Heras Barberá, SM.; Jordan Prunera, JM.; Botti V.; Julian Inglada, VJ. (2013). Argue to agree: A case-based argumentation approach. International Journal of Approximate Reasoning. 54(1):82-108. doi:10.1016/j.ijar.2012.06.005.

Argue to Agree: A Case-Based Argumentation Approach

Stella Heras*, Jaume Jordán, Vicente Botti, Vicente Julián

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera sn 46022 Valencia (Spain)

Abstract

The capability of reaching agreements is a necessary feature that large computer systems where agents interoperate must include. In these systems, agents represent self-motivated entities that have a social context, including dependency relations among them, and different preferences and beliefs. Without agreement there is no cooperation and thus, complex tasks which require the interaction of agents with different points of view cannot be performed. In this work, we propose a case-based argumentation approach for Multi-Agent Systems where agents reach agreements by arguing and improve their argumentation skills from experience. A set of knowledge resources and a reasoning process that agents can use to manage their positions and arguments are presented. These elements are implemented and validated in a customer support application.

Keywords: Argumentation, Agreement Technologies, Case-Based Reasoning, Multi-Agent Systems

1. Introduction

Currently, most business transactions are operated by means of software components in large scale computer systems. These components are often implemented as agents in a Multi-Agent System (MAS). In this way, their reasoning autonomy, their social features and the fact that they can represent human interests and preferences are highlighted. In MAS, agents perform complex tasks that require different levels of intelligence and give rise to interactions among them. From these interactions, conflicts of interest and opinion can arise among different agents, specially when MAS become open with heterogeneous agents dynamically entering in or leaving the system). Therefore, software agents willing to participate in open systems will require to include extra capabilities to explicitly represent and generate *agreements*, on top of the simpler capacity to interoperate [1]. This paves the way for a new area of research to develop software components and techniques that allow agents to reach agreements in these systems.

Argumentation is a natural way of reaching agreements between several parties with opposing positions about a particular issue. Agents can reach agreements by engaging in argumentation dialogues with their opponents. This has made Artificial Intelligence (AI) researchers to pay their attention on argumentation theory [2, 3, 4]. However, some work in the area assumes human users interacting with software tools, such as the approaches for argument authoring and diagramming [5], OVA[1]. Other research works where the computational modelling of arguments has been studied are those on case-based argumentation. From the first uses of argumentation in AI, arguments and cases are intertwined [6]. Case-based argumentation particularly reported successful applications in American common law [7], whose judicial standard orders that similar cases must be resolved with similar verdicts. In [8] a model of legal reasoning with cases is proposed. But, again, this model assumed human-computer interaction and cases were not thought to be only acceded by software agents. Case-Based Reasoning (CBR) systems [9] allow agents to learn from their experiences. In MAS, the research in case-based argumentation is quite recent with just a few proposals to date [10]. These proposals are highly domain-specific (e.g. persuasion [11], sensor networks [12] and classification [13]) or centralise the argumentation functionality either in a *mediator* agent, which manages the dialogue between the agents of the system [14], or in a specific module of the system itself

*Corresponding author

[1] `OVA at ARG:dundee: www.arg.dundee.ac.uk`

[15]. Thus, the notions of argument and argumentation resources of these systems are not conceived for being acceded only by software agents that perform automatic reasoning processes over them and that can have partial knowledge about the domain.

In addition, agents can form societies that impose *social dependencies* among them and can have *values* that they want to promote or demote. These values can be personal goods (e.g. efficiency, accuracy, etc.) or also social goods inherited from the agents' dependency relations. Thus, we endorse the view of value-based argumentation frameworks [16, 17, 8, 18], which stress the importance of the audience (the set of individuals that share a particular preference order over the set of values) in determining whether an argument is persuasive or not. Value-based argumentation frameworks extend abstract argumentation frameworks by addressing issues about the rational justification of choices. Preferences over values can determine the reasons that lead an agent to propose a specific argument or to accept or refuse an argument from another agent. To illustrate this statement, let us propose the following running example, which is based on the evaluation scenario presented in Section 4. We have a MAS that controls the operation of a call centre that provides customer support. In this centre, agents represent technicians that must reach agreements about the best solution to apply to different types of software or hardware problems reported to the centre. To solve each problem, a group of technicians engage in an persuasion dialogue proposing their individual solutions and justifying the reasons that they have to propose them. Therefore, each agent argues and interchanges arguments with the other agents to *persuade* them to accept its solution as the best way of solving the problem. In this scenario, an agent representing a technician that wants to promote *efficiency* may prefer to offer quick solutions to the problem without taking into account their quality (e.g. if the problem reports an error on a computer that reboots automatically at random, this technician may prefer to propose a complete reinstallation of the OS rather than wasting time finding out the actual cause of the problem). However, let us now assume that the same agent belongs to a group that prefers to promote *accuracy* over *efficiency* and that imposes its value preferences on its members. In this case, if the agent is also able to generate an alternative solution that promotes *accuracy* (e.g. running a test to detect a possible virus and, if the system is infected, install a tool to clean it), the agent would propose that solution rather than the one that promotes its individual preferences. Therefore, we also consider values as an important element of the social context of an agent.

Starting from the idea that the *social context* of agents determines the way in which they can argue and reach agreements, it should have a decisive influence on the computational representation of arguments and on the argument management process. This opens a new research challenge for the development of a computational mechanism for reaching agreements in MAS in which the participating software agents are able to manage and exchange arguments between themselves taking into account their social context. To deal with this challenge the reasoning process by which agents can automatically generate, select and evaluate arguments must be specified. Also, to allow agents to learn from argumentation experiences and thus, to improve the efficiency and outcomes of further agreement processes is an interesting add-on feature for this type of systems.

In this paper, we propose a reasoning process that agents of a society can follow to engage in argumentation dialogues to reach agreements. This reasoning process makes use of the Case-Based Reasoning (CBR) methodology [9] as an appropriate technique to manage arguments and enhance agreement processes. Reasoning with cases is specially suitable when there is a weak domain theory, but acquiring examples encountered in practice is easy. Most argumentation systems produce arguments and reason about them by applying a set of inference rules (e.g. [19, 20]). Rule-based systems require to elicit a explicit model of the domain. In open MAS, where agents can enter or leave the system, the domain is highly dynamic and the set of rules that model it is difficult to specify in advance. However, tracking the arguments that agents put forward in argumentation processes could be relatively simple. Therefore, these arguments can be stored as cases codified in a specific case representation language that different agents are able to understand. This is easier than creating an explicit domain model, as it is possible to develop case-bases avoiding the knowledge-acquisition bottleneck. With these case-bases, agents are able to perform *lazy learning* processes over argumentation information, as will be illustrated in the example of Section 3. Another important problem with rule-based systems arises when the knowledge-base must be updated (e.g. adding new knowledge that can invalidate the validity of a rule). Updates imply to check the knowledge-base for conflicting or redundant rules. Case-based systems are easier to maintain than rule-based systems since, in the worst case, the addition of new cases can give rise to updates in some previous cases, but does affect the correct operation of the system, although it can have an impact

in its performance. Hence, a case-based representation of the domain knowledge of the system is more suitable for being applied in dynamic open MAS.

To allow agents to reason over arguments and the argumentation process itself, they must be able to generate, select and evaluate arguments. Therefore, we propose a set of case-based knowledge resources that agents can use to perform these tasks. In addition, we present an argumentation ontology to represent the elements of these knowledge resources. This ontology enables a common understanding about the argumentation concepts and allows heterogeneous agents to engage in agreement processes. Based on the knowledge resources proposed, we present the reasoning process that agents follow to reach agreements taking into account their argumentation experience and social context. Also, this process allows agents to enhance their argumentation skills by learning from past argumentation experiences. All these elements form a case-based argumentation framework for reaching agreements in agent societies. This framework has been implemented and tested in a call centre that provides customer support.

This paper is structured as follows: Section 2 introduces our proposed argumentation framework for agent societies; Section 3 presents the case-based reasoning process that agents follow to engage in argumentation processes; in Section 4, the reasoning process is evaluated in a customer support application; Section 5 provides a discussion about some issues and assumptions made in this work and compares it with related approaches; finally, Section 6 summarises the contributions of this paper.

2. Case-Based Argumentation Framework

In open multi-agent argumentation systems the arguments that an agent generates to support its position can conflict with arguments of other agents and these conflicts are solved by means of argumentation dialogues between them. In [21, Chapter 3] we presented a computational case-based argumentation framework that agents can use to manage argumentation processes. This section outlines the main components of this framework. Thus, the framework consists of:

- A definition for the notion of arguments that agents use to justify their proposals.
- The logical language that agents use to represent knowledge and engage in argumentation processes.
- A definition for the concept of conflict between arguments, which specifies when an argument attacks other different argument.
- A definition for the concept of defeat between arguments, which specifies which attacks over arguments succeed.
- A definition for the possible acceptability status of an argument during the dialogue, which classifies it as *acceptable* (not defeated by other argument), *unacceptable* (defeated) or *undecided* (there is not enough information to decide its acceptability status) in view of its relations with other arguments [22].

In addition, we proposed three types of knowledge resources that the agents can use to generate, select and evaluate arguments by using our framework:

A case-base with domain-cases that represent previous problems and their solutions. Agents can use this knowledge resource to generate their positions in a dialogue and arguments to support them. Therefore, the *position* of an agent represents the *solution* that this agent proposes and both terms are used interchangeably in this paper. Also, the acquisition of new domain-cases increases the knowledge of agents about the domain under discussion.

A case-base with argument-cases that store previous argumentation experiences and their final outcome. Argument-cases have three main objectives: they can be used by agents 1) to generate new arguments; 2) to select the best position to put forward in view of past argumentation experiences; and 3) to store the new argumentation knowledge gained in each agreement process, improving the agents' argumentation skills.

A database of argumentation schemes with a set of schemes with the structure proposed in [23], which represent stereotyped patterns of common reasoning in the application domain where the framework is implemented. An argumentation scheme consists of a set of premises and a conclusion that is presumed to follow from them. Also, each argumentation scheme has associated a set of *critical questions* that represent potential attacks to the conclusion supported by the scheme. The concrete argumentation schemes to be used depend on the application domain.

In our proposal, arguments that agents interchange are tuples of the form:

Definition 2.1 (Argument)**.** *$Arg = \{\phi, v, < S >\}$, where ϕ is the conclusion of the argument, v is the value that the agent wants to promote and $< S >$ is a set of elements that justify the argument (the support set).*

The support set S can consist of different elements, depending on the argument purpose. On one hand, if the argument justifies a potential solution for a problem, the support set is the set of features (*premises*) that represent the context of the domain where the argument has been put forward (those premises that match the problem to solve and other extra premises that do not appear in the description of this problem but that have been also considered to draw the conclusion of the argument) and optionally, any knowledge resource used by the proponent to generate the argument (*domain-cases*, *argument-cases* and *argumentation schemes*). This type of argument is called a *support argument*. On the other hand, if the argument attacks the argument of an opponent, the support set can also include any of the allowed attack elements of our framework. These are: *distinguishing premises*, *counter-examples* or *critical questions*, as proposed in [8]. This other type of argument is called an *attack argument*. Then, the support set consists of the following tuple of sets of support elements [2]:

Definition 2.2 (Support Set)**.** *A support set for an argument consists of a set of elements: $S = <$ {premises}, {domainCases}, {argumentCases}, {argumentationSchemes}, {criticalQuestions}, {distinguishingPremises}, {counterExamples} $>$*

2.1. Domain-cases

The structure of domain-cases that an argumentation system that implements our framework has depends on the application domain. As example, Figure 1 illustrates the structure of a possible domain-case in the call centre example. Here, when a technician of the call centre, say *operator 1 (O1)*, is presented with the problem of a HP computer with Windows XP installed that reboots at random, he searches its domain-cases case-base to generate a possible solution for that problem. In this example, *O1* has found the domain-case *DC1* that matches de description of the problem to solve, also including an extra feature (the specification of the computer model). Note that in the call centre example we assume that domain-cases also store the value promoted by the solution that they represent. This is a domain-dependent design decision. In other different application scenarios of our framework we may make different assumptions, for instance, that all positions generated by an agent promote its most preferred value.

2.2. Argumentation schemes

The concrete set of argumentation schemes used also depends on the application domain of our argumentation framework. For instance, in the call centre example the following argumentation scheme (based on Walton's *Appleal to Expert Opinion* scheme [23]) represents the general pattern of reasoning that gives credence to the opinion of an experienced operator, say an *expert*. Following this scheme, if a technician is considered as expert in solving a specific type of problems, his solution should be taken as the most appropriate solution to apply. However, as any Walton-like argumentation scheme, this scheme includes critical questions that represent potential attacks to the conclusion drawn from the scheme. Thus, in our example the position of technician T (proposing solution S) may be rebutted if another technician shows evidence that other different expert has proposed an alternative solution.

[2] This representation is only used for illustrative purposes and efficiency considerations about the implementation are obviated.

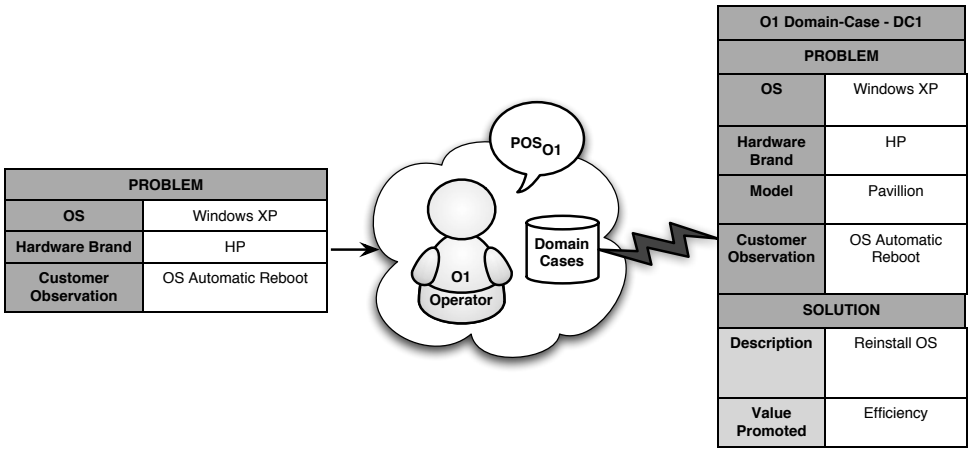

Figure 1: Example Structure of a Domain-Case

ARGUMENTATION SCHEME

Major Premise: Technician T is an expert in solving software problems on HP computers
Minor Premise: T says that S is the best solution to apply for solving a software problem P on a HP computer
Conclusion: S is the best solution to apply for solving the software problem P

Critical Questions
Consistency Question: Is S consistent with the solutions proposed by other experts?
...

