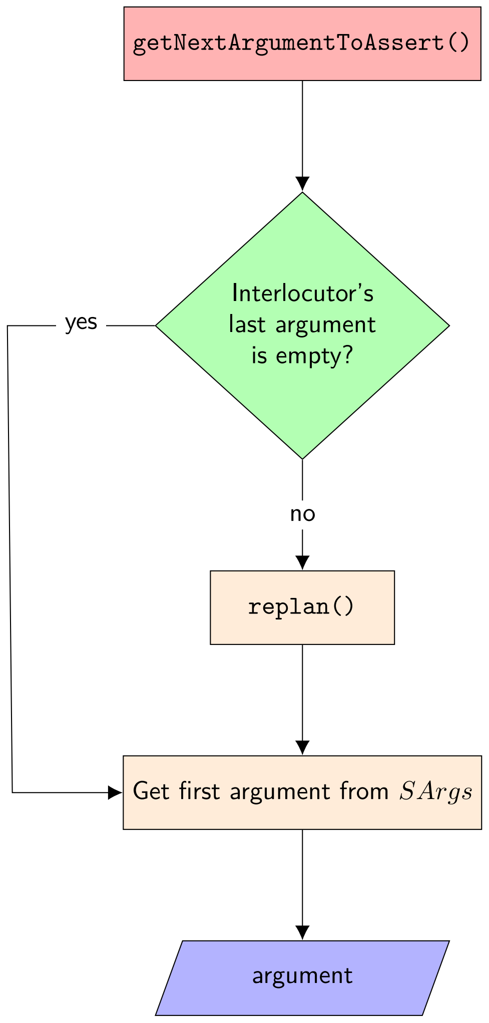

getNextArgumentToAssert()
Interlocutor's last argument is empty?
yes
no
replan()
Get first argument from $SArgs$
argument

Figure 4.4: The process of updating a strategy for a Re-planning agent

be consisted of all the unpredicted arguments if one does not exist. The probabilities in the updated opponent model will be uniform, as it is clear that the knowledge the proponent thought he had was inaccurate. Nonetheless, as it is impossible to determine that opponent model holds no true information, no sets will be removed. For example, if a proponent has the following opponent model $\mathcal{M} = \{(\{a\}, 0.4), (\{b\}, 0.6)\}$ and the opponent asserts c, then the opponent model will be updated as follows $\mathcal{M} = \{(\{a, c\}, 1), (\{b, c\}, 1), (\{c\}, 1)\}$. If now the opponent asserts another unpredicted argument, d, the opponent model will be updated again to $\mathcal{M} = \{(\{a, c, d\}, 1), (\{b, c, d\}, 1), (\{c, d\}, 1)\}$.

4.3 Dialogue

A dialogue represents the process in which two agents, a proponent and an opponent, exchange arguments in turns.

A dialogue is handled in accordance with the following rules: the proponent always gets the first turn, and must assert a non-empty argument in that turn; an agent may only assert one argument in its turn, which may be the empty argument (except for the case described in the first rule); each argument can only be asserted once in a dialogue, by one of the agents; and, if at any point in a dialogue an agent is successful in achieving its dialogue goals, it will not assert further arguments even if its list of arguments, $SArgs$, is not empty, unless it becomes unsuccessful again. The last rule is enforced by the dialogue and not by the agent. It it were to be implemented as part of the agent's behaviour, it would have required either a circular dependency between the dialogue and the agent, or maintaining the state of the dialogue not only in the dialogue itself, but also in two different agents. Therefore, an agent will only be asked to assert an argument if and only if it has not achieved its goal.

A dialogue will terminate if neither agent is able to assert a non-empty argument, or if both agents asserted the empty argument consecutively. If at the end of the dialogue the proponent's set of goals is in the grounded extension of $\mathcal{AF}_D = \langle \mathcal{AR}_D, attacks \rangle$, where $\mathcal{AR}_D \subseteq \mathcal{AR}$ is the set of arguments asserted in the dialogue by both agents, then the proponent won the dialogue. Otherwise, it has failed to achieve its dialogue goals.

```
input  : A proponent agent $\mathcal{P}$, and an opponent agent $\mathcal{O}$
output: True if the dialogue goal of the proponent was achieved, or false otherwise
agents[0] ← $\mathcal{P}$
agents[1] ← $\mathcal{O}$
lastAssertedArg ← $\alpha$                    // a non-null place holder argument
turn ← 0
repeat
    agent ← agents[turn mod2]
    if not IsSuccessful(agent) then
        lastAssertedArg ← AssertArgument(agent)
        agents[0].args ← agents[0].args − lastAssertedArg
        agents[1].args ← agents[1].args − lastAssertedArg
    else if lastAssertedArg is null then
        return IsSuccessful(agent)
    else
        lastAssertedArg ← null
    end
    ++turn
until not IsEmpty($\mathcal{P}$.args) or not IsEmpty($\mathcal{O}$.args)
return IsSuccessful(agent)
```
Algorithm 2: dialogue

4.4 Problem Files

The properties of the argumentation frameworks to be used in dialogues, as well as the models of the proponent and opponent, are defined in text files equivalent to the examples used in [6]. In the following sections, these text files will be referred to as problem files.

Problem files specify the constituents required to configure simulations and are used as the source of data for constructing agents and dialogues. They are also used to generate the input for the planner.

A problem file must prescribe two components. The first is the argumentation framework, $\mathcal{AF}$, which will be used to identify the arguments that may be used in the course of a dialogue, and to establish whether a particular argument is acceptable with respect to different extensions. The second, is a partial description of a proponent agent $\mathcal{AG}_{\mathcal{P}}$, with $Args, \mathcal{G}, \mathcal{M},$ and μ, where μ is optional, and $\pi_{acceptance}\,(Goal)$ is determined by the agent type.

As a side note, as every argumentation framework can be represented as a graph (as explained in the background chapter), the terms argumentation framework and graph will be used interchangeably in the following sections.

4.4.1 Generating and Adapting Problem Files

Problem files can be either provided as input or produced randomly.

