An Approach to Argumentative Reasoning Servers with Multiple Preference Criteria

Juan Carlos Teze[1,2,3], Sebastian Gottifredi[1,3], Alejandro J. Garcia[1,3] and Guillermo R. Simari[1]

[1]Artificial Intelligence Research and Development Laboratory (LIDIA)
Department of Computer Science and Engineering (DCIC)
Universidad Nacional del Sur (UNS) - Alem 1253
(8000) Bahía Blanca, Buenos Aires, Argentina

[2]Agents and Intelligent Systems Area, Fac. of Management Sciences,
Universidad Nacional de Entre Ríos (UNER)
(3200) Concordia, Entre Ríos, Argentina

[3]Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
e-mail:{jct,sg,ajg,grs}@cs.uns.edu.ar

Abstract. Argumentation is an attractive reasoning mechanism due to its dialectical and non monotonic nature, and its properties of computational tractability. In dynamic domains where the agents deal with incomplete and contradictory information, to determine the accepted or warranted information, an argument comparison criterion must be used. Argumentation systems that use a single argument comparison criterion have been widely studied in the literature. In some of these approaches, the comparison is fixed and in others the criterion can be replaced in a modular way. In this work we introduce an argumentative server that provides recommendations to its client agents and the ability to decide how multiple argument comparison criteria can be combined. In the proposed formalism, the argumentative reasoning is based on the criteria selected by the client agents. As a result, a set of operators to combine multiple preference criteria is presented.

1 Introduction

An essential characteristic of Multi-Agent Systems (MAS) is the modeling of the interaction among agents. Agents in a MAS interact to perform tasks, that can be collectively carried out by a set of agents or can be individually done by one agent. Generally, deliberative agents reason using two types of knowledge: public knowledge that is shared with other agents, and private knowledge that in part come from their own perception of the environment; in [10] a client-server model was proposed allowing the representation of both private and shared knowledge.

A defeasible argumentation system provides ways to confront contradictory statements to determine whether some particular information can be accepted or, using a technical term, warranted [9, 1, 6, 7]. The result of the argumentation process leads to an answer involving many stages; the comparison of conflicting

arguments to decide which one prevails is a particularly important one. For this reason, the definition of a formal criterion for comparing arguments becomes a central problem in defeasible argumentation.

Argumentation systems using a single argument comparison criterion have been widely studied in the literature [18, 9, 5, 15]. The argument comparison criterion represents a fundamental part of an argumentation system because the inferences an agent can obtain from its knowledge will depend on the criterion that is used. In the literature of argumentative systems, several approaches use a fixed comparison criterion embedded in the system and in others the criterion can be replaced in a modular way. In [2, 11, 8], the authors also focused their works in multiple criteria, however, in a different manner to the way is proposed in this paper. The main contribution of this paper is to provide a framework where several comparison criteria can be selected and combined for deciding which argument prevails. Next, we show an example that will serve two purposes: to motivate the main ideas of our proposal, and as a running example to be used in the rest of the paper.

Example 1 *Lets consider a hotel that has the following general information about its clients:*

- *if a client travels alone and she does not want a jacuzzi included with the room, the hotel does not recommend a junior suite,*
- *if a client travels alone and she wants a safe and a fridge included with the room, the hotel recommends a junior suite,*
- *if a client plans a short stay and she does not want a jacuzzi with the room, the hotel recommends a single room,*
- *if a client plans a short stay and she wants a fridge with the room, the hotel does not recommend a single room,*
- *finally, if the hotel recommends a junior suite, it does not recommend a single room,*

Now, two clients (Joan and Michael) request a room to the hotel. Suppose that for both clients provide the same information:

- *each travels alone.*
- *each plans for a short stay trip.*
- *each wants a room with safe and fridge.*

They also share the same criteria for selecting a room, i.e., comfort and price; however, Joan and Michael combine them differently. For Joan the room must satisfy both criteria of the comfort and price. But, Michael is more tolerant and he expects that the room will meet at least one criterion, either comfort or price. In this situation each client can receive contradictory recommendations. This shows how the client's criteria to select a room can be used to establish which recommendation prevails. Since, each client combine the same criteria on different ways, the results are different.