2.3. Type of Attacks

As pointed out before, there are three types of elements that agents can use to attack arguments in our argumentation framework: *distinguishing premises*, *counter-examples* or *critical questions*. The following functions provide a formal definition for these elements. First, let us assume that we have a problem to solve denoted as P, a set of cases denoted as $C = \{c_1, c_2, ..., c_n\}$, a set of premises denoted as $F_i = \{f_1, f_2, ..., f_m\}$ such that $f_j \in P$ represents a premise that describes the problem P and $f_i \in c_i$ represents a premise that describes the case c_i ($f_i, f_j \in F_i$, thus, we represent both problems and the problem description of cases with a set of premises), a function $value_{c_i}(f_i)$ that returns the value of a premise in a case $c_i \in C$ (i.e. the actual data of that premise, do not confuse with the notion of value promoted by arguments), a function *acceptable(c_i)* that returns *true* if the case c_i was deemed acceptable at the end of the argumentation dialogue, and a function *conclusion(c_i)* that returns the conclusion of the case c_i (i.e. the solution promoted by the case). Also, we have a function that defines a *match* between two cases and a function that defines a *subsumption* between two cases.

Definition 2.3 (Match)**.** *A match between two cases $c_i, c_j \in C$ is defined as: $match(c_i, c_j) : C \times C \rightarrow true$ iff $F_i \cap F_j \neq \emptyset$ and $\forall f \in F_i \cap F_j,\ value_{c_i}(f) = value_{c_j}(f)$.*

Hence, two cases match if their common features match. Note that this does not mean that both cases have the same features, since one of them can have extra features that do not appear in the other case. In this case, we say that the case with the extra features subsumes the other case.

Definition 2.4 (Subsumption)**.** *A case c_i subsumes other case c_j (from a set of cases C): $subsumes(c_i, c_j) : C \times C \rightarrow true$ iff $match(c_i, c_j)$ and $\forall f_i \in c_i, \exists f_j \in c_j / value_{c_i}(f_i) = value_{c_j}(f_j)$.*

Therefore, we also describe problems as cases without solution and assume that a match between the problem to solve and a stored case means that the latter has some features of the problem and with the

same values[3]. A total match between a problem and a case or between two cases means that both cases have the same features and with the same values. Then, the attack elements of our framework are defined as follows.

Definition 2.5 (Distinguishing Premise)**.** *A distinguishing premise f_i with respect to a problem P between two cases $c_1, c_2 \in C$ is defined as: $\exists f_i \in c_1 \wedge \nexists f_i \in P / \exists f_i \in c_2 \wedge value_{c_1}(f_i) \neq value_{c_2}(f_i)$ or else, $\exists f_i \in c_1 \wedge \exists f_i \in P / value_{c_1}(f_i) = value_P(f_i) \wedge \nexists f_i \in c_2$*

That is a premise that does not appear in the description of the problem to solve and has different values for two cases or a premise that appears in the problem description and does not appear in one of the cases. For instance, the premise *"Model"* would be a distinguishing premise between the domain-case shown in Figure 1 and another domain-case, say *DC2*, that has exactly the same problem description than *DC1*, but that stores different data for this premise (a different model for the HP computer).

Definition 2.6 (Counter-Example)**.** *A counter-example for a case $c_1 \in C$ with respect to a problem P is another case $c_2 \in C$ such that: $acceptable(c_2) \wedge \forall f_i \in c_2 \cap P / value_{c_2}(f_i) = value_P(f_i) \wedge \forall f_i \in c_1 / (\exists f_i \in c_2 \wedge value_{f_i}(c_2) = value_{f_i}(c_1)) \wedge conclusion(c_2) \neq conclusion(c_1)$*

That is, a counter-example for a case is a previous case (i.e. a domain-case or an argument-case that was deemed acceptable), where the problem description of the counter-example matches the current problem to solve and also subsumes the problem description of the case, but proposing a different solution. To illustrate this concept, let us assume that the domain-case *DC2*, which has the same problem description than *DC1*, proposes an alternative solution (e.g "Check for Memory Errors") that promotes the value *accuracy*. Therefore, *DC2* could be used to generate a counter-example attack to *DC1* and vice-versa.

Finally, as explained before in this section, a *critical question* is a question associated to an argumentation scheme that represents a potential way in which the conclusion drawn from the scheme can be attacked. Therefore, if the opponent asks a critical question, the argument that supports this argumentation scheme remains temporally rebutted until the question is conveniently answered. This characteristic of argumentation schemes makes them very suitable to devise ways of attack the conclusions drawn from other agents.

2.4. Argument-cases

Argument-cases are the main structure that we use to computationally represent arguments in agent societies. Also, their structure is generic and domain-independent. Argument-cases store the information about a previous argument (or a set of similar arguments) that an agent posed in certain step of a dialogue with other agents. To illustrate the components of an argument-case, let us now assume that the operator *O1* has generated a support argument *SA1* to persuade another operator *O2* to accept his position. Recalling, this position proposes a solution for a software problem on a HP computer that reboots automatically at random. The solution proposed by *O1* was generated by using the domain-case *DC1* and consist on a reinstallation of the operative system, which promotes the same value than *DC1* promotes, say *efficiency*. If $p_1 = \{"OS" = Windows\ XP\}$, $p_2 = \{"Hardware\ Brand" = HP\}$, $p_3 = \{"Model" = Pavillion\}$ and $p_4 = \{"Customer\ Observation" = OS\ Automatic\ Reboot\}$, then:

$$SA1 = \{"Reinstall", Efficiency, < \{p_1, p_2, p_3, p_4\}, \{DC1\}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-} >\}$$

Figure 2 illustrates an example of an argument-case that stores the information of the support argument $SA1$ in the call centre domain. Argument-cases have the three possible types of components that usual cases of CBR systems have: the description of the state of the world when the case was stored (*Problem*); the solution of the case (*Conclusion*); and the explanation of the process that gave rise to this conclusion (*Justification*).

The *problem* description has a *domain context* that consists of the *premises* that characterise the argument. In addition, if we want to store an argument and use it to generate a persuasive argument

[3]Different types of matches could define other types of similarity between cases. For instance, a different match function could establish the threshold under which two features can be considered as similar or when a feature subsumes other feature in a hierarchy (and hence the more specific feature could be considered as a matching feature).

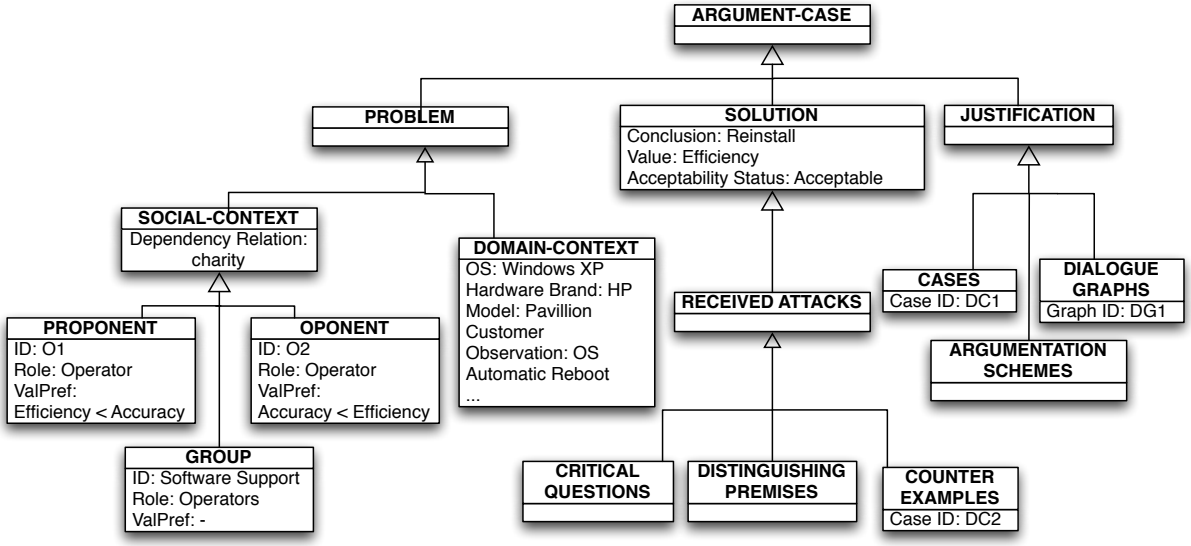

Figure 2: Example Structure of an Argument-Case

in the future, the features that characterise its *social context* must also be kept. The social context of the argument-case includes information about the *proponent* and the *opponent* of the argument and about their *group*. Moreover, we also store the preferences (*ValPref*) of each agent or group over the set of *values* pre-defined in the system. Finally, the *dependency relation* between the proponent's and the opponent's roles is also stored. In this work, we consider three types of dependency relations as defined in [24]: *Power*, when an agent has to accept a request from other agent because of some pre-defined domination relationship between them; and *Charity*, when an agent is willing to answer a request from other agent without being obliged to do so. For instance, in the argument-case of Figure 2 proponent and opponent play the same role (operators) and hence, we have set a charity dependency relation between them. Also, both operators belong to the same group, which provides *software support* and does not impose any specific value preference order over its members in this case.

In the *solution* part, the *conclusion* of the case, the *value* promoted, and the *acceptability status* of the argument at the end of the dialogue are stored. The last feature shows if the argument was deemed *acceptable*, *unacceptable* or *undecided* in view of the other arguments that were put forward in the agreement process. Thus, the conclusion of the argument-case in Figure 2 represents the solution proposed by the operator *O1* when it was presented with the automatic reboot problem on the HP computer. The argument-case promotes the same value than the solution proposed (*efficiency*). In addition, the conclusion part includes information about the possible *attacks* that the argument received during the process. These attacks could represent the justification for an argument to be deemed unacceptable or else reinforce the persuasive power of an argument that, despite being attacked, was finally accepted. As explained before, arguments in our framework can be attacked by putting forward *distinguishing premises*, *counter-examples* or *critical questions* to them.

In the example of Figure 2, we can see that the opponent, say operator *O2*, attacked the argument represented by this argument-case with the counter-example *DC2*. Therefore, *O2* generated an attack argument *AA2* with this counter-example in the support set:

$$AA1 = \{\text{"Check for Memory Errors"}, Accuracy, < \{p_1, p_2, p_3, p_4\}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \{DC2\} >\}$$

However, as illustrated in Figure 2, *O1*'s support argument remained acceptable at the end of the dialogue (when the argument-case that represents it was retained in the argument-cases case-base). Attacked arguments remain acceptable if the proponent of the argument is able to rebut the attack received, or if the opponent that put forward the attack withdraws it.

The *justification* part of an argument-case stores the information about the knowledge resources that were used to generate the argument represented by the argument-case (the set of domain-cases and

argument-cases). In the example of Figure 2, the justification part of the argument-case includes the domain-case *DC1*, that the operator *O1* used to generate his position. In addition, the justification of each argument-case has a *dialogue-graph* (or several) associated, which represents the dialogue where the argument was put forward (*DG1* in Figure 2). In this way, the sequence of arguments that were put forward in a dialogue is represented, storing the complete conversation as a directed graph that links argument-cases. This graph can be used later to improve the efficiency in an argumentation dialogue, as explained in the following section.

3. Reasoning Process

This section presents the reasoning process that allows agents to use the knowledge resources presented before to generate, select and evaluate their positions and arguments. Also, with this process, agents are able to automatically learn from the argumentation experiences. The reasoning process consists of the following subprocesses:

- Position Management. By this process agents are able to:
  - Generate their positions.
  - Select the best position to propose.
  - Evaluate their positions in view of other different positions.
- Argument Management. By this process agents are able to:
  - Generate support and attack arguments for arguments of other different agents.
  - Select the best argument to put forward in each step of the argumentation dialogue.
  - Evaluate their arguments in view of other different arguments.

Thus, as first step of the reasoning process, agents generate their potential positions (points of view) and select the best one to propose. At this point, agents can either defend their positions if they are attacked or else, attack other different positions proposed by their partners. To defend their positions, agents have to evaluate the attack arguments received and generate and select the best support argument to propose. To attack different positions, agents have to ask the agent that they want to attack for providing them with a support argument for the position to attack. Then, agents can generate and select the best argument to attack the support argument provided. The following sections explain these subprocesses in detail. Along them, we assume that a set of agents with different positions are arguing to reach an agreement to solve a complex problem. At this level of abstraction, we assume that this is a generic problem of any type (e.g. resource allocation, classification, prediction, etc.) that could be described with a set of features.

3.1. Position Management

In the first step to reach an agreement about the best solution for a problem to solve, an agent can generate its individual position, which represents the best solution for the problem from the agent's point of view. This is not a compulsory step for agents to engage in the argumentation process to reach the agreement, since some agents could argue about positions of others' without having necessarily generated their own. Then, with the set of generated positions agents generate associated argument-cases to support them. After that, they can use their case-bases of argument-cases to select the best position to propose. Finally, agents evaluate their positions in view of the others' positions and their previous experience.

3.1.1. Position Generation.

In our argumentation framework, agents have several ways to generate positions, depending on their design or even on strategical considerations. Thus, an agent could follow different mechanisms to generate positions:

1. **From the Problem Description and Domain-Cases:** This option would be followed by agents that rely more on their experiences. The agent retrieves from the domain case-base those cases that match with the specification of the current problem. With the solutions that were applied in these cases, the agent generates a potential solution for the problem at hand, which represents its position with respect to the problem. Note that the set of retrieved cases could provide different solutions for the same problem.
2. **From the Problem Description and Argumentation Schemes:** This option would be followed by agents that rely more on its pre-defined argumentation schemes. Then, an agent can generate its position as the conclusion drawn by using the problem description to match (totally or partially) the premises of a scheme.
3. **From the Problem Description, Cases and Argumentation Schemes:** This option would be selected by agents that prefer to exploit all their resources and follow an hybrid generation policy. The agent retrieves from its domain case-base the cases that match the current problem. Then, it can extend the problem description by adding the set of attributes of the retrieved cases that are consistent with this description (do not appear in the problem description) and with the set of retrieved cases (have the same value in all retrieved cases that they appear). Finally, from this new problem description and the argumentation schemes the agent can generate its position (or positions), as explained above. This method for generating positions follows the idea of broadening the space of possible solutions by considering features that, despite not being specified in the current problem, have been observed in similar problems in the past. Note that only the cases that match the problem description are retrieved and hence, the space of potential positions could not be extended if we only consider positions generated from cases. However, the extended problem description could add a new feature that makes the description to match an argumentation scheme that was not considered before.