Randomly produced problem files are created based on properties specified by the user:

Number of arguments	the number of the arguments in the framework, that is $|\mathcal{AR}|$.
Opponent Models Size	the number of sets of arguments that the opponent may take, $|\mathcal{E}|$.
Probability of attacks	the probability that for each $a_1, a_2 \in \mathcal{AR}$, there is an attack $(a_1, a_2) \in$ *attacks*. In other words, this property sets the density of the graph. For example, for a random graph with $|\mathcal{AR}| = 16$, there are $16 \cdot 15 = 240$ possible attacks. Setting the probability of attacks to 0.25 means that the generated graph will have $0.25 \cdot 240 = 60$ attacks.
Avoid self attacks	specifies whether or not self-attacking arguments are allowed, i.e., whether it is allowed to have an arguments $a \in \mathcal{AR}$ such that $(a, a) \in attacks$.
Enforce tree shape	specifies whether or not to enforce the generated graph to be a connected acyclic graph.
Graph name	the name that will be given to the newly generated graph.

The implementation makes use of the Tweety Library [23] to generate a random graph with the settings above. However, other then an argumentation framework, a problem file also needs a description of the proponent, and more specifically a description of a proponent agent $\mathcal{AG}_{\mathcal{P}}$, with $Args, \mathcal{G}, \mathcal{M}$, but no μ due to the scope of the project. Assuming a rational proponent, deciding $Args$ requires selecting a random conflict-free subset from AR. This is done by finding all the conflict free subsets in AR, and selecting a random non-empty subset $\mathcal{CF}$, such that $\mathcal{CF}$ has the maximal number of arguments while $|\mathcal{CF}| \leq \lceil \frac{1}{2} \mathcal{AR} \rceil$. The reason for this is that increasing the number of arguments means increasing the probability to have a conflict in a subset. Therefore, there is an inverse relationship between the number of arguments in a subset, k, and the number of occurrences of conflict-free subsets with size k. This means that if this restriction is removed, it is likely that the proponent will have a small set of arguments with which it is impossible to ever win. Simulating dialogues with problems where one party can never win provides no beneficial information about the behaviour of the different agents, as the result could be deduced simply by inspecting the argumentation framework. For the same reason, the opponent model is carefully selected to ensure at least one $\mathcal{O}_i \in \mathcal{E}$ can contradict the proponent's thesis. First, the proponent's goal argument is sampled randomly from a discrete uniform distribution over $Args$. Then, a number of subsets, as determined by the user, are selected from all conflict-free subsets of $\mathcal{AR} \setminus Args$, denoted by $\mathcal{O}$. If the proponent's goal argument is in the grounded extension when $\mathcal{O}_i$ is in the grounded extension, for every $\mathcal{O}_i \in \mathcal{E}$, one $\mathcal{O}_i \in \mathcal{E}$, will be selected randomly to be removed and will be replaced by some $\mathcal{O}_i \in \mathcal{W}$, where $\mathcal{W} \subseteq \mathcal{O}$ is a subset of all sets $\mathcal{O}_i$ such that if $\mathcal{O}_i \in \mathcal{W}$ is in a grounded extension, the proponent's goal is necessarily out. The probability distribution $\mathcal{P}$ over the produced $\mathcal{M}$ is a discrete uniform distribution, such that $\mathcal{P}(\mathcal{O}_i) = \frac{1}{n}$ for every $\mathcal{O}_i \in \mathcal{E}, 1 \leq i \leq n$.

Other than generating random graphs, it is also possible to re-generate an opponent model for existing problem files. This feature is particularity useful for existing problems with a meaningful graph structure, such as the $cycle_n$ and $ladder_n$ graphs discussed in section 2.4. The new sets $\mathcal{O}_i \in \mathcal{E}$ will be selected randomly from all the conflict free subsets of $\mathcal{AR} \setminus Args$. The new $\mathcal{M}$ will have no more than k sets $\mathcal{O}_i \in \mathcal{E}$ with equal probability, where $1 \leq k \leq 8$ is given as input, and the actual number of sets $|\mathcal{E}|$ is bounded by the number possible conflict-free subsets of $\mathcal{AR} \setminus Args$.

4.4.2 Problem Files for Planning Opponents

As the planner intended to search for an effective strategy for a proponent, there needs to be a way to represent a planning opponent, given by $\mathcal{M}$ in some problem file, as a proponent agent, $\widehat{\mathcal{AG}_\mathcal{P}}$, with $\widehat{Args}, \widehat{\mathcal{G}}$ and $\widehat{\mathcal{M}}$ of its own, in a new problem file. However, $\mathcal{AF}$ and μ will remain unchanged.

To decide what $\widehat{\mathcal{AG}_\mathcal{P}}$ needs to be, one set of arguments, $\mathcal{O}_i \in \mathcal{E}$, can simply be sampled using Pr.

Then, $\widehat{\mathcal{G}}$ can be determined according to the agent type. If the agent is of a type with *Goal* such that $acceptance = true$, then a random $g \in \widehat{Args}$ will be selected as the new goal: $\widehat{\mathcal{G}} \leftarrow g$. Whereas, if is of a type with *Goal* such that $acceptance = false$, then the agent has the aim to fail $\mathcal{AG}_\mathcal{P}$ from achieving its dialogue goal. In this case, $\widehat{\mathcal{G}} = \mathcal{G}$, and contradicting $\mathcal{G}$ will be achieved by editing the domain file used by the planner to reflect this objective.