In [3], a framework to reason from multiple points of view on an inconsistent knowledge base was proposed. In that formalism each preference is associated with a context, and these contexts are totally ordered; consequently, the relation among each preference is fixed by this ordering. In contrast to [3], our approach does not depend on a fixed pre-ordering between preferences. Here we generalize the way several preferences can relate to each other through a set of operators used to combine them. In our approach, following a client-server model, the client agent can decide, for instance, the order in which the criteria will be used.

Recommender systems [16, 17, 14] have become an important research area in AI over the last decade. We focus on a particular form of implementing recommender systems called Recommender Servers that extends the integration of argumentation and recommender systems to a MAS setting. Recommender Servers are based on an implementation of DeLP [9] called DeLP-Server [10]. In this paper we will introduce a defeasible logic programming recommender server that gives to the clients the ability to decide how multiple argument comparison criteria can be combined. A set of criteria-combination operators is proposed to provide that capability.

The rest of the paper is structured as follows. Next, in Section 2 we will present the necessary background introducing basic definitions and some works that will be used in the rest of the paper; then, in Section 3 we will introduce a preference based recommender server. To illustrate the formalism, in Section 4 we introduce an example in DeLP. Finally, in Section 5 we discuss related work and offer our conclusions and the possible directions for our future work.

2 Background

In [10] an implementation of DeLP, called DeLP-server, has been presented; this system provides an argumentative reasoning service for multi-agent systems. A DeLP-server is a stand-alone application that stores a DeLP-program that is used to answer client queries. To answer queries, the DeLP-server will use the public knowledge stored and represented as Defeasible Logic Program complementing it with individual knowledge that clients might send, thus creating a particular scenario for the query. The information that modifies the public knowledge stored in the DeLP-server is called context, denoted $\mathbf{C}$.

In DeLP, knowledge is represented using facts, strict rules and defeasible rules. Facts are ground literals representing atomic information or the negation of atomic information using the strong negation "$\sim$". Defeasible Rules are denoted $L_0 \prec L_1, \ldots, L_n$ and represent defeasible knowledge, *i.e.,* tentative information, where the head L_0 is a literal and the $body\{L_i\}_{i>0}$ is a set of literals. DeLP-servers consider a restricted form of programs that do not have strict rules.

Example 2 *Continuing with Example 1, let $\mathcal{P}_h$ be a DeLP-program that models the described information about hotel clients;*

$$\mathcal{P}_h = \left\{ \begin{array}{l} \sim junior\_suite \prec travel\_alone, \sim jacuzzi. \\ junior\_suite \prec travel\_alone, safe, fridge. \\ single\_room \prec short\_stay, \sim jacuzzi. \\ \sim single\_room \prec short\_stay, fridge. \\ \sim single\_room \prec junior\_suite. \end{array} \right\}$$

The private pieces of information related to the client's particular context will be represented through the following DeLP-program:

$$\mathbf{C}_{Joan} = \mathbf{C}_{Michael} = \mathcal{P}_c = \left\{ \begin{array}{l} travel\_alone. \\ short\_stay. \\ fridge. \\ safe. \end{array} \right\}$$

where $\mathbf{C}_{Joan}$ is Joan's particular context and $\mathbf{C}_{Michael}$ is Michael's particular context. For this example, the contextual information is the same for both clients.

In [10], three operators for DeLP-programs were introduced to consider different ways in which the specific information of the clients is taken into account at the moment of computing answers; these proposed operators would temporally modify the public knowledge stored in the server just for obtaining the answer. Here, our research is not focussed in these contextual operators, and we will use the union operator $\cup$ as a simple context treatment operator.