Algorithm 1 presents the pseudocode of the process to generate positions from domain-cases, argumentation schemes or both of them. In the algorithm, *DomainCasesCB* represents the case-base of domain-cases and *ArgumentationSchemesOnt* represents the ontology of argumentation schemes. If the generation method to follow is "D", the algorithm generates positions from the case-base of domain-cases. In case that the method is "S", the algorithm generates positions from the ontology of argumentation schemes. Similarly, if the method to follow is "M", the algorithm generates positions from both the case-base of domain-cases and the ontology of argumentation schemes.

Also, *computeSimilarity* is a domain-dependent function that computes the similarity between a current problem and the description of the domain-cases or argumentation schemes stored in the knowledge resources of the system. This similarity degree is stored for each potential position to solve the problem (when the similarity degree exceeds a pre-defined threshold). *generateSolutions* is a domain-dependent function that generates potential solutions for the problem to solve from the solutions of the domain-cases that are deemed similar to the current problem or the conclusions of the similar argumentation schemes. *addPosition* is a function that adds a new position to the list of potential solutions for the problem to solve. *aggregateDescriptions* is a function that adds to the problem description the extra consistent features that are found in the problem description of the similar domain-cases. Finally, *addGenerationMethod* is a function that forces the algorithm to execute a specific generation method.

3.1.2. Position Selection.

With the set of potential positions, the agent has to decide the one it will propose first. The first step for this selection is to order the positions in subsets, taking into account the value that promotes each position. Thus, the agent will assign to each subset a *Suitability Level (SL)*. Positions that promote the agent's most preferred value will be labelled with suitability level 1, positions that promote the second most preferred value will be labelled with level 2 and so on. Then, positions will be ordered within each level by its *Similarity Degree (SimD)* with the problem to solve, computed by using a domain-dependent similarity measure.

After that, the agent will use the argumentation knowledge stored in its argument-cases case-base. For each position generated, the agent creates an argument-case that represents this position. Thus, it fills the argument-case structure with the available information for each element (the domain and social contexts, the initial acceptability status set to *undecided* and the cases that form part of the justification). Note that, depending on the actual problem to solve and the domain application, the proponent agent

could not know some data about its opponent (e.g. in persuasion dialogues agents are usually unwilling to share information about its values or preferences with other agents).

Then, the agent compares the argument-case created for each position with its case-base of argument-cases and retrieves the sets arg of argument-cases that match the argument-case associated to each position. In this way, the agent can assign to each position a *Support Factor (SF)* from the argumentation perspective and decide which argument-case (and thus, which position) is most suitable to propose in view of its past argumentation experience and its current social context. We consider the parameters shown in the list below as criteria for making such decision.

In the equations, $argC$ is the subset of argument-cases from the whole set arg stored in the agent case-base with the same problem description and conclusion as the current argument; $argAccC$ are those in $argC$ that were deemed acceptable at the end of the dialogue; $argAccCAtt$ are those in $argAccC$ that were attacked; $minAtt$ and $maxAtt$ are the minimum and maximum number of attacks received by any argument-case in $argAccCAtt$; $minS$ and $maxS$ are the minimum and maximum number of steps from any argument-case in $argC$ to the last node of its dialogue graph and; $minKr$ and $maxKr$ are the minimum and maximum number of knowledge resources used to generate any argument-case in $argC$. Also, we assume that the modifier # represents the number of elements of a set.

- **Persuasiveness Degree (PD):** is a value that represents the expected persuasive power of an argument by checking how persuasive an argument-case with the same problem description and conclusion was in the past. To compute this degree, the number $\#argAccC$ of argument-cases that were deemed acceptable out of the total number of retrieved argument-cases $\#argC$ with the same problem description and conclusion as the current argument is calculated:

$$PD = \begin{cases} 0, & if\ \#argC = \emptyset \\ \dfrac{\#argAccC}{\#argC}, & otherwise \end{cases} \tag{1}$$

  with $\#argAccC$, $\#argC \in \mathbb{N}$ and $PD \in [0, 1]$, from less to more persuasive power.

- **Support Degree (SD):** is a value that provides an estimation of the probability that the conclusion of the current argument was acceptable at the end of the dialogue. It is based on the number of argument-cases $\#argAccC$ with the same problem description and conclusion that where deemed acceptable out of the total number of argument-cases arg retrieved.

$$SD = \begin{cases} 0, & if\ \#arg = \emptyset \\ \dfrac{\#argAccC}{\#arg}, & otherwise \end{cases} \tag{2}$$

  with $\#argAccC, \#arg \in \mathbb{N}$ and $SD \in [0, 1]$ from less to more support degree.

- **Risk Degree (RD):** is a value that estimates the risk for an argument to be attacked in view of the attacks received for an argument(s) with the same problem description and conclusion in the past. It is based on the number of argument-cases $\#argAccCAtt$ that were attacked out of the total number of $\#argAccC$ argument-cases with the same problem description and conclusion retrieved that were deemed acceptable.

$$RD = \begin{cases} 0, & if\ \#argAccC = \emptyset \\ \dfrac{\#argAccCAtt}{\#argAccC}, & otherwise \end{cases} \tag{3}$$

  with $\#argAccCAtt$, $\#argAccC \in \mathbb{N}$ and $RD \in [0, 1]$, from less to more risk of attack.

- **Attack degree (AD):** is a value that provides an estimation of the number of attacks att received by a similar argument(s) in the past. To compute this degree, the set of argument-cases with the

same problem description that were deemed acceptable is retrieved. Then, this set is separated into several subsets, one for each different conclusion that these argument-cases entail. The sets whose conclusion match with the conclusions of the arguments to assess are considered, while the other sets are discarded. Thus, we have a set of argument-cases for each different potential argument (and its associated conclusion) that we want to evaluate. For each argument-case in each set, the number of attacks received is computed (the number of distinguishing premises and counter-examples received). Then, for each set of argument-cases, the average number of attacks received is computed. The attack degree of each argument is calculated by a linear transformation:

$$AD = \begin{cases} 0, & if\ maxAtt = minAtt \\ \dfrac{att - minAtt}{maxAtt - minAtt}, & otherwise \end{cases} \tag{4}$$

with $minAtt$, $maxAtt$, $att \in \mathbb{N}$ and $AD \in [0, 1]$ from less to more degree of attack.

- **Efficiency degree (ED):** is a value that provides an estimation of the number of steps that it took to reach an agreement posing a similar argument in the past. It is based on the depth n from the node representing a similar argument-case retrieved to the node representing the conclusion in the dialogue graphs associated with it. To compute this degree, the same process to create the subsets of argument-cases as in the above degree is performed. Then, for each argument-case in each subset, the number of dialogue steps from the node that represents this argument-case to the end of the dialogue is computed. Also, the average number of steps per subset is calculated. Finally, the efficiency degree of each argument is calculated by a linear transformation:

$$ED = \begin{cases} 0, & if\ maxS = minS \\ 1 - \dfrac{n - minS}{maxS - minS}, & otherwise \end{cases} \tag{5}$$

with $minS$, $maxS$, $n \in \mathbb{N}$ and $ED \in [0, 1]$ from less to more efficiency.

- **Explanatory Power (EP):** is a value that represents the number of pieces of information that each argument covers. It is based on the number kr of knowledge resources were used to generate each similar argument-case retrieved. To compute this number, the same process to create the subsets of argument-cases as in the above degrees is performed. Then, for each argument-case in each set, the number of knowledge resources in the justification part is computed (the number of domain-cases and argument-cases). Then, for each set of argument-cases, the average number of knowledge resources used is computed. The explanatory power of each argument is calculated by a linear transformation:

$$EP = \begin{cases} 0, & if\ maxKr = minKr \\ \dfrac{kr - minKr}{maxKr - minKr}, & otherwise \end{cases} \tag{6}$$

with $minKr$, $maxKr$, $kr \in \mathbb{N}$ and $EP \in [0, 1]$ from less to more explanatory power.

Finally, the suitability factor of a new argument-case and its associated position is computed by the formula:

$$\begin{aligned} SF = &((w_{PD} * PD + w_{SD} * SD + w_{RD} * (1 - RD) \\ &+ w_{AD} * (1 - AD) + w_{ED} * ED + w_{EP} * EP)) \end{aligned} \tag{7}$$

where $w_i \in [0, 1], \sum w_i = 1$ are weight values that allow the agent to give more or less importance to each decision criteria. Finally, positions are ordered from more to less suitability by following the equation:

$$Suitability = w_{SimD} * SimD + w_{SF} * SF \tag{8}$$

where $w_i \in [0, 1], \sum w_i = 1$ are weight values that allow the agent to give more or less importance to the similarity degree or the support factor. Finally, the most suitable position of suitability level 1 is

selected as the one that the proponent agent is going to propose and defend first. Then, we assume that agents follow their value preference criteria when they select the positions to propose. However, each agent keeps the rest of positions to make alternative proposals if its original position is attacked and it is forced to withdraw it.

Algorithm 2 presents the pseudocode of the algorithm that implements the generation of positions, the generation of the associated argument-cases and the selection of positions. In the algorithm, the function *generatePositions* generates the n first positions by using the Algorithm 1; *generateArgument-Case* is a function that generates for each position its associated argument-case; *retrieveSimilarityDegree* is a function that retrieves the similarity degree of each position with regard to the problem to solve; *selectPosition* is a domain-dependent function that orders the set of positions from more to less suitable with respect to some domain-dependent criteria; and *mostSuitable* is a domain-dependent function that returns the most suitable position to solve the problem.

Also, *computeSF*, as shown in Algorithm 3, is a function that computes the support factor for each position by means of its associated argument-case. In this algorithm, the function *retrieveSameProblem* retrieves from the case-base of argument-cases those that have the same problem description than the current argument-case; *retrieveSameConclusion* retrieves from the case-base of argument-cases those that have the same problem description and conclusion than the current argument-case; *retrieveAccepted* retrieves from the case-base of argument-cases those that have the same problem description and conclusion than the current argument-case and were deemed acceptable; *retrieveAcceptedAttacked* retrieves from the case-base of argument-cases those that have the same problem description and conclusion than the current argument-case, were deemed acceptable and were attacked; *computeNumberOfAttacks* computes the number of attacks received by an argument-case; *computeNumberOfSteps* computes the number of steps from an argument-case to the node that represents the final conclusion in its associated dialogue-graph; and *computeNumberOfKR* computes the number of knowledge resources used to generate an argument-case.

3.1.3. Position Evaluation.

Each agent engaged in the agreement process can receive attacks to its position or, on the contrary, decide to attack the position of other agents. Thus, agents are able to evaluate its position with regard to other positions. The first step to evaluate an agent's position is to check if it is consistent with the positions of other agents. For the sake of simplicity, here we assume that a position is consistent with other position if they are the same (they totally match)[4].

On one hand, if the opponent's position matches the proponent's position no attack arguments are necessary, but the proponent generates a support argument to defend its position when it is attacked. On the other hand, if the opponent's position is in the set of positions generated by the proponent, but not ranked first, the proponent would accept the opponent's position if the latter has a power relation over the proponent and would try to attack the opponent's position otherwise. Finally, if the opponent's position is not in the set of positions generated by the proponent and the opponent does not have a power relation over the proponent, the proponent can try to generate an argument to attack the opponent's position. Otherwise, the proponent must accept the opponent's position. Next section explains the type of arguments that agents can generate and how these arguments are selected and evaluated.

Algorithm 4 presents the pseudocode of the position evaluation process. In the algorithm, the function *checkDependencyRelation* checks the dependency relation between the proponent and the opponent. As explained above, if the opponent's position is in the list of potential positions of the proponent but not ranked first, the proponent can use the function *decideAttack* to decide if it would attack the incoming position or just change its preferences. Also, *askForSupport* is a function that an agent can use to ask other agent to support its position.

3.2. Argument Management

As introduced in Section 2, agents can generate different types of arguments depending on their purposes. Also, agents can select the best argument to put forward and evaluate their arguments in view of other agents' arguments. The argument management process depends on the type of information that

[4]Broaden notions of consistency, such as one position being part or a more general position (e.g. an action that is part of a course of action proposed to solve a problem), can be considered in specific domains.

an agent receives from other agent. Here, we assume that this type can be identified from the type and content of the locutions that agents receive from other agents.

3.2.1. Argument Generation.

Agents generate arguments when they are asked to provide evidence to support a position (*support arguments*) or when they want to attack others' positions or arguments (*attack arguments*). To offer support evidences, a proponent agent can generate a support argument which support set consists of the set premises that describe the problem and match the knowledge resources that the proponent has used to generate and select its position and of any of these resources (domain-cases, argument-cases and argumentation schemes).

Attack arguments are generated when the proponent of a position provides an argument to justify it and an opponent wants to attack the position or more generally, when an opponent wants to attack the argument of a proponent. Algorithm 5 presents the pseudocode of the argument generation process. In the algorithm, the function *evaluateIncomingRequest* evaluates the type of the incoming request received. If the agent has been asked for supporting its position, it can decide to generate a support argument by using the domain-dependent function *decideSupport*. If the agent receives an attack, it must evaluate its current argument in view of the attacking argument by using the function *evaluateArgument*, which will be explained later in Section 3.2.3.

The attack arguments that the opponent can generate depend on the elements of the support set of the argument that the proponent has put forward. On one hand, if the support set includes a set of premises, the opponent can generate an attack argument including in its support set distinguishing premises that it knows (e.g. if it is in a privileged situation and knows extra information about the problem). Alternatively, the opponent can generate a distinguishing premise attack if it finds any premises implicit in a case that it used to generate its own position and matches the problem description, but these premises do not appear in the proponent's support argument. In this situation, the opponent could also generate an attack argument with this case as counter-example.

On the other hand, if the justification provided by the proponent is a domain-case or an argument-case, the opponent can check its case-bases and try to find counter-examples to generate an attack argument that includes them in the support set. Alternatively, it can also try to generate an attack argument with distinguishing premises extracted from these cases.

Algorithm 6 presents the pseudocode of the attack generation process. In the algorithm, the function *checkSupportSet* checks the elements of the support set of the incoming argument. With the function *selectElementToAttack*, the agent selects which element(s) of the support set it wants to attack. By means of *genetateDPAttack* the agent tries to attack the incoming argument with a distinguishing premise. The function *generateASAttack* tries to attack the incoming argument with a critical question of the argumentation scheme that supports the incoming argument. Also, *storeAS* is a function that agents can use to store an unknown scheme and decide later if it will be added to the ontology of argumentation schemes. With the function *generateCounterExample* the agent tries to generate a counter-example and with the function *generateCEAttack* the agent tries to attack the incoming argument with this counter-example.