Finally, $\widehat{\mathcal{AG}_\mathcal{P}}$ needs to be expressed in the form of $\widehat{\mathcal{M}} = \langle \widehat{\mathcal{E}}, \widehat{\Pr} \rangle$. For $\widehat{\mathcal{E}}$, n possible sets of arguments will sampled from a uniform discrete distribution over $2^{Args} \setminus \varnothing$, where $n \geq 1$ is determined by the user. Then, $\widehat{\Pr}$ is a uniform discrete distribution over $\widehat{\mathcal{E}}$, such that

$$p(\widehat{\mathcal{O}}_i) = \frac{1}{n}, \quad i = 1, ..., n, \text{ for each } \widehat{\mathcal{O}}_i \in \widehat{\mathcal{E}}$$

In addition, it is always the case that $\widehat{\mathcal{E}}$ contains an accurate representation of $\widehat{\mathcal{AG}_\mathcal{P}}$. If $\mathcal{AG}_\mathcal{P}$. If $\widehat{\mathcal{AG}_\mathcal{P}}$ is a stochastic agent, then its set of strategy argument, $SArgs$, will be in $\widehat{\mathcal{E}}$. However, or all other proponent types, the accurate representation will be included in the broader form of

Args, for reasons related to their types. For Random agents, having the exact set of arguments they will use will defeat the purpose of this type of agent (see 4.5.2), while for planning agents, this will require generating a strategy first, which may change in the case of a Re-planning agent, or not even exist.

The newly created file (or files) will be used only by the planning opponent, and the real proponent will use the original file to decide its arguments. Also, a new PDDL problem file will be created to take into account the argument that the proponent has asserted in its first turn. This necessity stems from the design of the planning problem, which requires that the proponent will open the dialogue. That is, from the point of view of the planner when searching for a strategy for the planning opponent, the planning opponent is the proponent, and it is impossible to update any argument made by the real proponent otherwise.

4.5 Planning Strategies

The simulator uses an external application [9], which accompanied [6], that translates a given text problem file into a PDDL problem file in accordance with [6], and uses a planner to find an effective strategy for a planning agent. Since planning proponents in their first turn will all have the same strategy (as identical problem files with identical domain files will generate the same strategies, and in the first turn no changes are made to the problem file), and considering that planning for a strategy may be computationally expensive, depending on the problem file, the process can be optimised. Therefore, each strategy is only searched for once and is then used by all planning agents when their strategy is initialised for the first time.

4.5.1 Proponents

Given a problem file, a proponent has a set of possible argument *Args*, from which it picks a subset of arguments to assert in line with its agent type and a dialogue goal such that $\pi_{acceptance} = true$.

Similarly to the way they will select the next argument to assert, stochastic proponents randomly select their set of arguments from *Args*. A naive stochastic proponent will simply sample a subset from a discrete uniform distribution over all subsets of *Args* that contain the goal argument(s), while a rational stochastic agent will randomly pick a subset from discrete uniform distribution over all the conflict free subsets of *Args* that contain the goal argument(s).

A random proponent will select its set of arguments in a similar manner to the stochastic proponent type, except it will sample its subset of arguments from a discrete uniform distribution over $\left(2^{Args} \cup null\right) \setminus \varnothing$. Both stochastic proponents and random opponents will sample one set of arguments to use throughout the dialogue.

Planning agents, on the other hand, will make use of the knowledge they have on the opponent, $\mathcal{M}$. The initial states for a proponent of each of the two types of a planning agent, Simple Planning or Re-planning, is identical. Using a planner and a problem file, a simple planning proponent will generate a strategy by feeding the planner with the problem file, which will result in an ordered list of arguments $\bar{s} \in Args$, given that an effective strategy does exist for the problem. For the Simple Planning proponent, this is the only strategy it will use until the end of the dialogue. The Re-planning agent, on the other hand, may generate several other strategies during a dialogue. In each turn, should the Re-planning proponent assert an argument according to the dialogue rules, and only if the opponent asserted a non-empty argument in its last turn, the Re-planning proponent will generate a new strategy. To do this, it will need to edit the problem file with an updated state of the dialogue. This update needs to include the arguments that each agent asserted.

4.5.2 Opponents

Opponent agents type assert arguments exactly as their proponent equivalent types. However, do differ in the manner in which they obtain their arguments.

Stochastic opponents again select their set of argument randomly, this time by sampling a set of arguments from $\mathcal{E}$ using Pr. Random opponents also pick a random set of arguments. However, unlike other types of opponent agents, random opponents are not restricted by $\mathcal{M}$, but select their set of arguments from $\left(\mathcal{AR} \setminus \left(2^{Args \cup null}\right)\right) \setminus \varnothing$.

Planning opponents will first need to create a new problem file (see 4.4.2) before they can generate a strategy, and can be divided into types according to their goal: a Random Goal Strategy Opponent type or a Contradictory Strategy Opponent. As their names imply, the goal of the first type is simply a random argument from its set of arguments, and the second type employs a Contradictory Strategy. An agent from the second type will need to create a new PDDL domain file to generate its strategy, as described in section 4.4.2. These two planning opponent types can be further distinguished by their ability to use the closure operator, such that a planning opponent is a deriving agent if it is able to derive new arguments using μ. Therefore, a Simple Planning Deriving Opponent may update its strategy if a new argument is obtained from the closure function.

4.6 Simulation

A simulator takes as input a list of dialogue models, and create and run dialogues as prescribed in each dialogue model. Each dialogue model specifies a proponent type, an opponent type, a problem file, number of dialogues to run between the aforementioned proponent and opponent for the given problem file, and a batch number which serves as an identification number of the group of dialogue models with which this instance is affiliated. A batch of Dialogue Models is the result of a Cartesian product on proponent types, opponent types and problem files, where the same number of runs is attached to every combination, as illustrated in table 4.6. In each batch, there is a dialogue model for every proponent type in the batch against every opponent type in the batch. For each batch, opponent type in the batch, problem file in the batch and run, the simulator will create one opponent agent. It will then create a dialogue between the opponent agent against a proponent agent, for each proponent type, as shown in table 4.6. This ensures that a fair comparison of the performance of agents of different types can be made since all proponents play against the same opponents while allowing the user to run multiple combinations at once.