Example 3 *Consider Example 2.*

$$\mathcal{P}_h \cup \mathcal{P}_c = \mathcal{P}_m = \left\{ \begin{array}{l} \sim junior\_suite \prec travel\_alone, \sim jacuzzi. \\ junior\_suite \prec travel\_alone, safe, fridge. \\ single\_room \prec short\_stay, \sim jacuzzi. \\ \sim single\_room \prec short\_stay, fridge. \\ \sim single\_room \prec junior\_suite. \\ travel\_alone. \\ short\_stay. \\ fridge. \\ safe. \end{array} \right\}$$

When reasoning with contradictory and dynamic information, DeLP builds arguments from this program. An argument for a literal L is a minimal and non contradictory set of defeasible rules such that thogheter with programs strict knowledge that allows for the derivation of L. For a given program the set of all posible arguments will be denoted as *Args*.

Example 4 *Extending Example 3, from program $\mathcal{P}'$ two arguments can be built. The argument $\mathcal{A}$ in favor of recommending a junior suite:*

$$\mathcal{A} = \langle \{ junior\_suite \prec travel\_alone, safe, fridge. \} \rangle$$

and the argument $\mathcal{B}$ in favor of not recommending a junior suite:

$$\mathcal{B} = \langle \{ \sim junior\_suite \prec travel\_alone, \sim jacuzzi. \} \rangle$$

Given an argument $\mathcal{A}_2$ that is in conflict with the argument $\mathcal{A}_1$, in order to decide which one prevails, these two arguments must be compared using some criterion. For example, in [9], if the argument $\mathcal{A}_2$ is preferred to $\mathcal{A}_1$ w.r.t. the comparison criterion, then $\mathcal{A}_2$ prevails and it will be called a *proper defeater* for $\mathcal{A}_1$. If $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$, then $\mathcal{A}_2$ will not be considered as a defeater for $\mathcal{A}_1$, and $\mathcal{A}_1$ prevails. If neither argument is preferred over the other, a blocking situation occurs, and we will say that $\mathcal{A}_2$ is a *blocking defeater* for $\mathcal{A}_1$. In a blocking defeater situation between $\mathcal{A}_2$ and $\mathcal{A}_1$, both arguments are defeated.

A query is a literal Q, the set of all possible queries will be denoted $\mathbb{Q}$. In [10], several contextual queries were defined. These types of queries allow the inclusion of private pieces of information related to the agents' particular context to be taken into consideration at the moment of computing the answers. In DeLP a query Q is warranted from a program $\mathcal{P}$ if there exists a non-defeated argument $\mathcal{A}$ supporting Q. To establish whether an argument $\mathcal{A}$ is a non-defeated argument, defeaters for $\mathcal{A}$ are considered. In turn, each defeater could be defeated, generating a sequence of arguments called *argumentation line*.

To establish if the argument $\mathcal{A}$ is non-defeated, it is necessary to analyze all argumentation lines that have $\mathcal{A}$ as its first element. For each argument may there exist more a defeater; the presence of multiple defeaters for an argument produces an argumentation lines ramification, giving rise a defeaters tree which is called *dialectical tree*. Each path from the root to a leaf corresponds to one different acceptable argumentation line. To establish the state of the argument in the root, the dialectical tree is marked. Marking of a dialectical tree is a process which will be done by making every node from the leaf to root. Leaf nodes in a dialectical tree will be marked as "*U*", an inner node will be marked as "*D*" iff it has at least a child marked as "*U*", and an inner node will be marked as "*U*" iff each of its children is marked as "*D*".

The process of argumentation finishes when DeLP-server returns an answer. The answer for a query Q from a DeLP-program $\mathcal{P}$ is either: YES, if Q is warranted from P; NO, if the complement of Q is warranted from $\mathcal{P}$; UNDECIDED, if neither Q nor its complement are warranted from $\mathcal{P}$; or UNKNOWN, if Q is not in the language of the program $\mathcal{P}$.

3 Argumentative reasoning with multiple preferences

As we have stated, our focus of research here is formalizing a server model with multiple comparison criteria. In first place, we will provide a conceptual guide to

address this issue by means of what we call a preference-based reasoning server, or $\mathcal{PRS}$ for short. The proposed reasoning server will be integrated by five components: a DeLP-program, a set of DeLP operators, a DeLP-interpreter, a set of preference criteria, and a set of operators to combine preference criteria. The formal definition of $\mathcal{PRS}$ will be introduced after the definition of its components. Figure 1 shows the graphical representation of the proposed server.