3.2.2. Argument Selection.

Depending on the content of their knowledge resources, agents can generate several support or attack arguments. Thus, the agent has to select the best argument to put forward from the potential candidates. To select the best argument, the agent follows a similar process as it does for selecting positions. Therefore, the agent creates an argument-case that represents each potential argument and uses its argument-cases case-base to decide which one is most suitable to propose in view of its past argumentation experience and its current social context.

In the case of argument-cases associated to support arguments, all possible support arguments generated can be represented with the same argument-case (share the same problem description) and have the same conclusion (the position of the agent). Thus, the *Suitability Factor* does not provide useful information, and we assume that arguments that include more justifications for a position are more persuasive and should be proposed first.

In the case of attack arguments, as for selecting positions, the agent uses the information gained from previous argumentation processes to decide which argument would have more *Suitability Factor*. In

the case of draw, the agent has to decide which argument is going to put forward. In many cases this decision depends on the actual implementation of the agent and the dialogue strategy that it follows. However, by default, in our argumentation framework we propose a reflexive and transitive pre-order relation $<_p$ among the persuasive power of the knowledge resources used to generate an argument: $premises <_p distinguishing-premises <_p domain-cases <_p argument-cases$. Accepted argument-cases are the most persuasive knowledge resource to show as a justification for an argument, since they store the the maximum quantity of information about past arguments and argumentation processes (domain context, social context, if it was attacked and still remained accepted, etc.). Domain-cases are also very persuasive, since they store the final solution applied for a problem, but they do not provide information about the argumentation process and the social context. Finally, the premises that describe the problem to solve are known by any agent in the argumentation process and have the lowest persuasive power. In the case of attack arguments, distinguishing premises are more persuasive than description premises, since they provide extra information about which description premises have been taken into account to generate positions. Therefore, argument-cases with at least one argument-case in the justification part would be preferred to others and so on. If the draw still persists, a random choice will be made.

3.2.3. Argument Evaluation.

When agents receive arguments from other agents, they have to evaluate them. Then, a proponent agent can decide if an opponent's argument conflicts with its argument and hence, its argument is deemed acceptable, non-acceptable or remains undecided (it cannot make a decision over it). This evaluation is performed by using the *defeat relation* between arguments defined in our framework [21, Chapter 3], which specifies which attacks over arguments succeed.

If the proponent considers that its argument defeats the opponent's argument, it can try to generate a new argument to attack the opponent's, which would change the preliminary acceptability status of the opponent's argument to non-acceptable. Then, the opponent would evaluate the proponent's argument and try to counter-attack. On the contrary, if the proponent's argument cannot defeat the opponent's, it can still try to withdraw its last argument and send to the opponent a different argument to support its position, or even propose an alternative position (from its list of generated positions). In this case, the proponent's argument acceptability status would preliminary change to undecided. The final acceptability status of arguments is decided at the end of the argumentation dialogue. To define the rules of this process, we follow a *dialogue-game* protocol which complete definition is detailed in [21, Chapter 4].

Algorithm 7 presents the pseudocode of the argument evaluation process. As pointed out above, if the proponent argument defeats the opponent's argument, it can try to attack it by using the function *generateAttack*, shown in Algorithm 6. With the function *generateNewSupport* the agent can try to generate a new support for its position (since it cannot defeat the opponent's argument). However, if no new support can be generated, the agent has to withdraw its position from the dialogue by using the function *withdraw*.

In this section, we have presented the reasoning process that agents can use to generate, select and evaluate positions and arguments. In the following section, this process is illustrated by implementing them in a customer support application.

4. Evaluation Example

In this section, we evaluate the performance of the case-based reasoning process presented in this paper by running a set of empirical tests. With this objective, the underlying case-based argumentation framework has been implemented in the domain of a customer support application. Concretely, we consider a society of agents that act on behalf of a group of technicians that must solve problems in a Technology Management Centre (TMC). TMCs are entities which control every process implicated in the provision of technological and customer support services to private or public organisations. Usually, TMCs are managed by a private company that communicates with its customers via a call centre. This kind of centres allow customers to obtain general information, purchase products or lodge a complaint. They can also efficiently communicate public administrations with citizens. In a call centre, there are a number of technicians attending to a big amount of calls with different objectives –sales, marketing, customer service, technical support and any business or administration activity–. The call centre technicians have

computers provided with a helpdesk software and phone terminals connected to a telephone switchboard that manages and balances the calls among technicians. The current implementation is based in previous work that deployed a case-based multi-agent system in a real TCM [25]. This system was implemented and is currently used by the TCM company. In the original implementation, agents were allowed to use their case-bases to provide experience-based customer support. In this work, the system has been enhanced by allowing agents to argue about the best way of solving the incidents that the call centre receives.

Therefore, we consider a society composed by call-centre technicians with two possible roles, *operator* and *expert*. Operators form groups that must solve the problems that the call centre receives. Experts are specialised operators that have case-bases with knowledge about the suitable solutions to provide for specific problems. Therefore, the dependency relations in this society establish that experts have a power relation over operators and that technicians with the same role have a charity relation among them.

In this application domain we assume that each technician has a helpdesk application to manage the big amount of information that processes the call centre. The basic functions of this helpdesk are the following:

- To register the ticket information: customer data, entry channel and related project, which identifies the customer and the specific service that is being provided.
- To track each ticket and to scale it from one technician to a more specialised one or to a different technician in the same level.
- To warn when the maximum time to solve an incident is about to expire.
- To provide a solution for the ticket. This means to generate an own position or to ask for help to the members of a group.

In addition, this helpdesk would implement an argumentation module to solve each ticket as proposed in our framework. Hence, we assume the complex case where a ticket must be solved by a group of agents representing technicians that argue to reach an agreement over the best solution to apply. Each agent has its own knowledge resources (acceded via his helpdesk) to generate a solution for the ticket. The data-flow for the problem-solving process (or argumentation process) to solve each ticket is the following:

1. The system presents a group of technicians with a new ticket to solve.
2. If possible, each technician generates his own position by using the argumentation module. This module supports the argumentation framework proposed in this paper.
3. All technicians that are willing to participate in the argumentation process are aware of the positions proposed in each moment.
4. The technicians argue to reach an agreement over the most suitable solution by following a deliberation dialogue controlled by the proposed dialogue game protocol.
5. The best solution is proposed to the user and feedback is provided and registered by each technician helpdesk.

The helpdesk of each technician is provided with a case-based reasoning engine that helps them to solve the ticket. The new argumentation module will allow different technicians to reach agreements over the best solution to apply in each specific situation. In this example application, we assume that the most efficient technicians are acknowledged and rewarded by the company. Therefore, each technician follows a *persuasion* dialogue with their partners, trying to convince them to accept its solution as the best way to solve the ticket received, while observing the common objective of providing the best solution for a ticket from its point of view.

For the tests, a real database of 200 tickets solved in the past is used as domain knowledge. Translating these tickets to domain-cases, we have obtained a tickets case-base with 48 cases. Despite the small size of this case-base, we have rather preferred to use actual data instead of a larger case-base with simulated data. The argument-cases case-bases of each agent are initially empty and populated with cases as the agents acquire argumentation experience in execution of the system.

To diminish the influence of random noise, for each round in each test, all results report the average and confidence interval of 48 simulation runs at a confidence level of 95%, thus using a different ticket of

the tickets case-base as the problem to solve in each run. The results report the mean of the sampling distribution (the population mean) by using the formula:

$$\mu = \bar{x} \pm t * \frac{s}{\sqrt{n}} \tag{9}$$

where, $\bar{x}$ is the sample mean (the mean of the 48 experiments), t is a parameter that increases or decreases the standard error of the sample mean ($\frac{s}{\sqrt{n}}$), s is the sample standard deviation and n is the number of experiments. For small samples, say below 100, t follows the *Student's t-distribution*, which specifies certain value for the t parameter to achieve a confidence level of 95% for different sizes of population. In our case, with a population of 48 experiments the Student's t-distribution establishes a value of 2.0106 for t.

In each simulation experiment, an agent is selected randomly as initiator of the discussion. This agent has the additional function of collecting data for analysis. However, from the argumentation perspective, its behaviour is exactly the same as the rest of agents and its positions and arguments do not have any preference over others (unless there is a dependency relation that states it). The initiator agent receives one problem to solve per run. Then, it contacts its partners (the agents of its group) to report them the problem to solve. If the agents do not reach an agreement after a maximum time, the initiator chooses the most supported (the most voted) solution as the final decision (or the most frequent in case of draw). If the draw persists, the initiator makes a random choice among the most frequent solutions.

The case-based argumentation framework proposed in this work has been evaluated from different perspectives. On one hand, the *performance* of the system that implements the framework in the customer support application domain is tested and analysed. On the other hand, the ability of the system to take into account the *social context* of the participating agents is also verified.

4.1. Testing the Performance

The performance tests have been repeated and their results compared for the following decision policies:

- Random policy (CBR-R): each agent uses its Domain CBR module to propose a solution for the problem to solve. Then, a random choice among all solutions proposed by the agents is made. Agents do not have an Argumentation CBR module.
- Majority policy (CBR-M): each agent uses its Domain CBR module to propose a solution for the problem to solve. Then, the system selects the most frequently proposed solution. Agents do not have an Argumentation CBR module.
- Argumentation policy (CBR-ARG): agents have Domain and Argumentation CBR modules. Each agent uses its Domain CBR module to propose a solution for the problem to solve and its Argumentation CBR module to select the best positions and arguments to propose in view of its argumentation experience. Then, agents perform an argumentation dialogue to select the final solution to apply.

To evaluate the effect of the available argumentative knowledge that agents have, some tests are also repeated for the following specific settings of the argumentation policy. These settings cover the more interesting options regarding which agents have argumentation skills:

- CBR-ARG All-Argument-Cases (CBR-ARG AAC): All participating agent have argument-cases in their argument-cases case-base.
- CBR-ARG Initiator-Argument-Cases (CBR-ARG IAC): Only one agent, say the initiator agent, has argument-cases in its argument-cases case-base. Note that the selection of the initiator as the agent that has argumentative knowledge is just made for the sake of simplicity in the nomenclature. The behaviour of this agent only differs from the other agents' in the fact that it is in charge of starting the dialogue process and conveying the information about the final outcome. This do not affect its performance as dialogue participant and does not grant this agent any privileges over their partners.

- CBR-ARG Others-Argument-Cases (CBR-ARG OAC): All agents except from one, say the initiator, have argument-cases in their argument-cases case-bases.

With these tests, we evaluate the efficiency of the system that implements our framework under the different decision policies. By default, all agents know each other, all are in the same group and the dependency relation between them is *charity*. The values of each agent have been randomly assigned and agents know the values of their partners. Also, all agents play the role of *operator*. The influence of the social context will be evaluated in the Section 4.2. In addition, the agents that follow the argumentation policy assign weights to the *similarity degree* (w_{SimD}) and the *support factor* (w_{SF}) proportionally to the number of domain-cases and argument-cases that they have in their case-bases. Depending on the application domain, a different assignment for the weights could influence the performance of the system. However, due to the reduced size of our tickets case-bases, a proportional assignment is be suitable enough for the objectives of this performance evaluation. Equation 10 shows the simple rule that has been used in the tests.

$$\begin{aligned} w_{SimD} &= \frac{\#domaincases}{\#domaincases + \#argumentcases} \\ w_{SF} &= \frac{\#argumentcases}{\#domaincases + \#argumentcases} \end{aligned} \tag{10}$$

Also, by default agents set the same weight for all elements of the support factor.

4.1.1. Number of cases that the framework learns with respect of the time.

To perform this test, all agents follow the argumentation policy, with an initial number of 5 domain-cases in their domain-cases case-bases. The argument-cases case-base of all agents are initially empty. In each iteration, the agents use their CBR modules to propose and select positions and arguments and after this process, each agent updates its case-bases with the knowledge acquired.

If the system works properly, the knowledge acquired about past problem solving processes should increase with the time until some threshold, where the learning process should stabilize (because the cases in the case-bases of the agents cover most possible problems and arguments in the domain). To perform this test, we have executed several rounds to simulate the use of the system over certain period of time. For each repetition, we compute the average number of domain-cases and argument-cases in the case-bases of the agents. Figure 3 illustrates the results obtained in this test. The experiment has been repeated for 3, 5, 7 and 9 agents and the average number of domain-cases (DC) and argument-cases (AC) that all agents learn in each iteration has been computed. As expected, in all cases, the agents are able to learn the 48 domain-cases of the tickets case-base. However, if more agents participate in the dialogue, the quantity of domain knowledge that agents have available and interchange among them increases and the domain-cases case-bases are more quickly populated. Also, the quantity of argument-cases that agents are able to learn increases with the number of agents, since more potential positions and arguments give rise to more complex argumentation dialogues. As shown in the figure, the learning curve for the argument-cases is less soft than for the domain-cases, presenting peaks at some points. This is due to the fact that at some points of the dialogue, the agents can learn a specific domain-case that change its opinion about the best solution to apply for a specific category of problem. Therefore, the outcome of subsequent dialogues differ from the outcome that could be expected taking into account past similar dialogues and the argument-cases learning rate of the agents in those situations notably increases.

The results of this test have helped us to set the value of some parameters of the subsequent evaluation tests. The test shows that in 48 simulation runs, 3 agents are able to learn an average of the 54.8 % domain-cases of the tickets case-base, 5 agents the 56,6 %, 7 agents the 66,9 % and 9 agents the 73.1 %. The maximum number of argument-cases that agents are able to learn reaches an average of 20 argument-cases when 9 agents participate in the dialogue (18 argument-cases in the worst case). Therefore, due to the small size of the whole tickets case-base and the learning rates obtained in this test, the evaluation tests have been executed with a maximum number of 9 agents participating in the dialogue, with domain-cases case-bases populated with a maximum number of 45 domain-cases and argument-cases case-bases populated with a maximum number of 20 argument-cases (except from the social context tests, where a more varied choice of social contexts enables the learning of a larger number of argument-cases). Thus, the domain-cases of the case-bases of the agents will be randomly populated and increased from 5 to 45 cases in

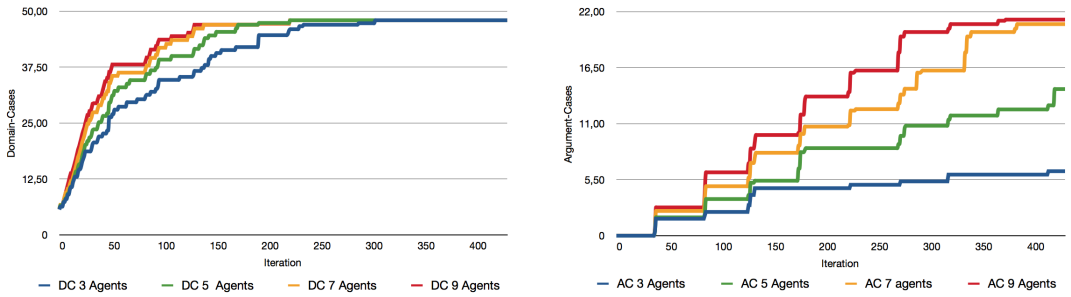

Figure 3: Number of domain-cases (left) and argument-cases (right) that agents learn.

each experimental round. The argument-cases case-bases of the agents for the argumentation-based policy are populated with 20 randomly selected argument-cases (from those acquired during the performance of the present test). Also, to evaluate the influence of the quantity of argumentative knowledge that the agents have in some tests, those tests are repeated for the case of 7 agents, setting the number of domain-cases of the case-bases of the agents to 20 (approximately the half part of the available cases in the tickets case-base), while varying the number or argument-cases of the argumentative agents from 0 to 18 cases, with an increase of 2 randomly selected argument-cases in each round. The number of the agents for these tests has been set to 7 to allow complex enough argumentation dialogues where the use of argument-cases can be useful, while having a reasonable case learning rate to avoid filling the case-bases with all the available knowledge for this case of study with a small number of simulations.