Problem File	Proponent Type	Opponent Type	Number of Runs	Batch Number
Problem1	Type1	Type1	2	1
Problem1	Type2	Type1	2	1
Problem1	Type1	Type2	2	1
Problem1	Type2	Type2	2	1
Problem2	Type1	Type1	1	2
Problem2	Type1	Type2	1	2

Table 4.1: Example: simulator input

Run	Proponent	Opponent	Problem File
1	new Proponent of Type1	Opponent1 ← new Oppoent of Type1	Problem1
2	new Proponent of Type2	Opponent1	Problem1
3	new Proponent of Type1	Opponent2 ← new Oppoent of Type2	Problem1
4	new Proponent of Type2	Opponent2	Problem1
5	new Proponent of Type1	Opponent3 ← new Oppoent of Type1	Problem1
6	new Proponent of Type2	Opponent3	Problem1
7	new Proponent of Type1	Opponent4 ← new Oppoent of Type2	Problem1
8	new Proponent of Type2	Opponent4	Problem1
9	new Proponent of Type1	new Opponent of Type1	Problem2
10	new Proponent of Type1	new Opponent of Type2	Problem2

Table 4.2: Dialogue to simulate given the input in table 4.6

4.7 Reports

The results of all simulated dialogues are exported into a csv report. For each dialogue, the report will record the proponent type, opponent type, arguments asserted by the proponent, arguments asserted by the opponent, whether or not the proroponent type and opponeponent has achieved its goal, the problem file, and if applicable, the strategies found by the planner for the proponent, and the search time and guaranteed probability of success for each strategy.

4.7.1 Optimisation

Sometimes it is impossible or unhelpful to simulate a dialogue for a given problem file, proponent type, and opponent type. In these cases, the simulator will note the details in the report and will skip the relevant runs. There are five possible cases to consider, summarised in table 4.7.1.

The first case occurs when the search for a strategy has timed out after ten minutes, and the strategy of the proponent is empty. In that case, the problem is too big, and all dialogue models related to this problem file will be removed so no dialogues for this problem file will run. The second case occurs when it is impossible for the opponent to win against any agent. That is, it is enough for any proponent to assert their dialogue goal in order to win. As simulating such dialogue does not provide any useful information about the involved agents and will always end with the same result, the simulator will remove any dialogue model with this problem file from the list and will not proceed to run a dialogue, and this will be recorded in the report. An equivalent case from the perspective of the planning agent is the third case, where the planner returns an empty set as strategy and the guaranteed probability of success is 0. As a proponent cannot open a dialogue with an empty argument, this case is handled in the same manner as the previous one. The three remaining cases will all lead to dialogues. The fourth case is when proponent has a probability of guaranteed success of 1 because the planner has found an effective strategy. This means that the opponent could potentially win against another a proponent of a different strategy, but the simple strategy is invincible. The fifth case is when the strategy of the proponent consists only from its goal arguments, that is there is nothing to guarantee that the proponent can win against the opponent. Unlike the second case where the opponent has no chance to win, there are instances where the proponent may win despite having no arguments to defeat its goal. One example is if the opponent is random and the representation the proponent has on the opponent is not accurate. The sixth and last case is when both agents have a good probability to win.

Case	Probability of Guaranteed Success	Proponent Strategy	Run Dialogue?
1	N \A	$\mathcal{S} = \emptyset$	No
2	1	$\mathcal{S} = \mathcal{G}$	No
3	0	$\mathcal{S} = \emptyset$	No
4	1	$\mathcal{S} \neq \emptyset \wedge \mathcal{S} \neq \mathcal{G}$	Yes
5	< 1	$\mathcal{S} = \mathcal{G}$	Yes
6	< 1	$\mathcal{S} \neq \emptyset \wedge \mathcal{S} \neq \mathcal{G}$	Yes

Table 4.3: Summary of possible planner outputs and the corresponding simulator decision on whether to run a dialogue in each case

Chapter 5

Professional and Ethical Issues

5.1 Ethical Issues

Finding effective strategies for the persuasion problem is at the heart of this projects, and its applications can contribute to improve the quality of communication between agents, and even encourage humans to make better decisions and lead a healthier life. Hunter [10] explored how argumentation can be used to change human behaviour in a positive way, for example, dealing with addictions, addressing anti-social behaviour or urging financial responsibility. However, it is easy to see how such applications in the wrong hands can be potentially misused for malevolent purposes. For example, imagine a gambling website that uses a strategy to persuade a user to take bigger chances, or to continue gambling even when they user decides to stop. Other examples can be pressuring vulnerable individuals to harm themselves [2], influencing elections [3] and inciting hate crimes [16]. This raises major concerns that should be taken into account when developing systems with persuasion capabilities.

5.2 British Computing Society Code of Conduct

The rules and professional standards set by the British Computing Society were followed throughout the process of creating this project. The system developed for this project does not provide the functionality to conduct unethical acts such as those discussed above and has due regard “for public health, privacy, security and well-being of others and the environment”[22].

Chapter 6

Evaluation

The performance evaluation criteria can be categorised into two types. The first is concerned with the performance of a simple planning proponent, and how it is affected by different properties of a dialogue and opponents, while the second intends to draw a comparison between the simple planning proponent and proponent of other types, and specifically the re-planning proponent. The data was gathered by running simulations on a selection of proponents, opponents and argumentation frameworks.

In Planning for Persuasion [6], the effectiveness of optimal simple strategies was measured in terms of λ, the probability of the proponent's guaranteed success regardless of the opponent's strategy. Therefore, λ could not be used to evaluate the effectiveness of the optimal simple strategy against different types of opponents, and a new measure was needed; and so, the results presented in this chapter were focused on the probability of success, denoted by θ. The probability of success was calculated by dividing the number of times a proponent embodying a strategy won a simulation of a dialogue under certain settings, by the total number of such dialogues in which they participated. For example, if proponent $\mathcal{P}$ won x out of y dialogues for an argumentation framework $cycle_5$, and against all types of opponents, then the probability of success for $\mathcal{P}$ is $\theta = \dfrac{x}{y}$ for $cycle_5$. Using θ can also help to understand if and how λ can predict success in practice. Then, to have a fair comparison between different types of proponents, each θ will be divided by $\theta*$, which is the maximum value achieved among the proponents. The ratios $\dfrac{\theta}{\theta*}$ of each proponent will be summed to a score, where a higher score indicates a better performance. Having defined θ and a scoring mechanism, the probability of success can now be used to answer several key questions about the performance of the optimal simple strategy.