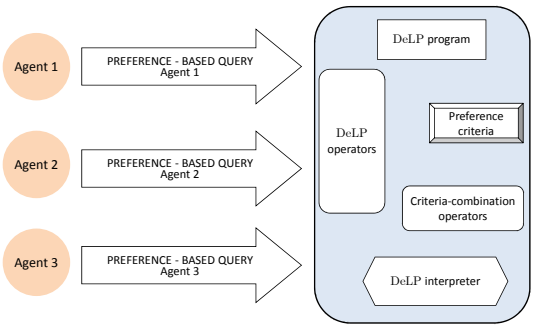

Fig. 1. Preference-based reasoning server.

The figure depicts three client agents sending a contextual query, and the main components of a preference-based reasoning server. These components will be defined and explained below.

As we have said, given two arguments $\mathcal{A}_1$ and $\mathcal{A}_2$ in conflict it is necessary to use a preference criterion to decide which argument prevails and if $\mathcal{A}_1$ is the prevailing argument, $\mathcal{A}_1$ is said to be a defeater of $\mathcal{A}_2$. We will denote a preference criterion with the letter C. A $\mathcal{PRS}$ will be integrated by a set of preference criteria $\mathsf{S} = \{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_n\}$. Continuing with our running example, we assume the criterion $\mathsf{C}_{comfort}$ that favors "comfort" and the criterion C_{price} that favors "price". For clarity purposes, these criteria are applied in a particular application domain as showed in Section 4.

A preference criteria can be represented by a preference relation or a function. For our convenience, we say that, given a set de arguments $Args$, a preference criterion is a function $\mathsf{C} : Args \times Args \longrightarrow \{\bot, \top\}$, obtaining $\top$ when the first argument is preferred over the second, and $\bot$ otherwise.

Usually, existing reasoning services based on defeasible argumentation make use of a unique preference criterion that is an integral part of the inference mechanism. A distinctive feature of our proposed server, is the capacity of combining multiple preference criteria. To achieve this, the server will have available specific

operators to combine different criteria. Thus, we represent a criteria-combination operator as θ^n where n represents its arity. Next, some examples of possible operators will be introduced.

Example 5 *Given two conflicting arguments $\mathcal{A}_1, \mathcal{A}_2 \in Args$ and the preference criteria $\mathsf{C}_1, \mathsf{C}_2 \in \mathsf{S}$. Consider the following three binary operators applied to the criteria $\mathsf{C}_1, \mathsf{C}_2$:*

- *the operator $\boxtimes$ is such that the expression $\mathsf{C}_1 \boxtimes \mathsf{C}_2$ says the argument $\mathcal{A}_1$ prevails iff $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$ for both preference criteria.*
- *the operator $\boxminus$ is such that the expression $\mathsf{C}_1 \boxminus \mathsf{C}_2$ establishes that the argument $\mathcal{A}_1$ prevails iff $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$ for at least one criterion.*
- *the operator $\boxdot$ is such that the expression $\mathsf{C}_1 \boxdot \mathsf{C}_2$ says the argument $\mathcal{A}_1$ prevails iff $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$ with respect to C_1 and if not, it then checks if $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$ with respect to C_2. That is, it returns the same result to $\mathsf{C}_1 \boxminus \mathsf{C}_2$ but the order of evaluation is fixed and must be done by the server in such order: first C_1 and then C_2.*

These operators will be formally defined below.

We have introduced three possible operators, nevertheless, the available operators will depend on the particular reasoning server. It is also possible to define new operators that could have a particular behavior related to a specific application; for instance, some operators could requiere sets of preference criteria, meanwhile others can be used to enable or disable criteria. Moreover, depending on their properties, these operators may be combined leading to more complex expressions.

A contextual query has the particularity of including the client's own information and that information will be used by the server to compute an answer. However, if the server uses the criteria chosen by a client agent, then it will be necessary to adapt the structure of the context query to include them. The change consists in expanding the contextual query with an expression indicating to the server how to solve the query using the criteria selected by the client agents.