4.1.2. Percentage of problems that were properly solved with respect to the knowledge of the agents.

In this test, the percentage of problems that the system is able to solve, providing a correct solution, are computed. To check the solution accuracy, the solution agreed by the agents for each ticket requested is compared with its original solution, stored in the tickets case-base. One can expect that with more knowledge stored in the case-bases the number of problems that were correctly solved should increase. Figure 4 illustrates how as the number of agents participating in the dialogue increases, the solution proposed by them is more appropriate and similar to the actual solution registered in the tickets case-base for the ticket that has been requested to the agents (the mean error percentage in the solution predicted decreases). Obviously, if more agents participate in the problem solving process, the probability that one or more of them have a suitable domain-case that can be used to provide a solution for the current problem increases. The same happens if the number of domain-cases of the agents case-base increases. This applies also in the case of the random policy, although this policy never achieves the 100% of correct solution predictions. Also, the results achieved by the argumentation policy improve those achieved by the other policies, even when the domain-cases case-bases are populated with a small number of cases. The argumentation policy achieves more than a 50% of improvement for a domain-cases case-base size up to 25 cases if 3 agents participate in the dialogue, up to 20 cases if 5 agents participate and up to 15 cases if there are 7 or 9 agents participating. These results demonstrate that if agents have the ability of arguing, the agents whose solutions are more supported by evidence have more possibilities of wining the argumentation dialogue and hence, the quality of the final solution selected among all potential solution proposed by the agents increases. Finally, Figure 5 presents the results of this test if the number of domain-cases is set to 20 and the number of argument-cases that the agents have is increased in each round. The results show that the argumentative knowledge has no substantial influence on the accuracy of the solution proposed, at least for the data used in this case of study.

4.1.3. Percentage of agreements reached with respect to the knowledge of the agents.

In this test, we evaluate the percentage of times that an agreement is reached and a frequency-based or a random choice among all possible solutions proposed by the agents is not necessary. Figure 6 presents the results obtained. For all policies, the overall trend of the agreement percentage is to increase as the knowledge about the domain that agents have increases. Nevertheless, figures show slight fluctuations between results. This behaviour can be explained since the addition of a small quantity of new domain-cases between two simulation rounds can give rise to temporary situations, such as some agents changing

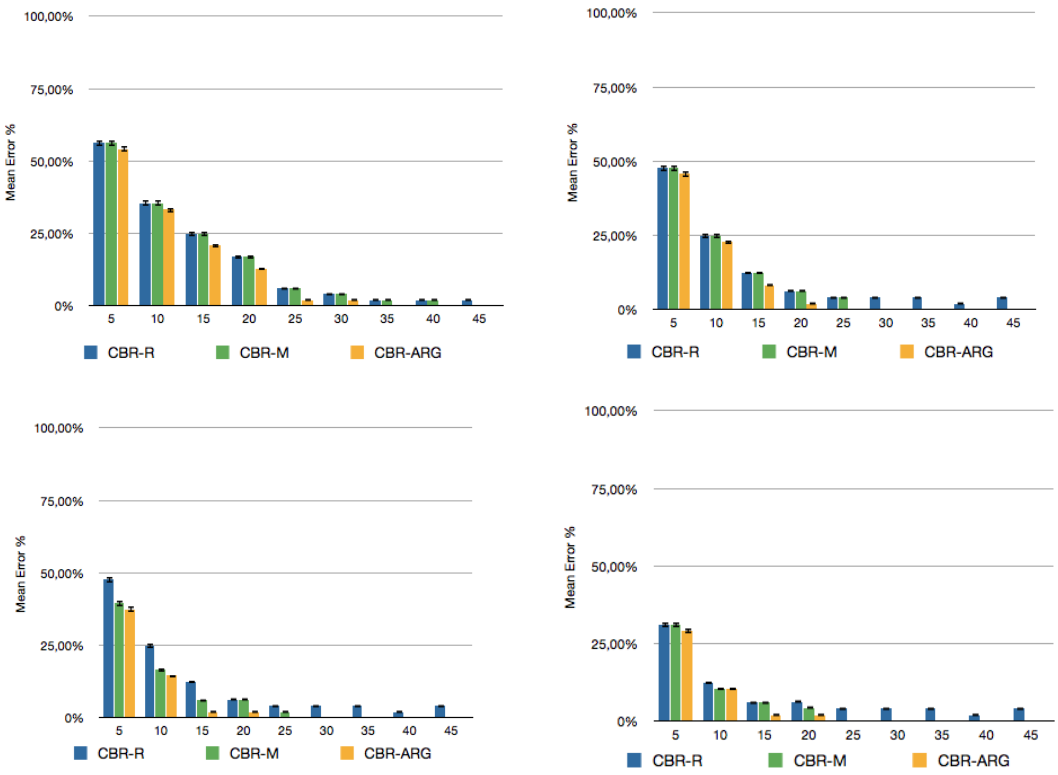

Figure 4: Solution prediction accuracy achieved by 3 (top-left), 5 (top-right), 7 (bottom-left) and 9 (bottom-right) agents ([5, 45]Δ5 domain-cases; 20 argument-cases).

temporarily their opinions until new information is gained or obtaining the same *suitability degree* for several positions and arguments. In the last case random choices are made, which can have a slight negative effect on the overall performance of the system.

For the case of 3 agents, the small number of participants in the dialogue results in all policies achieving similar agreement percentages. However, when the number of agents grows up to 5, if agents follow an argumentative policy that allows them to argue and persuade other agents, those who have more support for their positions win the dialogue and convince the others to accept them. Thus, the percentage of agreements reached increases. In addition, the improvement on the agreement percentage grows more quickly for a larger number of agents, reaching more than the 80% with little knowledge about the domain (e.g. 10 domain-cases) for 7 and 9 agents participating in the dialogue. These results capture the fact that with more participating agents, the knowledge available among all of them to solve tickets increases and more useful argument-cases improve the performance of complex argumentation dialogues.

More interesting results can be observed if we compare the agreement percentage that the argumentation policy achieves when useful argument-cases are available. To perform this test, the percentage of agreement that agents reach in those cases that they have been able to find useful argument-cases (argument-cases which problem description matches the current situation) has been computed. Note that the fact that agents have argument-cases in their argument-cases case-bases does not necessarily mean that these cases match the current dialogue context and are actually used by the agents to make their decisions. Therefore, Figure 7 illustrates the percentage of agreements that the argumentation policy achieves when one or more agents use their argumentative knowledge. In these tests, the fluctuations between subsequent simulation rounds are notably greater than in the previous tests. These fluctuations are due to the fact that the percentage of useful argument-cases highly depends on the domain knowledge that agents have and on the dialogue context.

In the case of 3 agents, the small number of dialogue participants give rise to very simple dialogues and no argument-cases are actually used. This explains that the CBR-ARG policy gets the same results in the

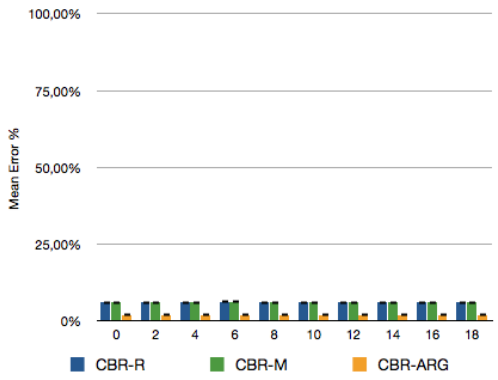

Figure 5: Solution prediction accuracy achieved by 7 agents (20 domain-cases; [0, 18]Δ2 argument-cases).

agreement percentage as the other policies, as shown in Figure 6. For 5, 7 and 9 agents, we can observe that when enough domain knowledge is available and agents engage in more complex dialogues (up to 30 domain-cases for 5 agents and up to 15 domain-cases for 7 and 9 agents), the agreement percentage has a global trend to increase when the initiator agent is the only agent that has useful argument-cases. This behaviour shows how the use of argumentative knowledge allows the initiator to argue better and persuade the other agents to accept their positions and reach an agreement. However, if more agents are also able to improve their argumentation skills by using their argumentative knowledge (CBR-ARG AAC and CBR-ARG OAC policies), less agents are persuaded to accept other agents' positions and hence, no agreement is reached in almost all simulations (except for the case of 45 domain-cases in the agents domain-cases case-bases).

As in Figure 6, Figure 7 illustrates how when the number of agents that participate in the dialogue increases, the agreement percentage also increases for the CBR-ARG IAC policy. This can be observed by comparing the agreement percentage achieved between 5 and 7 agents. Between 7 and 9 agents, no significant changes in the agreement percentage for the CBR-ARG IAC policy are observed, while the CBR-ARG AAC and CBR-ARG OAC policies improve their results when agents have a high amount of domain knowledge. However, this increase has less to do with the use of argumentative knowledge than with the fact that more agents participate in the dialogue with almost full knowledge about the domain. Thus, most of them are able to provide the same accurate solution for the problem to solve.

Figure 7 also presents the average number of locutions interchanged among the agents during the argumentation dialogue. As expected, results demonstrate that more locutions are needed to solve tickets if there are more agents participating in the process. However, the number of interchanged locutions seems to stabilize when the percentage of agreements reached approaches to 100%. Also, when only one agent has argumentative knowledge, the number of locutions (or let us say, the number of dialogue steps) that are necessary to reach a final decision among agents is more stable than in the cases where more agents use their argument-cases. In fact, the dialogue steps in the cases of 7 and 9 agents are almost the same for this policy. Therefore, the CBR-ARG IAC policy is also the more efficient policy, achieving the best performance results with shorter argumentation dialogues among the agents.

Finally, to evaluate the influence of the amount of argumentative knowledge of the agents on the agreement percentage, Figure 8 presents the results obtained by the argumentation policy when the number of argument-cases available for one or more agents is increased. When the initiator agent is the only agent that uses argumentative knowledge, as this knowledge increases, the probability of finding useful argument-cases to apply in each argumentation dialogue also increases. Therefore, this agent improves its argumentation skills and it is able to persuade the others to reach an agreement and accept its position as the best solution to apply for the ticket to solve. However, when several agents have a small quantity of argument-cases, the probability of finding a useful argument-case is very low. In these cases (CBR-ARG AAC with 6 argument-cases and CBR-ARG OAC with 2 argument-cases), the performance of the system suffers from a high randomness, and this agent that finds a useful argument-case has a higher advantage over the others, being able to persuade them to reach an agreement that favours its

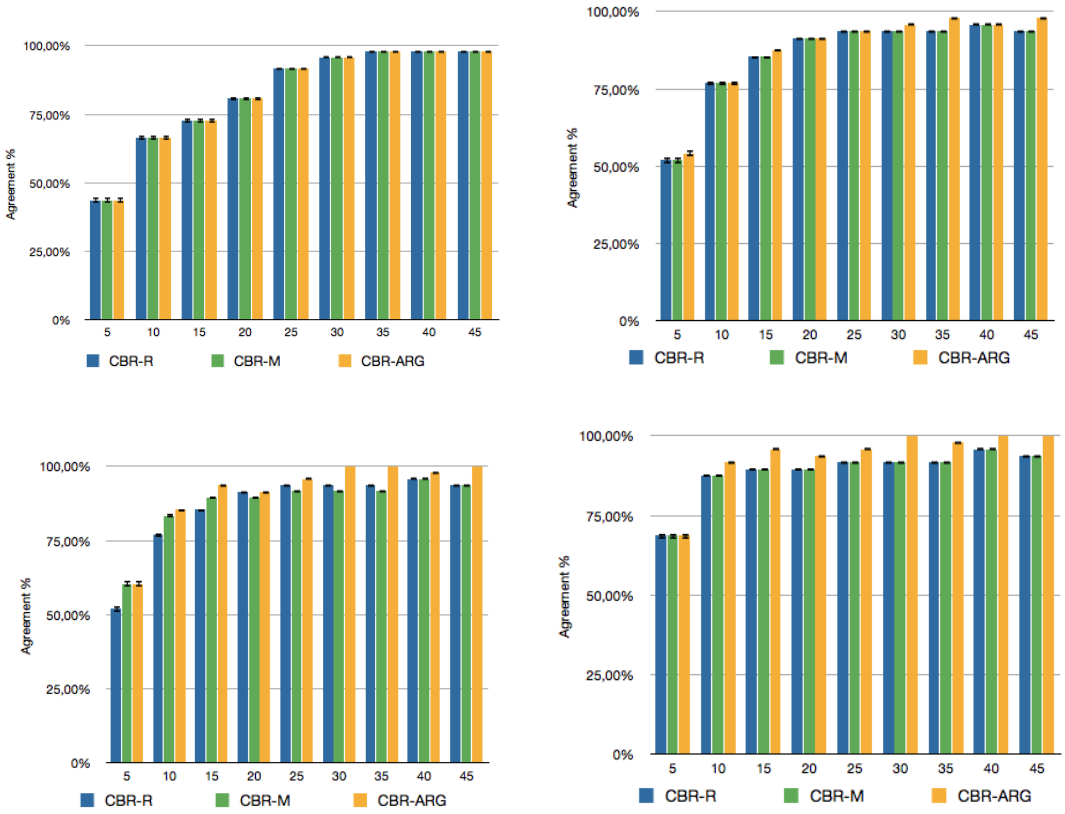

Figure 6: Percentage of agreement reached by 3 (top-left), 5 (top-right), 7 (bottom-left) and 9 (bottom-right) agents ([5, 45]Δ5 domain-cases; 20 argument-cases).

preferences. Regarding the number of locutions interchanged among the agents, Figure 8 illustrates how the number of locutions to reach the agreement is stable for all policies and does not depend on the argumentation knowledge that agents have. Thus, as pointed out before, the CBR-ARG IAC policy gets higher percentage of agreement when useful argument-cases are actually used.

