Computational Models of Argument
B. Verheij et al. (Eds.)
IOS Press, 2012

doi:10.3233/978-1-61499-111-3-83

Towards an Argument-based Music Recommender System

Cristian E. BRIGUEZ [a,b], Maximiliano C. D. BUDÁN [a,b], Cristhian A. D. DEAGUSTINI [a,b], Ana G. MAGUITMAN [a,b], Marcela CAPOBIANCO [a,b], and Guillermo R. SIMARI [a]

[a] *Artificial Intelligence Research and Development Laboratory*
Department of Computer Science and Engineering
Universidad Nacional del Sur - Alem 1253, (8000) Bahía Blanca, Buenos Aires
[b] *Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)*

Abstract.
The significance of recommender systems has steadily grown in recent years as they help users to access relevant items from the vast universe of possibilities available these days. However, most of the research in recommenders is based purely on quantitative aspects, *i.e.*, measures of similarity between items or users. In this paper we introduce a novel hybrid approach to refine recommendations achieved by quantitative methods with a qualitative approach based on argumentation, where suggestions are given after considering several arguments in favor or against the recommendations. In order to accomplish this, we use *Defeasible Logic Programming* (DeLP) as the underlying formalism for obtaining recommendations. This approach has a number of advantages over other existing recommendation techniques. In particular, recommendations can be refined at any time by adding new polished rules, and explanations may be provided supporting each recommendation in a way that can be easily understood by the user, by means of the computed arguments.

Keywords. Defeasible Argumentation, Recommender Systems, Qualitative Recommendations

1. Introduction

Recommendation systems are support mechanisms that assist users in their decision-making process while interacting with large or complex information spaces. Most recommender systems are aimed at helping users to deal with the problem of information overload by facilitating access to relevant items [3]. Recommenders attempt to generate a model of the user or user's task and apply diverse heuristics to anticipate what information may be useful to the user. In order to come up with recommendations, conventional recommender systems rely on similarity measures between users or contents, computed on the basis of methods coming either from the information retrieval or the machine learning communities.

Recommender systems adopt mainly two different views to help predict information needs. The first approach is known as user modeling and relies on the use of a profile or model of the users, which can be created by observing users' behavior (e.g., [4]). The

second approach is based on task modeling, where recommendations are based on the context in which the user is immersed (e.g.,[5]).

Two main techniques have typically been used to compute recommendations: content-based and collaborative filtering. Content-based recommenders [7] are driven by the premise that user's preferences tend to persist through time. These recommenders frequently use machine-learning techniques to generate a profile of the active user, typically stored as a list of rated items. In order to determine if a new item is a potentially good recommendation, content-based recommender systems rely on similarity measures between the new items and the rated items stored as part of the user model. Recommender systems based on collaborative filtering [8] relies on the assumption that users' preferences are correlated. These systems maintain a pool of users' profiles associated with items that the users rated in the past. For a given active user, collaborative recommender systems find other similar users whose ratings strongly correlate with the current user. New items not rated by the active user can be presented as suggestions if similar users have rated them highly. A combination of collaborative-filtering and content-based recommendation gives rise to hybrid recommender systems (e.g., [11]). These systems typically generate a model of the active user by monitoring the user behavior or by analyzing user's declared interests or feedback. The generated user model is combined with the user information needs and a request for recommendations is presented to a search engine. In addition, the system maintains a pool of profiles from other users, making possible the application of collaborative filtering to further refine the selected set of recommendations. Although hybrid recommender systems are substantially more effective than the basic content-based and collaborative filtering approaches, existing systems are still limited.

Existing recommender systems cannot deal formally with the defeasible nature of users' preferences in complex environments [6]. Decisions about user preferences are mostly based on heuristics which rely on ranking previous user choices or gathering information from other users with similar interests. Besides, the quantitative approaches adopted by most existing recommender systems do not have a clean underlying model. This makes it hard to provide users with a clear explanation of the factors and procedures that led the system to come up with certain recommendations.