A server will answer a query as long as the preference criteria and the criteria-combination operators indicated by the client are part of a criteria-combination expression, or *cc-exp* for short. We use $\mathbb{E}$ to denote the set of all possible criteria-combination expressions.

Definition 1 (Criteria-combination expression) *Let S a set of preference criteria and Θ a set of criteria-combination operators. An expression E is a cc-exp iff:*

- $E \in \mathsf{S}$ *or*
- $E = \theta^n(E_1, E_2, \ldots, E_n)$ *where E_i is a cc-exp and $\theta^n \in \Theta$ with arity-n.* $(1 < i < n)$

In an expression could arise two situations, either the expression is a preference criterion, or the expression is an operator applied to a set of *cc-exp*.

Example 6 *Consider the operators of Example 5. Given two conflicting arguments $\mathcal{A}_1, \mathcal{A}_2 \in Args$ and a set of preference criteria $\mathsf{S}_1 = \{\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3, \mathsf{C}_4\}$. Two possible case of criteria-combination expressions will be presented bellow.*

$E_1 = ((\mathsf{C}_1 \boxminus \mathsf{C}_2) \boxtimes \mathsf{C}_3)$.
$E_2 = ((\mathsf{C}_1 \boxtimes \mathsf{C}_2) \boxdot (\mathsf{C}_3 \boxtimes \mathsf{C}_4))$

In E_1, we will say that the argument $\mathcal{A}_1$ prevails over $\mathcal{A}_2$ iff it is preferred by the criterion C_3 and at least one of the rest of the criteria. In E_1, the argument $\mathcal{A}_1$ prevails over $\mathcal{A}_2$ iff it is preferred by both criteria, C_1 and C_2, or else by the criteria C_3 and C_4.

Example 7 *Consider Example 1 and the operators introduced in Example 5. The criteria to select a room for Joan and Michael, respectively, will be represented by means of the following criteria-combination expressions:*

$E_{Joan} = (\mathsf{C}_{comfort} \boxtimes \mathsf{C}_{price})$
$E_{Michael} = (\mathsf{C}_{comfort} \boxminus \mathsf{C}_{price})$

Thus, the client agents can indicate how their queries have to be solved by the server. For that reason, the *cc-exp* denoting how the client wants to use the server preference criteria, will be included in the queries. This new type of contextual query will be called *preference-based query.*

Definition 2 (Preference-based query) *A preference-based query* PQ *is a tuple* $[\mathbf{C}, E, Q]$ *where* $\mathbf{C}$ *is a particular context for* PQ*, E is a cc-exp, and Q is a query.*

Example 8 *Going back to Example 2 and considering Example 7. Given the query "junior_suite", two preference based queries can be built:*

$[\mathbf{C}_{Joan}, E_{Joan}, junior\_suite]$
$[\mathbf{C}_{Michael}, E_{Michael}, junior\_suite]$

The criteria-combination expressions are solved by the inference mechanism. In particular, the DeLP-interpreter of a $\mathcal{PRS}$ will be responsible of the processing and answering of client queries. As defined next, a DeLP-interpreter will be represented, in general, as a function such that given a program and a query, returns the corresponding answer.

Definition 3 (DeLP-interpreter) *Let $\mathbb{P}$ be the set of valid DeLP-programs, $\mathbb{E}$ be the set of possible cc-exps and $\mathbb{Q}$ be the set of possible queries. A DeLP-interpreter is a function $\mathcal{I} : \mathbb{P} \times \mathbb{E} \times \mathbb{Q} \longrightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of possible answers for $\mathcal{PRS}$, i.e., $\mathbb{R} = \{$*NO, YES, UNDECIDED, UNKNOWN*$\}$.*

Given two conflicting arguments $\mathcal{A}_1, \mathcal{A}_2 \in Args$ and a *cc-exp* E. To solve a *cc-exp* the interpreter will use a function $eval(E, \mathcal{A}_2, \mathcal{A}_1)$ such that its range is $\{\bot, \top\}$ which correspond to the answers for a *cc-exp* given.