4.1.4. Percentage of positions accepted with respect to the number of argument-cases.

This test evaluates the percentage of position that an agent (the initiator, for instance) gets accepted by the other agents in different settings of the argumentative policy, when useful argument-cases are used. The CBR-R and the CBR-M policies do not allow agents to argue and hence, they do not accept or defeat the position of other agents. Therefore, these policies are not considered for this test. Figure 9 illustrates how, independently of the number of agents participating in the dialogue, once the knowledge about the domain overpasses certain threshold (15 domain-cases), if an agent has argument-cases that match the current dialogue context, it gets its position accepted, even if the other participants have also useful argument-cases. However, if this agent does not have argumentative knowledge, but the other agents does, the percentage of acceptance depends on how useful the argumentative knowledge of the others is and it varies until the agent has enough domain knowledge to propose a good enough solution, as shown by the results obtained by the CBR-ARG OAC policy.

Finally, to evaluate the influence of the amount of argumentative knowledge that an agent has in the percentage of acceptance of its positions, Figure 10 presents the results obtained by setting the number of domain-cases to 20 and increasing the number of argument-cases by 2 in each round. Results show that if only one agent has argumentative knowledge (CBR-ARG IAC policy), once it has argument-cases that apply to the current dialogue situation and enough domain knowledge, it gets its position accepted. If other agents also have argumentative knowledge (CBR-ARG AAC policy), the percentage of acceptance

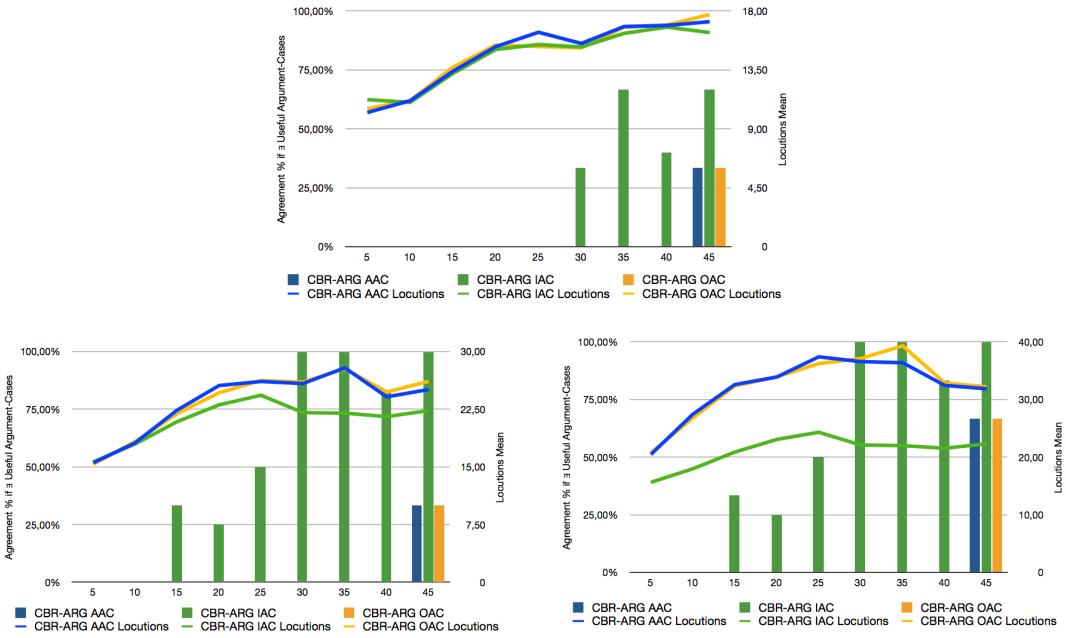

Figure 7: Percentage of agreement reached by 5 (top), 7 (bottom-left) and 9 (bottom-right) agents when useful argument-cases are available ([5, 45]Δ5 domain-cases; 20 argument-cases).

varies until enough useful argument-cases can be used (from 10 forwards), since this percentage depends also on how good are the argumentation skills of the other participants. Finally, if all participants have useful argumentative knowledge (CBR-ARG OAC policy), the percentage of acceptance of the initiator's position depends on which agent is able to create more persuasive arguments and convince the others to accept its position as the best to solve the requested ticket.

4.2. Testing the Social Context

The ability of the framework to represent the social context of the system has also been evaluated. To perform these tests, the system has been executed with 7 participating agents, following the argumentation policy (CBR-ARG). These settings are selected by taking into account the results of the performance tests, which show that this configuration allows agents to argue with fair enough information to provide suitable positions and arguments, but leaving room for the argumentation to make sense. The knowledge about the domain that each agent has is increased by 5 domain-cases in each round, from 5 to 45 domain-cases. Argumentative agents have a full argument-cases case-base populated with 20 cases. By default, all agents know each other, all are in the same group and the dependency relation between them depends on the specific test. The influence of different degrees of friendship and group membership are difficult to evaluate with the limited amount of data of our tickets case-base and remains future work. The values of each agent have been randomly assigned from a set of pre-defined values (*efficiency* of the problem solving process, *accuracy* of the solution provided and *savings* in the resources used to solve the ticket).

In addition, argumentative agents assign weights to the *similarity degree* (w_{SimD}) and the *support factor* (w_{SF}) proportionally to the relation between the number of domain-cases and argument-cases that they have in their case-bases, as it was done in the performance tests. Also, by default agents set the same weight for all elements of the support factor.

Following, the influence of the presence of an expert and the knowledge about the values of other agents in the system performance is evaluated.

4.2.1. Presence of an Expert

In this test, an agent has been allowed to play the role of an *expert*, while the rest of agents play the role of *operators*. An expert is an agent that has specific knowledge to solve certain types (categories) of

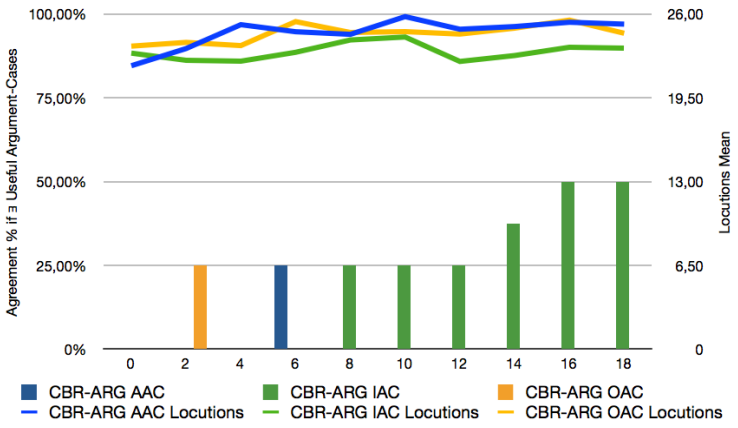

Figure 8: Percentage of agreement reached by 7 agents when useful argument-cases are available (20 domain-cases; [0, 18]Δ2 argument-cases).

problems and has its case-base of domain-cases populated with cases that solve them. Thus, the expert domain-cases case-base has as much knowledge as possible about the solution of past problems of the same type. That is, if the expert is configured to have 5 domain-cases in its domain-cases case-base, and there are enough suitable information in the original tickets case-base, these cases represent instances of the same type of problems. In the case that the tickets case-base has less than 5 cases representing such category of problems, 3 for instance, the remaining two cases are of the same category (if possible).

In our case, the expert agent has an *power* dependency relation over other technicians. Therefore, if it is able to propose a solution for the ticket requested, it can generate arguments that support its position and that will defeat other operators' arguments. This relation assigns more importance to the arguments of an agent that has a power dependency relation over other agents.

All simulation tests have been executed and their results compared for the random based decision policy (CBR-R Expert), the majority based decision policy (CBR-M Expert) and the argumentation based policy (CBR-ARG Expert). For these policies, the domain-cases case-base of the expert has been populated with expert domain knowledge. To evaluate the global effect of this expert knowledge, the results obtained for the accuracy of predictions when the domain-cases case-base of all agents are populated with random data are also shown for each policy (CBR-R, CBR-M and CBR-ARG).

Figure 11 illustrates how all policies are able to solve the same percentage of problems, but the accuracy of predictions is higher if agents are allowed to argue following the CBR-ARG Expert policy. Comparing the results obtained when the initiator has (CBR-R Expert, CBR-M Expert and CBR-ARG Expert) or does not have expert knowledge (CBR-R, CBR-M and CBR-ARG), as expected, agents are able to reach better accuracy in their final prediction when they are able to argue and there is an expert participating in the argumentation dialogue (CBR-ARG Expert). This demonstrates that the decisions of the expert prevail and, as it has more specialised domain-knowledge to propose solutions, the predictions of the system are more accurate.

4.2.2. Knowledge about Other Agents' Social Context

With these tests, we have evaluated the influence that the knowledge about the social context has in the performance of the system. Therefore, we have compared the performance of the system when the participating agents follow an argumentation policy and have full information about the social context of their partners (CBR-ARG), or on the contrary, do not know the preference over values that their partners have (CBR-ARG NV). In a real company, the dependency relations over technicians and the group that they belong are known by the staff. Hence, we assume that agents know this information about their partners.

In our evaluation domain, an agent assigns the same importance (weight) to both domain and argumentation knowledge to generate and select positions and arguments. However, if it does not know the value preferences of their partners, many times the agent uses argument-cases that are not suitable for

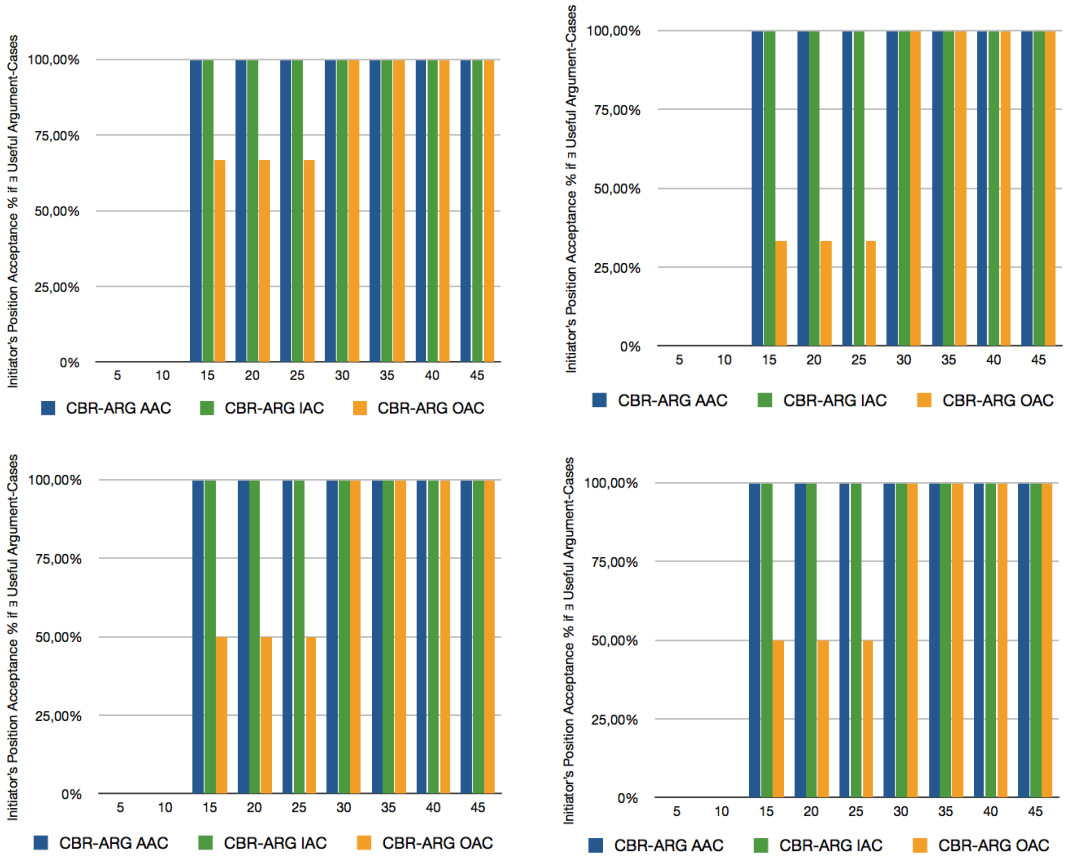

Figure 9: Percentage of positions accepted for 3 (top-left), 5 (top-right), 7 (bottom-left) and 9 (bottom-right) agents when useful argument-cases are available ([5, 45]Δ5 domain-cases; 20 argument-cases).

the current situation. This makes the agent to make wrong decisions that worsen the global performance of the system.

Figure 12 presents clearly that the performance of the system is negatively affected when argumentative agents use incorrect argument-cases to make their decisions. Then, for instance, the percentage of solved problems that argumentative agent are able to solve when they have full knowledge about the social context of their partners reaches almost the 100% with low knowledge about the domain (15 domain-cases), while it barely reaches the 50% when agents do not know the values of their partners. Similarly, the prediction error for well-informed agents is almost null with 15 domain-cases, while it still has a 5% error percentage with a high amount of knowledge about the domain (45 domain-cases) when agents ignore the values of the other agents.

Finally, Figure 13 also illustrates how system presents a poor performance in terms of the agreement percentage and the percentage of agents that agree when argumentative agents ignore the values of their partners. Again, the use of wrong argument-cases makes argumentative agents to propose solutions and arguments that hinder to reach agreements. This could be avoided if the system assigns less importance to the argumentative knowledge, by reducing the weight of the support factor (w_{SF}). In this way, a system that supports our framework can also perform well in domains where acquiring social context information about competitors is difficult, although this would significantly reduce the advantages of learning this type of information.