Black, Coles, and Hampson [6] used 100 randomly selected opponent models of each size

$|\mathcal{E}| = 1, 2, 4, 8$, with uniform probability and no closure operator, in their evaluation. As was pointed out in the background chapter, they found that changing the opponent model size had little effect on the guaranteed success of the optimal simple strategy. In the light of these findings, this chapter will try to answer what impact does the size of the opponent model has on the performance of a planning proponent. Moreover, planning approach was found to be more effective for certain frameworks, such as $ladder_n$, but not as much for others, namely $cycle_n$ frameworks. Experimenting with different frameworks will allow exploring how the variance of λ is reflected in the probability of success, and to cautiously estimate if and when the optimal simple strategy can be useful despite a lower λ. Another significant question this chapter will attempt to answer is how does a proponent's strategy may affect their success against different opponents. As the planning approach does not take into account the opponent's strategy, this question cannot be answered by λ, but only by experimenting by simulating dialogues. Particularly, it will be interesting to examine the behaviour of a planning proponent against a random opponent, since the effectiveness of the strategy can no longer be guaranteed if the opponent uses arguments that are not in the proponent's opponent model. The behaviour in such case can be benchmarked against a re-planning proponent that is able to revise their strategy when new information becomes available.

6.1 Experiment Set Up

Argumentation Frameworks

The example used in the experiments included frameworks of types $cycle_5$, $ladder_5$, and $bipartite_7$ to be consistent with the conditions in Black, Coles and Hampson's [6] evaluation. The sizes of the examples have been selected in consideration of the time it might take to complete each experiment, which is directly affected by the search time for a strategy. This is not so much of a problem for a simple planning proponent against a stochastic agent, for example. However, when re-planning agents are involved, either as proponents, opponents or both, this may increase the overall execution time dramatically. Nonetheless, the problems are large enough to provide a variety of dialogue settings. For each example, 10 opponent models were randomly selected for each opponent model size, $|\mathcal{E}| = 2, 4, 6$, amounting to 30 examples per framework.

In addition, three random frameworks were created, each with 16 arguments, density of 0.25, and opponent model of size 4. Due to the restrictions posed on the opponent model in random graphs described in the Design chapter, the application was struggling to generate graphs with more than 16 arguments or a density greater than 0.25. On the other hand, the planner was struggling with problems that had a density lower than 0.25, where it was forced to terminate after taking more than 10 minutes to find strategies for a problems of density 0.05 and 0.10, and having a reported search time of 0.15. In fact, the average real time it took to find a strategy for a random problem was 2.43 minutes, over 6.75 times higher than that of the other frameworks with $|\mathcal{E}| = 4$, which was 21.6 seconds. It is also important to note that the functionality of randomly selecting opponent models does not ensure an opponent model that can to win. While this is may not a problem for structures such as $cycle_n$, it does pose challenges for randomly generated problem files, as detailed in section 4.4.1. For the reasons outlined above, the density and sized of opponent models and argument sets were fixed, and a smaller selection of random files was used in comparison to other frameworks. In total, 3 different random frameworks were created, and 3 opponent models of size 4 were selected for each problem, where $|\mathcal{E}| = 4$ is the median of the sizes used for the other problems, and it is relatively large to increase the probability that the opponent has some chance to win. Overall, 9 different random problems were used, such that at least 7 out of 9 of the problems had opponent models that were able to win. Assuming that all agents are rational, neither of the random problems had self-attacking arguments. Furthermore, as a functionality to randomly select closure operator(s) was not implemented, the random problems did not have closure operators.

Table 6.1 summarises the files used for the experiments.

$|\mathcal{E}|$ / AF	2	4	6	Total
$cycle_5$	10	10	10	30
$ladder_5$	10	10	10	30
$bipartite_7$	10	10	10	30
$random1$	0	0	3	3
$random2$	0	0	3	3
$random3$	0	0	3	3
Total	30	30	39	99

Table 6.1: Summary of files used for experiments

Agents

To reiterate, λ denotes the probability of guaranteed success for the proponent, no matter what strategy is employed by the opponent, while θ is the actual probability of success observed in experiments.

From the results obtained in Planning for Persuasion 2.4, we can expect $\theta \geq \lambda$ in all cases where the opponent of the simple planning proponent is not random. However, it provides no indication on how θ will vary against different opponents. Moreover, there is no available information about the performance of other proponent strategies that can be used to benchmark the effectiveness the optimal simple strategy. Therefore, for each problem file identified in the previous section, each proponent type was pitted against each opponent type, with the exception of planning deriving agents, as neither of the problems files has a closure operator.

Dialogues

For each combination of a non-random problem file, proponent type and opponent type, 50 dialogues were simulated, such that all dialogues are of the same batch, and therefore each opponent is pitted against one proponent of each type, as described in section 4.6.

Similar setting were used for the random files. However, due to excessive running time, 20 dialogues were simulated for each dialogue model.

As detailed in the optimisation section 6.2.1 in the design and implementation chapter, dialogues in which either the opponent or the proponent can never win due the random selection of arguments will not be executed. Those case are noted in the result tables; a column filled with $0*$ denotes that the proponent could not win, and a column filled with $1*$ denotes that the opponent could not win. The results in these cases were not used for any calculation, and were included in the tables for completeness.

6.2 The effect of argumentation framework structure and opponent model size on the performance of the simple planning proponent

The results of the experiments found that the size of the opponent model had an insignificant effect on the probability of success, for both the simple planning proponent and the re-planning proponent, supporting Black, Coles and Hampson's [6] findings. The average probability of success across all non-random problem files, summarised in table 6.2 for the simple planning proponent, had a range of 0.06, for both simple planning and re-planning proponents. These results were consistent when each problem file was examined individually, with the exception of $cycle_5$, where the range of probability of success was 0.17, see tables A.1 and A.4 in Appendix A. As can be seen in table A.4, no dialogues were simulated for problem files 5, 6 and 8 for optimisation reasons, as the opponent could not win under any circumstances. Therefore, the increase in range could be attributed to the fact that only 7 out of the 10 files for $|\mathcal{E}| = 2$ were taken into account in the calculations, which is a small enough number for outliers to skew the averages.