The application pattern of the preference criteria is established when preferences combination operators are defined. For instance, consider the set of criteria-combination operators $\Theta_j = \{\boxtimes, \boxminus, \boxdot\}$ presented in Example 5 and two *cc-exp*s E_i and E_j. The evaluation of each operator belonging to the set Θ_j may be defined as:

i) $eval(E, \mathcal{A}_2, \mathcal{A}_1) = \mathsf{C}(\mathcal{A}_2, \mathcal{A}_1)$ if $E = \mathsf{C}$, or

i) $eval(E_i \boxtimes E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$ if $eval(E_i, \mathcal{A}_2, \mathcal{A}_1) = \top$ and $eval(E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$, or

ii) $eval(E_i \boxminus E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$ if $eval(E_i, \mathcal{A}_2, \mathcal{A}_1) = \top$ or $eval(E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$, or

iii) $eval(E_i \boxdot E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$ if $eval(E_i, \mathcal{A}_2, \mathcal{A}_1) = \top$ or else $eval(E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$, or

iv) $\bot$ in other case.

Now we formally present the concept of preference-based reasoning server.

Definition 4 (Preference-based reasoning server) *A Preference-based reasoning server is a 5-tuple $\mathcal{PRS} = \langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \mathsf{S}, \Theta \rangle$, where $\mathcal{I}$ is a DeLP-interpreter, $\mathcal{O}$ is a set of DeLP-operators, $\mathcal{P}$ is a DeLP-program, S is a set of preference criteria, and Θ is a set of criteria-combination operators.*

In Section 2 we stated that the focus of this paper is not on DeLP-operators; we refer the interested reader to [10] for the details of these operators. Here our approach is centred in the criteria-combination operators.

Definition 5 (Answer for a query) *Let $\mathcal{PRS} = \langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \mathsf{S}, \Theta \rangle$ be a preference-based reasoning service,* PQ $= [\mathbf{C}, E, Q]$ *be a preference-based query for $\mathcal{PRS}$, and $\mathcal{P}'$ be a modified program for the context $\mathbf{C}$, i.e., $\mathcal{P}' = \mathcal{P} \cup \mathbf{C}$. An answer for* PQ *from $\mathcal{PRS}$, denoted $Ans(\mathcal{PRS}, E, Q)$, corresponds to the result of the function $\mathcal{I}(\mathcal{P}', E, Q)$.*

4 Application example

In this section we will show in an example in DeLP how the answer to a query can vary according to the way in which the criteria used by the server are combined. Let $\mathcal{P}_h$ and $\mathcal{P}_c$ be the DeLP-programs presented in Example 2;

$$\mathcal{P}_h = \left\{ \begin{array}{l} \sim junior\_suite \prec travel\_alone, \sim jacuzzi. \\ junior\_suite \prec travel\_alone, safe, fridge. \\ simgle\_room \prec short\_stay, \sim jacuzzi. \\ \sim simgle\_room \prec short\_stay, fridge. \\ \sim simgle\_room \prec junior\_suite. \end{array} \right\}$$

$$\mathbf{C}_{Joan} = \mathbf{C}_{Michael} = \mathcal{P}_c = \left\{ \begin{array}{l} travel\_alone. \\ short\_stay. \\ fridge. \\ safe. \end{array} \right\}$$

Consider the preference-based queries introduced in Example 8;

1. $[\mathbf{C}_{Joan}, E_{Joan}, junior\_suite]$.
2. $[\mathbf{C}_{Michael}, E_{Michael}, junior\_suite]$.

such that

$E_{Joan} = (\mathsf{C}_{comfort} \boxtimes \mathsf{C}_{price})$.
$E_{Michael} = (\mathsf{C}_{comfort} \boxminus \mathsf{C}_{price})$.

As state above, we consider the criterion $\mathsf{C}_{comfort}$ that favors "comfort" and the criterion C_{price} that favors "price"; on the one hand, we assume that the function $\mathsf{C}_{comfort}$ will return $\top$ iff the first argument has as information that a bedroom has fridge and safe, and $\bot$ otherwise. On the other hand, the function C_{price} will return $\top$ iff the first argument has as information that the room has not jacuzzi, and $\bot$ otherwise.