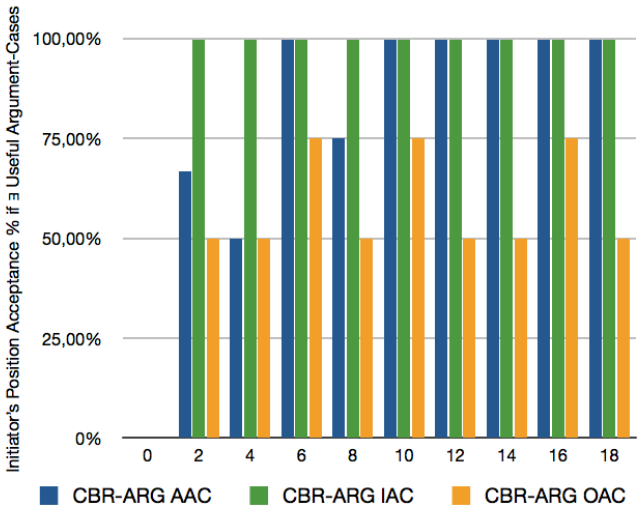

Figure 10: Percentage of positions accepted for 7 agents (20 domain-cases; [0, 18]Δ2 argument-cases).

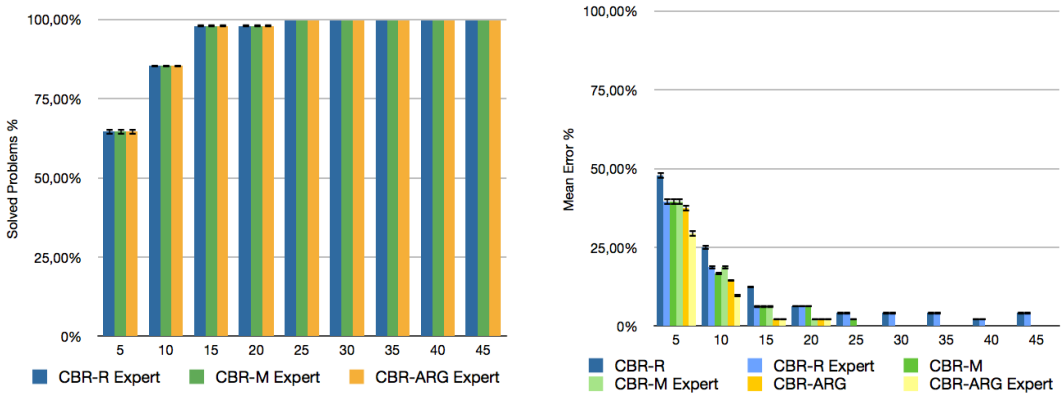

Figure 11: Percentage of problems that are solved by 1 expert and 6 operators (left) and accuracy of their predictions (right) ([5, 45]Δ5 domain-cases; 20 argument-cases).

5. Discussion

In the abstract formalisation of the argumentation framework proposed, the approach that we have followed is closely related with the proposal of value-based argumentation frameworks [26]. Also, a related work on abstract argumentation scheme frameworks [27] combines argumentation frameworks with argumentation schemes and makes use of the structure provided by the schemes to guide dialogues and provide contextual elements of argument evaluation. However, unlike our framework, the actual structure of arguments and their computational representation are obviated. In addition, previous argumentation experiences are not used to guide current argumentation processes such as we propose.

Other works use domain-dependent structures for the computational representation of arguments. The few current approaches for case-based argumentation in MAS, which use cases as previous knowledge to manage arguments, suffer from this domain-dependency or centralise the case-based argumentation abilities in a mediator agent [10]. The latter is the approach taken by the ProClaim system [14], which allows agents to argue and decide who is the best recipient of an organ transplantation. Close to the approaches on case-based argumentation is the research on experience-based argumentation using association rules, presented as the *PADUA* protocol in [28]. This work pools the opinions of several agents that have access

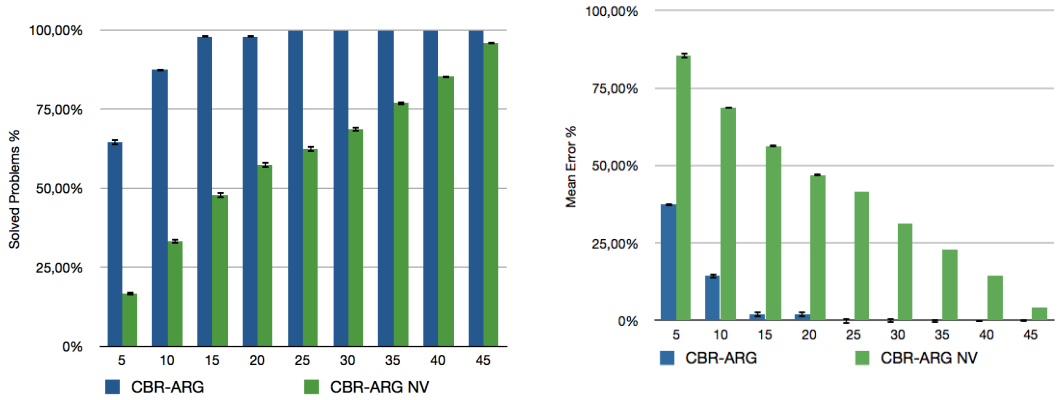

Figure 12: Percentage of problems that are solved by 7 agents (left) and accuracy of their predictions (right) ([5, 45]Δ5 domain-cases; 20 argument-cases).

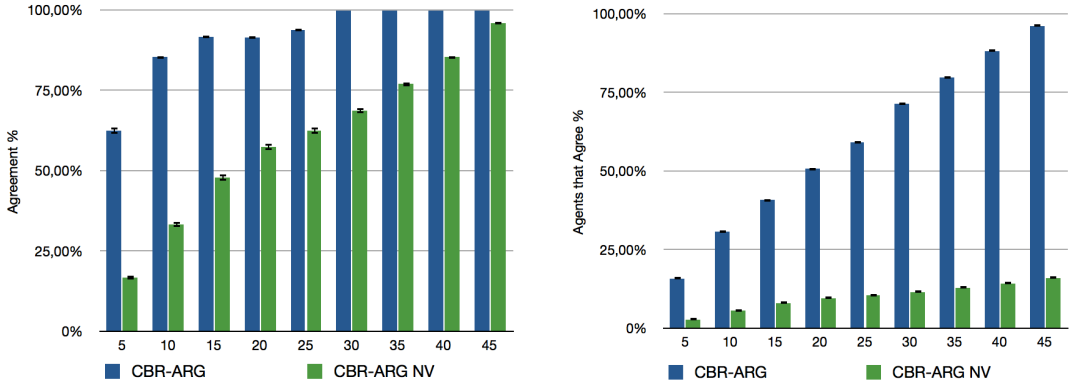

Figure 13: Percentage of agreement reached by 7 agents (left) and percentage of agents that agree (right) ([5, 45]Δ5 domain-cases; 20 argument-cases).

to different datasets to predict the classification of a new example. In a subsequent research, the PADUA protocol has been extended to allow multi-agent dialogues by proposing the *PISA* protocol [29, 30]. As in our approach, in this research agents take profit from previous experiences to solve a new problem, but the knowledge gained from the argumentation process is not stored nor used to improve the agents argumentation skills. In addition, PISA and PADUA have been designed to solve classification problems and not any type of problem that can be characterised by a set of features, which is the target of our research. However, the extension of our framework to allow agents to dynamically create groups and argue about the best move to make as a group in each step of the dialogue still remains future work.

In this work, we differentiate the concept of agent society from the concept of an agent organisation or a group of agents that play specific roles, follow some interaction patterns and collaborate to reach global objectives. Many works in argumentation in MAS that refer to the term 'agent societies' follow this approach [31][32], which is not targeted to the study of the structure of agent societies and the underlying social dependencies between agents. By contrast, we consider that the society is the entity that establishes the dependency relations between the agents, which have values and can dynamically group together to pursue common objectives in a specific situation. Similarly, our approach of an agent society differs from the notion of agent *coalitions* used in [33] and [34], which are temporary associations between agents in order to carry out joint tasks, without any consideration about the social links and values that characterise those agents.

Recent research presents a novel argumentation-based negotiation framework that allows agents to detect, manage and resolve conflicts related to their social influences in a distributed manner within a

structured agent society [35, 36]. However, this approach defines the social context of agents with a set of roles that agents can play, a set of generic relationships over them and a set of weighted social commitments for each of the active relationships, with no mention to values nor preferences over them. Other important difference between this proposal and our argumentation framework is the main objective pursued. While it focus on solving conflicts regarding conflicting social commitments between agents, our framework enables argumentation to solve a generic problem by using previous experiences, taking into account the social dependencies between agents but also their preferences over a set of values. Also, the authors do not specify the types of dependency relations that agents can have, leaving this concept as a generic relation. In our framework, the specific dependency relation between a pair of agents plays an important role in deciding if an argument posed in a past argumentation dialogue can be still persuasive in a current situation (where, possibly, agents hold a different dependency relation). For the time being, we do not deal with conflicts on dependency relations between agents, but this is an interesting extension that opens the pathway to future work.

For simplicity purposes, in this paper we have assumed that a proponent agent addresses its arguments to an opponent of its same group, having complete knowledge about the dependency relation that links them and about other agents' preferences (except from the social context tests). However, in real systems, some features of the social context of agents could be unknown. For instance, the proponent of an argument obviously knows its value preferences, probably knows the preferences of its group but, in a real open MAS, it is unlikely to know the opponent's value preferences. However, the proponent could know the value preferences of the opponent's group or have some previous knowledge about the value preferences of similar agents playing the same role as the opponent. If agents belong to different groups, the group features could be unknown, but the proponent could use its experience with other agents of the opponent's group and infer them.

Moreover, either the proponent or the opponent's features could represent information about agents that act as representatives of a group and any agent can belong to different groups at the same time. In any case, the framework is flexible enough to work with this lack of knowledge, although the reliability of the conclusions drawn from previous experiences would be worse.

Also, this paper does not show how agents can set different weights to the elements of the support factor to take strategic decisions about which arguments are more suitable in a specific situation. For instance, by assigning more importance to the efficiency degree, an agent can use the information stored in previous dialogue graphs to decide whether continuing with a current argumentation dialogue is worth. Then, to improve efficiency an argumentation dialogue could be finished if it is being similar to a previous one that didn't reach an agreement. Else, opponent moves in a dialogue could be inferred by looking a similar previous dialogue with the same opponent. These issues will be dealt with in future evaluation tests.

In our argumentation framework, cases are stored at the end of the argumentation process, but not all cases are necessarily stored. As in most CBR systems, argument-cases and domain-cases would be only stored if there is no similar enough case in the case-bases and the new domain and argumentation knowledge acquired must be kept. However, slightly different arguments could be represented with the same past argument-case by only updating its attacks information or attaching a new dialogue graph to its justification. Nevertheless, if the problem description, the acceptability of the argument or the conclusion change, a new argument-case has to be created. Also, to improve efficiency in searches, case-bases require a constant update to eliminate outdated cases, generalise and merge cases in a unique case when they are always indistinctly used, etc. Different case-base maintenance policies would be studied and tested in future research.

Regarding the application of the framework to different domains, our approach has also been tested in the context of a society of agents that must reach agreements about water resources allocation in a river basin [37]. Broadly speaking, the framework is suitable to be applied to any domain where different entities can engage in argumentation dialogues and the social context of these entities determines the way in which they can argue. In the call centre example we have assumed that agents do their best to win the argumentation process, thus following a persuasion dialogue. In this way, they get economical rewards and increase prestige. Despite that, those solutions that are better supported prevail. Hence, if agents do not follow a dialogue strategy that deviates the final outcome of the dialogue to fit their individual objectives, no matter if they give rise to the best solution to apply, the system reaches agreements that produce high quality solutions for the tickets received. This assumption has allowed us to perform more comprehensive

tests with the small amount of data that we have available. However, a cooperative approach where agents are not interested and collaborate to reach the best agreement would be appropriate for this example and will be implemented and evaluated in the future.

6. Conclusions

In this paper, we have followed a case-based approach to propose a reasoning process that agents can follow to automatically manage their positions and arguments and be able to engage in argumentation processes by taking into account their social context. This reasoning process makes use of the resources of a case-based argumentation framework for agent societies. Our proposals have been tested in a customer-support application domain.

The influence of the knowledge that an agent has about the social context of their partners has been evaluated. Results show that the arguments of experts are preferred, due to their power dependency relation over operators. Therefore, if an expert is actually better informed to assign high quality solutions to specific types of problems, the performance of the system improves. In view of the results achieved, we can conclude that our reasoning process actually performs an appropriate management of positions and arguments in argumentation dialogues that take place in agent societies.

Also, the performance of the system has been evaluated under different settings. The tests show how those agents that follow an argumentation policy based in our argumentation framework are able to provide more accurate solutions to the problems that the system receives. The ability of the agents to argue allows those who have better arguments to support their decisions to win the argumentation dialogue. Therefore, the solutions with higher quality are selected among those proposed.

In terms of the percentage of agreement, the argumentative agents get better results than agents following other policies, specially when the number of technicians that are arguing to solve the problem increases. Agents that have proposed less accurate solutions are persuaded to withdraw them and accept other agents' proposals, resulting in more agreements reached. In addition, it has been demonstrated that if an agent uses its argumentation knowledge and has enough domain-knowledge, its positions are accepted by other agents with more frequency.

The influence of the amount of argumentation knowledge that argumentative agents have has also been evaluated. If only one agent has argumentation knowledge that matches the context of current argumentation processes, as this knowledge increases, the amount of agreements reached and the number of agents that agree also increases. This demonstrates that this agent is effectively using its argumentation knowledge to select the best positions and arguments to put forward in a dialogue with other agents. Thus, as many useful arguments an agent has, as more proficient the agent is to persuade other agents to accept its proposals. However, if all or most agents have the ability of learning from argumentation dialogues, all of them have the same (high) persuasive power to defend their decisions and the agreement is difficult to achieve.

Future work will deal with the issues discussed in the previous section. On the one hand, the framework will be extended to allow agents to dynamically create groups and argue about the best move to make as a group in each step of the dialogue. Also, since conflicts on dependency relations between agents can appear in a real system, mechanisms to cope with them will be studied. On the other hand, further evaluation tests will set different weights to the elements of the support factor to take strategic decisions about which arguments are more suitable in a specific situation. In addition, different case-base maintenance policies would be studied and tested in future research. Finally, we plan to implement the framework in a cooperative environment where agents are not interested and collaborate to reach the best agreement for them all.

Acknowledgements

This work is supported by the Spanish government grants [CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2008-04446, and TIN2009-13839-C03-01] and by the GVA project [PROMETEO 2008/051].

References

[1] C. Sierra, V. Botti, S. Ossowski, Agreement Computing, KI - Künstliche Intelligenz DOI: 10.1007/s13218-010-0070-y.

[2] T. Alsinet, C. Chesñevar, L. Godo, S. Sadri, G. Simari, Formalizing argumentative reasoning in a possibilistic logic programming setting with fuzzy unification, International Journal of Approximate Reasoning 18 (3) (2008) 711–729.