Problem file	bipartite7			cycle5			ladder5		
$\lvert\mathcal{E}\rvert$	2	4	6	2	4	6	2	4	6
Random	0.57	0.65	0.66	0.29	0.32	0.28	0.64	0.68	0.67
Re-planning (con)	1.00	1.00	1.00	0.99	0.96	0.97	1.00	1.00	1.00
Re-planning (rnd)	1.00	1.00	0.99	0.98	0.94	0.96	1.00	0.99	1.00
Simple Planning (con)	0.87	0.93	0.92	0.89	0.64	0.56	1.00	0.99	1.00
Simple Planning (rnd)	0.88	0.83	0.87	0.79	0.60	0.59	1.00	0.98	1.00
Stochastic	0.57	0.52	0.44	0.75	0.44	0.30	1.00	0.95	0.92
Overall	0.82	0.82	0.81	0.78	0.65	0.61	0.94	0.93	0.93

Table 6.2: Average success for a simple planning proponent

While the probability of success seemed to be in line with previous evaluations [6], the probability of guaranteed success seemed to decrease with the growth of opponent models size, as detailed in table 6.2 . As the average λ is not affected by the number of experiments, but only by the number of problems files since λ has a fixed value for each problem file, this inconsistency may be due to the lower number of different opponent models used in the experiments, 10 for each size $|\mathcal{E}| = 2, 4, 6$, in comparison to 100 used in the original evaluation [6]. It is also clear from the table the probability of success is consistently higher than the probability of guaranteed success as expected, with the exception of $ladder_5$, for $|\mathcal{E}| = 2, 4$; in average, there is an increase of 43 percent for $cycle_5$ and $bipartite_7$. When looking closely at the results for $ladder_5$ by opponent type in tables A.8 and A.7, it is evident that the decrease of average θ is due to the results for random opponent. As λ values for $ladder_5$ were close or equal to 1, any decrease in θ is more pronounced.

The experiments also corroborate the suggestion that the effectiveness of the optimal simple strategy depends on the structure of the argumentation framework [6], and the simple planning proponent was found to be successful for the $ladder_5$ and $bipartite_7$ problems, with a probability of success $\overline{\theta} = 0.93$ and $\overline{\theta} = 0.82$ respectively, but less successful for $cycle_5$ problems with $\overline{\theta} = 0.68$. The simple planning strategy was also found to be effective for random argumentation frameworks, with $\overline{\theta} = 82$ (table A.20).

$\lvert\mathcal{E}\rvert$	**2**			**4**			**6**					
Problem file	θ	λ	**diff**	θ	λ	**diff**	θ	λ	**diff**	$\overline{\theta}$	$\overline{\lambda}$	$\overline{diff}$
bipartite7	0.82	0.57	42.75%	0.82	0.53	56.19%	0.81	0.45	80.64%	0.82	0.52	58.35%
cycle5	0.78	0.75	4.37%	0.65	0.44	48.10%	0.61	0.30	106.52%	0.68	0.49	37.63%
ladder5	0.94	1.00	-5.93%	0.93	0.93	0.72%	0.93	0.90	3.56%	0.93	0.94	-0.73%
Overall	0.86	0.80	7.85%	0.81	0.64	26.11%	0.79	0.56	42.02%	0.82	0.67	23.27%

Table 6.3: Differences between the average probability of success, θ, against all opponent types, and guaranteed probability of success, λ, for $bipartite_7$, $cycle_5$ and $ladder_5$ structures

6.2.1 The Difference in Performance Between the Simple Planning and Re-planning Proponents

Before proceeding to discuss the difference in performance between the simple planning and re-planning proponents, it is important to explain how to read the tables presented in this section, namely table 6.2.1 and table tab:repscore. These tables summarise the tables in the appendix, and show the sum of quality ratios, $\frac{\theta}{\theta*}$, for each problem file of a certain file. For example, the results under $ladder_5$ and opponent size 6, are the sum of quality ratios of all 10 problem files of that structure and opponent model size. The brackets next to each opponent model size indicate how many files were actually taken into account. For example, (6/10) means that only 6 out of 10 files were used in dialogues for optimisation reasons, as the other 4 files were excluded for the reason specified in section (e.g., the opponent can never win against any proponent). It also means that the maximum score that could be achieved for this structure and opponent model size is 6.

It was expected that the re-planning proponent will do the same or outperform the simple planning agent, in particular against the random opponent. The re-planning agent has scored only slightly higher against the random opponent (a difference of 0.03 percent), but a closer look at the results for each argumentation framework structure and opponent model size reveals that the re-planning proponent has outperformed the simple planning proponent in only 5 out of 10 problems, as detailed in tables 6.2.1 and 6.4. A reasonable explanation for this result may lie in the arguments of the Random opponent, which may include the empty argument and lead to a special case where the order of utterance matter. For example, given a random opponent $\mathcal{O}$, with arguments $OArgs = a, b, null$. Now, if two different proponents are participating in a dialogue against $\mathcal{O}$, all that is guaranteed is that $OArgs$ will remain unchanged, but as the random opponent randomly selects their next argument, there is no guarantee that they will assert their argument in the same order in each dialogue. Ignoring the attack relations for now, a possible start of a dialogue against some proponent $\mathcal{P}$ with arguments $PArgs = a$ and $a = \mathcal{G}$, can be $\mathcal{P} : a, \mathcal{O} : null$. At this state, $\mathcal{P}$ is winning, because a is in the grounded extension, and therefore, they return the empty argument: $\mathcal{P} : a, \mathcal{O} : null, \mathcal{P} : null$, and at this point the dialogue will end as both agents asserted the empty argument consecutively. However, against another proponent, it is possible that $\mathcal{O}$ will be able to defeat another proponent if it does not assert the empty argument so early. By examining the raw data, it seems that dialogues similar to the example above are not rare, and considering that there might be similar cases that could not be detected by observation, this might explain the results.