In both queries mentioned above, the same DeLP $\mathcal{P}_m$ presented in Example 3 is obtained. As showed in Example 4, from the program $\mathcal{P}_m$ two arguments can be built: the argument $\mathcal{A}$ in favor of recommending junior suite.

$\mathcal{A} = \langle\, junior\_suite \prec travel\_alone, safe, fridge. \,\rangle$

and the argument $\mathcal{B}$ in favor of not recommend junior suite.

$\mathcal{B} = \langle\, \sim junior\_suite \prec travel\_alone, \sim jacuzzi. \,\rangle$

Note that $\mathcal{A}$ and $\mathcal{B}$ are in conflict since they support contradictory conclusions. To answer the query, it is necessary to determine which one prevails. For this, we will use the function "*eval*". Given the *cc-exp*s E_{Joan} and $E_{Michael}$ from example 7, the possible results of comparing the arguments $\mathcal{A}$ and $\mathcal{B}$ using the criteria $\mathsf{C}_{comfort}$ and C_{price} will be showed bellow:

- if "*comfort*" is used as criteria, then the function $\mathsf{C}_{comfort}(\mathcal{A}, \mathcal{B}) = \top$.
- if "*comfort*" is used as criteria, then the function $\mathsf{C}_{comfort}(\mathcal{B}, \mathcal{A}) = \bot$.
- if "*price*" is used as criteria, then the function $\mathsf{C}_{price}(\mathcal{B}, \mathcal{A}) = \top$.
- if "*price*" is used as criteria, then the function $\mathsf{C}_{price}(\mathcal{A}, \mathcal{B}) = \bot$.

Now that the results of comparing the arguments $\mathcal{A}$ and $\mathcal{B}$ are known, so the results of the function "*eval*" for expressions E_{Joan} and $E_{Michael}$ can be obtained, in that case $eval(E_{Joan}, \mathcal{B}, \mathcal{A}) = \bot$, $eval(E_{Michel}, \mathcal{B}, \mathcal{A}) = \top$ and $eval(E_{Michel}, \mathcal{A}, \mathcal{B}) = \top$.

Now, consider the first preference-based query presented above:

$$[\mathbf{C}_{Joan}, E_{Joan}, junior\_suite]$$

Given that the function "*eval*" for the expression E_{Joan} establishes that $\mathcal{A}$ is preferred to $\mathcal{B}$ then $\mathcal{A}$ will defeat to $\mathcal{B}$. Since there are other counterarguments that could defeat $\mathcal{A}$, then $\mathcal{A}$ remains undefeated. The conclusion $junior\_suite$ is warranted, then the answer for the preference-based query is YES. However, in the following preference based query:

$$[\mathbf{C}_{Michael}, E_{Michael}, junior\_suite]$$

if the whole argumentative process is considered, the answer for the query *junior_suite* is UNDECIDED, *i.e.,* neither the conclusion *junior_suite* nor its complement are warranted.

The complete example shows that the same query with the same context but with different combination of criteria can give different answers. This was our goal.

5 Conclusions. Related and future work

We have presented a model that allows an argumentative reasoning server to handle multiple preference criteria. For this, we formally defined the notion of criteria-combination expressions, which use a set of criteria-combination operators. We have introduced three different operators, showing how these operators are evaluated within the combination expressions; these expressions are used by the client agents when performing queries to server. In our approach, DeLP was proposed for knowledge representation and therefore the DeLP-interpreter is in charge of solve these queries. To solve each conflict between arguments, the DeLP-interpreter uses the function *eval* that determines which argument prevails using the criteria-combination expressions contained in the queries. Thus, queries are answered using public knowledge stored in the server, considering the preference criteria indicated by the clients. In Section 4 an example in DeLP is presented where an agent perform two queries with the same context but with different preference criteria combinations getting different results. With the proposed model we have shown that argument comparison criteria are directly related to the inferences that can be obtained by an agent.