[3] I. Rahwan, G. Simari (Eds.), Argumentation in Artificial Intelligence, Springer, 2009.

[4] P. Baroni, F. Cerutti, M. Giacomin, G. Guida, Afra: Argumentation framework with recursive attacks, International Journal of Approximate Reasoning 52 (1) (2011) 19–37.

[5] I. Rahwan, F. Zablith, C. Reed, Laying the foundations for a world wide argument web, Artificial Intelligence 171 (10-15) (2007) 897–921.

[6] D. Skalak, E. Rissland, Arguments and cases: An inevitable intertwining, Artificial Intelligence and Law 1 (1) (1992) 3–44.

[7] T. Bench-Capon, P. Dunne, Argumentation in artificial intelligence, Artificial Intelligence 171 (10-15) (2007) 619–938.

[8] T. Bench-Capon, G. Sartor, A model of legal reasoning with cases incorporating theories and values, Artificial Intelligence 150 (1-2) (2003) 97–143.

[9] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations and system approaches, AI Communications 7 (1) (1994) 39–59.

[10] S. Heras, V. Botti, V. Julián, Challenges for a CBR framework for argumentation in open MAS, Knowledge Engineering Review 24 (4) (2009) 327–352.

[11] K. Sycara, Persuasive argumentation in negotiation, Theory and Decision 28 (1990) 203–242.

[12] L.-K. Soh, C. Tsatsoulis, A real-time negotiation model and a multi-agent sensor network implementation, Autonomous Agents and Multi-Agent Systems 11 (3) (2005) 215–271.

[13] S. Ontañón, E. Plaza, Learning and joint deliberation through argumentation in multi-agent systems, in: 7th International Conference on Agents and Multi-Agent Systems, AAMAS-07, ACM Press, 2007.

[14] P. Tolchinsky, K. Atkinson, P. McBurney, S. Modgil, U. Cortés, Agents deliberating over action proposals using the ProCLAIM model, in: 5th International Central and Eastern European Conference on Multi-Agent Systems and Applications, CEEMAS-07, Springer, 2007, pp. 32–41.

[15] N. Karacapilidis, D. Papadias, Computer supported argumentation and collaborative decision-making: the HERMES system, Information Systems 26 (4) (2001) 259–277.

[16] C. Perelman, L. Olbrechts-Tyteca, The New Rhetoric: A Treatise on Argumentation, University of Notre Dame Press, 1969.

[17] J. Searle, Rationality in Action, MIT Press, 2001.

[18] S. Kaci, L. van der Torre, Preference-based argumentation: Arguments supporting multiple values, International Journal of Approximate Reasoning 48 (3) (2008) 730–751.

[19] L. Amgoud, C. Devred, M.-C. Lagasquie-Schiex, Generating possible intentions with constrained argumentation systems, International Journal of Approximate Reasoning 52 (9) (2011) 1363–1391.

[20] N. Gorogiannis, A. Hunter, M. Williams, An argument-based approach to reasoning with clinical knowledge, International Journal of Approximate Reasoning 51 (1) (2009) 1–22.

[21] S. Heras, Case-Based Argumentation Framework for Agent Societies, Ph.D. thesis, Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. http://hdl.handle.net/10251/12497 (2011).
URL `http://hdl.handle.net/10251/12497`

[22] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n -person games, Artificial Intelligence 77 (1995) 321–357.

[23] D. Walton, C. Reed, F. Macagno, Argumentation Schemes, Cambridge University Press, 2008.

[24] F. Dignum, H. Weigand, Communication and Deontic Logic, in: R. Wieringa, R. Feenstra (Eds.), Information Systems - Correctness and Reusability. Selected papers from the IS-CORE Workshop, World Scientific Publishing Co., 1995, pp. 242–260.

[25] S. Heras, J. A. García-Pardo, R. Ramos-Garijo, A. Palomares, V. Botti, M. Rebollo, V. Julián, Multi-domain case-based module for customer support, Expert Systems with Applications 36 (3) (2009) 6866–6873.

[26] T. Bench-Capon, K. Atkinson, Argumentation in Artificial Intelligence, Springer, 2009, Ch. Abstract argumentation and values, pp. 45–64.

[27] K. Atkinson, T. Bench-Capon, Abstract argumentation scheme frameworks, in: Proceedings of the 13th International Conference on Artificial Intelligence: Methodology, Systems and Applications, AIMSA-08, Vol. 5253 of Lecture Notes in Artificial Intelligence, Springer, 2008, pp. 220–234.

[28] M. Wardeh, T. Bench-Capon, F. P. Coenen, PISA - Pooling Information from Several Agents: Multiplayer Argumentation From Experience, in: Proceedings of the 28th SGAI International Conference on Artificial Intelligence, AI-2008, Springer, 2008, pp. 133–146.

[29] M. Wardeh, T. Bench-Capon, F. P. Coenen, Padua: a protocol for argumentation dialogue using association rules, AI and Law 17 (3) (2009) 183–215.

[30] M. Wardeh, F. Coenen, T. Bench-Capon, Arguing in groups, in: 3rd International Conference on Computational Models of Argument, COMMA-10, IOS Press, 2010, pp. 475–486.

[31] J. Ferber, O. Gutknecht, F. Michel, From Agents to Organizations: an Organizational View of Multi-Agent Systems, in: Agent-Oriented Software Engineering VI, Vol. 2935 of LNCS, Springer-Verlag, 2004, pp. 214–230.

[32] E. Oliva, P. McBurney, A. Omicini, Co-argumentation artifact for agent societies, in: 5th International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-08, ACM Press, 2008, pp. 31–46.

[33] L. Amgoud, An argumentation-based model for reasoning about coalition structures, in: 2nd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-05, ACM Press, 2005, pp. 1–12.

[34] N. Bulling, J. Dix, C. I. Chesñevar, Modelling coalitions: Atl + argumentation, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS-08, Vol. 2, ACM Press, 2008, pp. 681–688.

[35] N. C. Karunatillake, Argumentation-based negotiation in a social context, Ph.D. thesis, School of Electronics and Computer Science, University of Southampton, UK (2006).

[36] N. C. Karunatillake, N. R. Jennings, I. Rahwan, P. McBurney, Dialogue Games that Agents Play within a Society, Artificial Intelligence 173 (9-10) (2009) 935–981.

[37] S. Heras, V. Botti, V. Julián, On a Computational Argumentation Framework for Agent Societies, in: AAMAS 7th International Workshop on Argumentation in Multi-Agent Systems, ArgMAS-10, ACM Press, 2010, pp. 55–72.

Algorithm 1 generatePositions

Require: ProblemDescription, n, generation method (D, S, M) *// The description of the problem to solve, the maximum number n of positions to generate (all possible if n=0) and the method to generate positions*

```
matchCases = ∅
solutions = ∅
positions = ∅
argSchemes = ∅
extendedDesc = ∅
similarity = 0
SD = ∅
i = 0; j = 0; k = 0
if D ∈ GenerationMethod then
    for all c ∈ DomainCasesCB do
        similarity = computeSimilarity(ProblemDescription, c)
        if similarity > δ then
            matchCases[i] = c //If the similarity exceeds certain threshold, the domain-case is selected to generate the position
            SD[i] = similarity //The similarity degree of this domain-case is stored
            i++
    solutions = generateSolutions(matchCases)
    if lenght(solutions) ≥ 1 then
        for [s = 0;s < lenght(solutions);s + +] do
            positions = addPosition(ProblemDescription, solutions[s], SD[i])
if M ∈ GenerationMethod then
    for all c ∈ DomainCasesCB do
        similarity = computeSimilarity(ProblemDescription, c)
        if similarity > δ then
            matchCases[i] = c //If the similarity exceeds certain threshold, the domain-case is selected to generate the position
            SD[i] = similarity //The similarity degree of this domain-case is stored
            i++
    extendedDesc = aggregateDescriptions(matchCases)
    if ProblemDescription ≠ extendedDesc then
        ProblemDescription = extendedDesc
        if S ∉ GenerationMethod then
            addGenerationMethod(S)
    else
        solutions = generateSolutions(matchCases)
    if lenght(solutions) ≥ 1 then
        for [s = 0;s < lenght(solutions);s + +] do
            positions = addPosition(ProblemDescription, solutions[s], SD[i])
if S ∈ GenerationMethod then
    for all as ∈ ArgumentationSchemesOnt do
        similarity = computeSimilarity(ProblemDescription, as)
        if similarity > η then
            argSchemes[j] = k //If the similarity exceeds certain threshold, the argumentation-scheme is selected to generate the position
            SD[j] = similarity //The similarity degree of this argumentation-scheme is stored
            j++
    solutions = generateSolutions(argSchemes)
    if lenght(solutions) ≥ 1 then
        for [s = 0;s < lenght(solutions);s + +] do
            positions = addPosition(ProblemDescription, solutions[s], SD[i])
Return positions
```

Algorithm 2 Position Generation and Selection

Require: ProblemDescription, generation method (D, S, M), w_{SimD}, w_{PD}, w_{SD}, w_{RD}, w_{AD}, w_{ED}, w_{EP} *//The description of the problem to solve, the generation method for generating cases from domain-cases (D), argumentation-schemes (S) or from both (M) and the weights for each element of the similarity degree and the support factor*

positions = $\emptyset$
argumentCases = $\emptyset$
SimD = $\emptyset$
SF = $\emptyset$
selectedPositions = $\emptyset$
positions = generatePositions(ProblemDescription, n)
for $[i = 1; i \leq lenght(positions); i++]$ **do**
    argumentCases[i] = generateArgumentCase(ProblemDescription, positions[i])
    SimD[i] = retrieveSimilarityDegree(positions[i])
for $[i = 1; i \leq lenght(argumentCases); i++]$ **do**
    SF[i] = computeSF(ProblemDescription, argumentCases[i], argumentCases, w_{PD}, w_{SD}, w_{RD}, w_{AD}, w_{ED}, w_{EP})
selectedPositions = selectPosition(positions, argumentCases, SD, SF)
Return mostSuitable(selectedPositions)

Algorithm 3 computeSF

Require: ProblemDescription, argCase, argumentCases, w_{PD}, w_{SD}, w_{RD}, w_{AD}, w_{ED}, w_{EP} *//The description of the problem to solve, an argument case, the set of all argument-cases that represent all potential positions and the weights for each element of the support factor*

SF=0; PD=0; SD=0; RD=0; AD=0; ED=0; EP=0
att=0; minAtt=0; maxAtt=0; attAC=0
n=0; minS=0; maxS=0; stepsAC=0
kr=0; minKr=0; maxKr=0; krAC=0
arg = $\emptyset$
argC = $\emptyset$
argAccC = $\emptyset$
argAccCAtt = $\emptyset$
arg = retrieveSameProblem(argCase, ArgumentCasesCB)
argC = retrieveSameConclusion(argCase, ArgumentCasesCB)
argAccC = retrieveAccepted(argCase, ArgumentCasesCB)
argAccCAtt = retrieveAcceptedAttacked(argCase, ArgumentCasesCB)
if lenght(argC) $\neq$ 0 **then**
    PD = argAccC/argC
if lenght(arg) $\neq$ 0 **then**
    SD = argAccC/arg
if lenght(argC) $\neq$ 0 **then**
    RD = argAccCAtt/argC
att = computeNumberOfAttacks(argCase)
minAtt = att
maxAtt = att
for all [$ac \in argumentCases$] **do**
    attAC = computeNumberOfAttacks(ac)
    if minAtt > attAC **then**
        minAtt = attAC
    if maxAtt < attAC **then**
        maxAtt = attAC
if maxAtt $\neq$ minAtt **then**
    AD = (att - minAtt)/(maxAtt - minAtt)
n = computeNumberOfSteps(argCase)
minS = n
maxS = n
for all [$ac \in argumentCases$] **do**
    stepsAC = computeNumberOfSteps(ac)
    if minS > stepsAC **then**
        minS = stepsAC
    if maxS < stepsAC **then**
        maxS = stepsAC
if maxS $\neq$ minS **then**
    ED = 1 - ((n - minS)/(maxS - minS))
kr = computeNumberOfKR(argCase)
minKr = kr
maxKr = kr
for all [$ac \in argumentCases$] **do**
    krAC = computeNumberOfKR(ac)
    if minKr > krAC **then**
        minKr = krAC
    if maxKr < krAC **then**
        maxKr = krAC
if maxKr $\neq$ minKr **then**
    EP = (kr - minKr)/(maxKr - minKr)
Return $SF = (w_{PD} * PD + w_{SD} * SD + w_{RD} * (1 - RD) + w_{AD} * (1 - AD) + w_{ED} * ED + w_{EP} * EP)$

Algorithm 4 Position Evaluation

Require: position, incPosition, positions //*The position of an agent, an incoming position and the set of potential positions generated by the agent*

```
dependencyR = checkDependencyRelation(opponent, proponent)
if (dependencyR =! "Power") then
    if position = incPosition then
        acceptPosition(incPosition) //If the positions are the same, the agent accepts the incoming position
    if (position ≠ incPosition) && (incPosition ∈ positions) then
        decideAttack(incPosition)
    if incPosition notin positions then
        askForSupport(incPosition)
else
    acceptPosition(incPosition) //If the opponent agent has a power relation over the proponent, it changes its position and accepts the incoming position
```

Algorithm 5 Argument Generation

Require: position, incArgument //*The position of the agent and the incoming argument*

```
requestType = evaluateIncomingRequest()
if requestType = supportAsked then
    decideSupport(position)
if requestType = attackReceived then
    evaluateArgument(incArgument)
```

Algorithm 6 generateAttack

Require: incArgument, DomainCasesCB, ArgumentCasesCB, ArgumentationSchemes //*The argument to attack, the case-bases of domain-cases and argument-cases and the argumentation-schemes ontology*

```
support = checkSupportSet(incArgument)
supportElement = selectElementToAttack(support)
if supportElement = Premises then
    generateDPAttack(incArgument)
if supportElement = ArgumentationSchemes then
    if ArgumentationScheme ∈ ArgumentationSchemesOnt then
        generateASAttack(incArgument)
    else
        storeAS(ArgumentationScheme)
if (supportElement = Domain-Case) or (supportElement = Argument-Case) then
    CE=generateCounterExample(incArgument)
    if CE ≠ ∅ then
        generateCEAttack(incArgument)
    else
        generateDPAttack(incArgument)
if support = ∅ then
    generateCEAttack(incArgument)
```

Algorithm 7 evaluateArgument

Require: incArgument //*The incoming argument to evaluate*

```
if defeats(currArgument, incArgument) then
    generateAttack(incArgument)
else
    newS = generateNewSupport(position)
    if newS == 0 then
        withdraw(position)
```