The overall results showed that the simple planning agent had a higher score by almost 0.06 percent. Nonetheless, the overall scores per opponent type reveal that the re-planning proponent performed slightly better against all opponents except for the re-planning opponents, against which the simple planning proponent scored 4.06 percent higher. By inspecting the raw data, it seems that the re-planning opponents behave differently against re-planning proponents. Unfortunately, it might be required to develop further functionalities that allow more insight into the behaviour of the opponent in order to find an explanation. Other than these variables, no other factors, such as argumentation framework or opponent model size, seem to affect the score of the proponents.

Apart from the effectiveness of their strategy, the performance of both proponent types can also be measured in terms of searching time. Table 6.2.1 show that the average accumulated planner search time for the re-planning agent per dialogue, is greater than the average search time per dialogue for a simple planning agent by over 32 percent. Therefore, it appears that the increase in search time for the re-planning proponent is greater than the increase in effectiveness they might provide.

Taking all these findings into account, there is no particular case in which the re-planning proponent outperforms the simple planning proponent with a significant difference in scores. Therefore, if searching time is a concern, perhaps the simple planning proponent should be preferred against the opponents.

6.2.2 The Effect of Proponent Strategy on the Performance Against the Stochastic Opponent

As expected, the simple planning and re-planning proponents outperformed the stochastic and random proponents against the stochastic opponent, with an average score 313.50 percent higher. As can be seen from table 6.7, the difference in effectiveness is particularly evident for

Problem file	*cycle*$_5$			*ladder*$_5$			*bipartite*$_7$			*Random*	**Overall**
Opponent size	**2** (6/10)	**4** (8/10)	**6** (9/10)	**2** (9/10)	**4** (10/10)	**6** (10/10)	**2** (7/10)	**4** (10/10)	**6** (10/10)	**4** (8/9)	(89/99)
Random	5.14	6.97	7.31	8.89	9.85	9.58	6.86	9.58	9.68	7.87	81.73
Re-planning (con)	6.00	8.00	9.00	9.00	10.00	10.00	7.00	10.00	10.00	8.00	87.00
Re-planning (rnd)	6.00	8.00	9.00	9.00	10.00	10.00	7.00	10.00	10.00	8.00	87.00
Simple Planning (con)	6.00	6.43	6.94	9.00	10.00	10.00	7.00	10.00	9.90	8.00	83.27
Simple Planning (rnd)	5.65	6.87	7.34	9.00	10.00	10.00	7.00	10.00	9.78	8.00	83.64
Stochastic	6.00	8.00	8.61	9.00	10.00	10.00	7.00	10.00	10.00	8.00	86.61
Total	34.79	44.27	48.20	53.89	59.85	59.58	41.86	59.58	59.36	47.87	509.25

Table 6.4: Summary of quality scores (sum of quality ratios $\frac{\theta}{\theta *}$) for the simple planning proponent

Problem file	*cycle*$_5$			*ladder*$_5$			*bipartite*$_7$			*Random*	**Overall**
Opponent size	**2** (6/10)	**4** (8/10)	**6** (9/10)	**2** (9/10)	**4** (10/10)	**6** (10/10)	**2** (7/10)	**4** (10/10)	**6** (10/10)	**4** (8/9)	(89/99)
Random	5.65	7.57	10.00	8.33	9.60	9.86	6.13	9.85	9.62	7.18	83.79
Re-planning (con)	4.86	4.82	4.53	9.00	9.86	9.86	4.78	8.10	7.80	6.95	70.56
Re-planning (rnd)	4.76	3.89	3.89	9.00	9.72	9.68	5.70	7.12	8.14	7.14	69.04
Simple Planning (con)	6.00	8.00	9.00	9.00	10.00	10.00	6.28	9.60	9.78	8.00	85.66
Simple Planning (rnd)	6.00	8.00	9.00	8.98	10.00	10.00	6.76	9.69	9.90	8.00	86.33
Stochastic	6.00	8.00	9.00	9.00	10.00	10.00	7.00	9.88	10.00	8.00	86.88
Total	33.27	40.28	45.42	53.31	59.18	59.40	36.65	54.24	55.24	45.27	482.26

Table 6.5: Summary of quality scores (sum of quality ratios $\frac{\theta}{\theta *}$) for the re-planning proponent

the more challenging structures, namely, *cycle*$_5$ and *ladder*$_5$. These results demonstrate that the problem of persuasion, whether for structured frameworks or random ones, is not easily solved by asserting arguments randomly, but require a more sophisticated mechanism. The optimal simple strategy has proven to offer an effective solution, scoring 82 out of 89 possible points.

It can also be observed that the simple planning proponent's score was 0.12 higher than the score of the re-planning proponent. However, this negligible difference could be the result of calculating the values with an accuracy of only two decimal places.

The results also revealed that the stochastic proponent scored higher than the random agent for almost all structures and opponent models size, with the exception of *cycle*$_4$ with $|\mathcal{E}| = 4$, for which both had the same score. The only difference between both proponent types is that the random proponent may also have the empty argument as part of its set of strategy arguments, $SArgs$, which it will use to randomly select arguments from. As outlined in the design and implementation section, an agent will assert the empty argument if $SArgs$ is empty, or if it already achieved its dialogue goal. Therefore, the difference in performance between

	Simple Planning	**Replanning**	**Increase**
bipartite7	0.03	0.05	59.19%
cycle5	5.41	7.27	34.33%
ladder5	1.10	1.46	32.48%
Random1#1	1.85	2.00	8.34%
Random1#3	8.59	12.04	40.23%
Random2#1	1.29	1.51	17.59%
Random2#2	4.97	5.68	14.23%
Random2#3	4.20	5.33	26.77%
Random3#1	2.68	3.32	23.97%
Random3#2	3.25	3.87	19.06%
Random3#3	17.11	18.98	10.97%
Total	2.13	2.82	32.69%

Table 6.6: Planner average search times for simple planning and re-planning proponents

the proponents may imply that asserting the empty argument when it is possible to assert a non-empty argument decrease the effectiveness of the proponent.