Our approach was in part inspired by [10], where several servers can be created, and knowledge can be shared through them. Nevertheless, both approaches have several differences. In contrast with us, they use a preference criteria embedded in the interpreter, *i.e.,* to answer a query the server is configured to use the same specific criterion. Finally, we provide clients with the possibility of selecting what criteria a server should use to compute the answer for a specific query.

In [3] an approach to handle multiple preference was proposed. To determine the acceptable arguments, the set of preferences is linearly ordered using another preference relation. Their main contribution is to take into account contextual preferences which means that several pre-orderings on the knowledge base may be taken into account together, *i.e.,* preferences which depend upon a particular context. Contextual preferences are given in terms of pre-orderings between beliefs. In contrast with us, they provide a framework where the preferences are ordered, in our framework this situation is a particular case, *i.e.,* depending on the criteria-combination operators defined in the server. On the other hand, several approaches about combination of preference criteria can be found in [12, 4, 13].

As future work we are developing an implementation of a DeLP-server that can dynamically handle multiple preference criteria. We are also interested in studying the properties of the criteria-combination operators to define operators for concrete reasoning servers. Another extension will be to integrate our proposed framework with others argumentative systems similar to DeLP.

References

1. Alsinet, T., Chesñevar, C.I., Godo, L., Simari, G.R.: A logic programming framework for possibilistic argumentation: Formalization and logical properties. Fuzzy Sets and Systems 159(10), 1208–1228 (2008)
2. Amgoud, L., Bonnefon, J.F., Prade, H.: An argumentation-based approach to multiple criteria decision. In: ECSQARU. pp. 269–280 (2005)
3. Amgoud, L., Parsons, S., Perrussel, L.: An argumentation framework based on contextual preferences. In: FAPR'2000. pp. 59–67 (2000)
4. Andréka, H., Ryan, M., Schobbens, P.Y.: Operators and laws for combining preference relations. J. Log. Comput. 12(1), 13–53 (2002)
5. Antoniou, G., Maher, M.J., Billington, D.: Defeasible logic versus logic programming without negation as failure. J. Log. Program. 42(1), 47–57 (2000)
6. Capobianco, M., Chesñevar, C.I., Simari, G.R.: Argumentation and the dynamics of warranted beliefs in changing environments. Autonomous Agents and Multi-Agent Systems 11(2), 127–151 (2005)
7. Capobianco, M., Simari, G.R.: A proposal for making argumentation computationally capable of handling large repositories of uncertain data. In: SUM. pp. 95–110 (2009)
8. Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. 28(4), 427–466 (2003)
9. Garcia, A., Simari, G.: Defeasible logic programming: An argumentative approach. Theory and Practice of Logic Programming (TPLP) 4, 95–138 (2004)
10. García, A.J., Rotstein, N.D., Tucat, M., Simari, G.R.: An argumentative reasoning service for deliberative agents. In: KSEM. pp. 128–139 (2007)
11. Godo, L., Marchioni, E., Pardo, P.: Extending a temporal defeasible argumentation framework with possibilistic weights. In: JELIA. pp. 242–254 (2012)
12. Kaci, S.: Working with Preferences: Less Is More. Cognitive Technologies, Springer (2011)
13. Kießling, W.: Foundations of preferences in database systems. In: VLDB. pp. 311–322 (2002)
14. Konstan, J.A.: Introduction to recommender systems: Algorithms and evaluation. ACM Trans. Inf. Syst. 22(1), 1–4 (2004)
15. Loui, R.P.: Defeat among arguments: a system of defeasible inference. Computational Intelligence 3, 100–106 (1987)
16. Maher, M.J., Rock, A., Antoniou, G., Billington, D., Miller, T.: Efficient defeasible reasoning systems. International Journal on Artificial Intelligence Tools 10(4), 483–501 (2001)
17. Resnick, P., Varian, H.R.: Recommender systems - introduction to the special section. Commun. ACM 40(3), 56–58 (1997)
18. Vreeswijk, G.: Abstract argumentation systems. Artificial Intelligence 90(1-2), 225–279 (1997)