Problem file	$cycle_5$			$ladder_5$			$bipartite_7$			*Random*	**Overall**
Opponent size	**2** (6/10)	**4** (8/10)	**6** (9/10)	**2** (9/10)	**4** (10/10)	**6** (10/10)	**2** (7/10)	**4** (10/10)	**6** (10/10)	**4** (8/9)	(89/99)
Random	0.00	0.13	0.00	1.02	1.72	2.60	1.91	2.59	3.02	3.71	16.70
Re-planning	6.00	8.00	8.00	7.00	9.00	9.00	7.00	9.88	10.00	8.00	81.88
Simple Planning	6.00	8.00	8.00	7.00	9.00	9.00	7.00	10.00	10.00	8.00	82.00
Stochastic	0.14	0.13	0.16	1.50	2.48	3.27	2.65	3.67	4.00	4.94	22.93

Table 6.7: Summary of quality scores (sum of quality ratios $\frac{\theta}{\theta *}$) for all proponents against the stochastic opponent

6.3 Software Testing

In addition to evaluation in the form of experiments, the correct execution of the application was ensured with extensive unit testing.

Chapter 7

Conclusion and Future Work

This project was the first known attempt to evaluate the Planning for Persuasion method [6] by using it in real dialogues. The resulting system was carefully tailored to the properties of the planning approach, with different agents each designed to validate previously published theoretical results [6] and new hypotheses.

The optimal simple strategy was found to be consistently effective against the stochastic, simple planning and re-planning opponents alike, for various problem files of different sizes, including challenging structures with cycles in which the proponent may undermine its dialogue goal(s), and randomly generated argumentation frameworks. As anticipated, the optimal simple strategy was not as effective against the random opponent due to the limitation of not taking into account what can be learnt about the opponent from the arguments it asserts throughout the dialogue.

In an attempt to enhance the performance of the optimal simple strategy against an unpredictable opponent, a new type of planning agent, re-planning, was created to emulate some sort of a dynamic policy, where at each state the proponent's strategy is re-evaluated with consideration of the current state of the dialogue. The results of the experiments were not significant enough to rule that the re-planning proponent had outperformed the simple planning proponent. As explained in the evaluation chapter, it is likely that the reason for that was the random agent's ability to assert the empty arguments and as such to terminate dialogues unexpectedly. More conclusive results could be achieved in future work by closer inspection of opponents behaviour, or by implementing new types of unpredictable opponents.

Moreover, this project considered one type of re-planning proponent. However, new versions of the re-planning proponent could be introduced to cover possible ways in which the agent can update its opponent model given new information. For example, it could disregard any previous knowledge once it learns its beliefs on the opponent are inaccurate.

References

[1] Leila Amgoud, Nicolas Maudet, and Simon Parsons. Modelling dialogues using argumentation. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 31–38. IEEE, 2000.

[2] BBC. Blue whale: Should you be worried about online pressure groups?, April 2017. [Online; posted 27-April-2017].

[3] BBC. Cambridge analytica files spell out election tactics, March 2018. [Online; posted 29-March-2018].

[4] Elizabeth Black and Katie Bentley. An empirical study of a deliberation dialogue system. In *International Workshop on Theorie and Applications of Formal Argumentation*, pages 132–146. Springer, 2011.

[5] Walton D. Argumentation theory: A very short introduction. In G.R. Simari and I. Rahwan, editors, *Argumentation in Artificial Intelligence*, chapter 1, pages 1–22. Springer, Boston, MA, 2009.

[6] Black E., Coles A.J., and C. Hampson. Planning for persuasion. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 933–942. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[7] Hadoux E., Beynier A., Maudet N., Weng P., and Hunter A. Optimization of probabilistic argumentation with markov decision models. In *IJCAI*, pages 2004–2010. International Joint Conferences on Artificial Intelligence, 2015.

[8] Emmanuel Hadoux and Anthony Hunter. Strategic sequences of arguments for persuasion using decision trees. In *AAAI*, pages 1128–1134, 2017.

[9] Christopher Hampson. Simple optimal strategies for persuasion. `https://github.com/christopher-hampson/argstrat`, 2017.

[10] Anthony Hunter. Opportunities for argument-centric persuasion in behaviour change. In *European Workshop on Logics in Artificial Intelligence*, pages 48–61. Springer, 2014.

[11] Rahwan I. Guest editorial: Argumentation in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 11(2):115–125, Sep 2005.

[12] Rahwan I., Larson K., and Tohmé F.A. A characterisation of strategy-proofness for grounded argumentation semantics. In *IJCAI*, pages 251–256, 2009.

[13] Wooldridge M. *An Introduction to MultiAgent Systems.* Wiley Publishing, 2nd edition, 2009.

[14] Michael J Maher. Complexity of exploiting privacy violations in strategic argumentation. In *Pacific Rim International Conference on Artificial Intelligence*, pages 523–535. Springer, 2014.

[15] Bernard Moulin, Hengameh Irandoust, Micheline Bélanger, and Gaëlle Desbordes. Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review*, 17(3):169–222, 2002.

[16] Karsten Müller and Carlo Schwarz. Fanning the flames of hate: Social media and hate crime. 2017.

[17] Maudet N., Parsons S., and Rahwan I. Argumentation in multi-agent systems: Context and recent developments. In N. Maudet, S. Parsons, and I. Rahwan, editors, *Argumentation in Multi-Agent Systems*, pages 1–16, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[18] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[19] Minh Dung P. 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.

[20] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[21] Henry Prakken. Formal systems for persuasion dialogue. *The knowledge engineering review*, 21(2):163–188, 2006.

[22] the British Computer Society. Bcs code of condut, June 2015. [Online; posted 3-June-2015].

[23] Matthias Thimm. The tweety library collection for logical aspects of artificial intelligence and knowledge representation. *KI-Künstliche Intelligenz*, 31(1):93–97, 2017.

[24] Bench-Capon T.J.M. and Dunne P.E. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10):619–641, 2007.