Another problem faced by recommender systems is that modeling the users' preference criteria is not an easy task, since it generally involves incomplete and potentially inconsistent knowledge about the search domain. To solve this, we propose to model the users' preference criteria in terms of a DeLP program [2]. In this way defeasible argumentation can be integrated into existing recommender system technologies, paving the way to solve the problems mentioned above.

2. Defeasible Logic Programming

We have chosen the formalism of Defeasible Logic Programming (DeLP) [2] as the key to integrate argumentation into recommender systems. The language of DeLP is based on the language of logic programming. Standard logic programming concepts (such as signature, variables, functions, etc) are defined in the usual way. Literals are atoms that may be preceded by the symbol "$\sim$" denoting *strict* negation, as in extended logic programming. Facts are simply literals. Strict rules are ordered pairs $L_0 \leftarrow L_1, \ldots, L_n$ whose

first component, L_0, is a literal, and whose second component, $L_1, \ldots, L_n$, is a finite non-empty set of literals. Defeasible rules are ordered pairs $L_0 -\!\prec L_1, \ldots, L_n$ whose first component, L_0, is a literal, and whose second component, $L_1, \ldots, L_n$, is a finite non-empty set of literals. Defeasible rules are used to represent defeasible knowledge (*i.e.*, , tentative information that can be used as long as nothing is posed against it), whereas strict rules are used to represent unbreakable information.

In DeLP, the state of the world is modelled as a *Defeasible Logic Program* (*de.l.p*), essentially a possibly infinite set of facts, strict rules and defeasible rules. In a given *de.l.p* $\mathscr{P}$, the subset of facts and strict rules is referred to as Π, and the subset of defeasible rules as Δ, thus a *de.l.p* $\mathscr{P}$ can also be noted as (Π, Δ). Since the set Π represents non-defeasible information it must be non-contradictory.

Given a *de.l.p* $\mathscr{P}$, a query posed to $\mathscr{P}$ is a ground literal Q which must be supported by an *argument*. We apply this setting in our recommender, so to determine whether a track should be recommended to a certain user or not, our system must compute arguments based on the rules in $\mathscr{P}$. Arguments are built on the basis of a *defeasible derivation* computed by backward chaining applying the usual SLD inference procedure used in logic programming.

Definition 1 Argument structure*[2] Let h be a literal, and $\mathscr{P} = (\Pi, \Delta)$ be a de.l.p. We say that $\langle A, h\rangle$ is an* argument structure *for h, if, and only if, A is a set of defeasible rules from $\mathscr{P}$ (i.e., , $A \subseteq \Delta$), such that: (1) there exists a defeasible derivation for h from $\Pi \cup A$, (2) the set $\Pi \cup A$ is non-contradictory, and (3) A is minimal with respect to set inclusion (i.e., , no $A' \subset A$ satisfies the previous conditions).*

Arguments can attack each other, an argument $\langle A_1, h_1\rangle$ *attacks* $\langle A_2, h_2\rangle$ at h if, and only if, there exists a sub-argument structure $\langle A, h\rangle$ from $\langle A_2, h_2\rangle$ such that $\Pi \cup \{h, h_1\}$ is contradictory. Defeat among arguments is defined combining the attack relation and a preference criterion "$\preceq$". An argument $\langle A_1, h_1\rangle$ *defeats* $\langle A_2, h_2\rangle$ if $\langle A_1, h_1\rangle$ attacks $\langle A_2, h_2\rangle$ at a literal h and $\langle A_1, h_1\rangle \preceq \langle A, h\rangle$ (proper defeater) or $\langle A_1, h_1\rangle$ is unrelated to $\langle A, h\rangle$ (*blocking defeater*).

In order to decide if a partial attack really succeeds, becoming a defeat, one or several comparison criteria must be used, establishing the relative strength of the arguments involved in the attack. This criterion will then solve clash situations between arguments. In our recommender system we have chosen to use a particular instantiation of DeLP preference criteria that combines the following elements:

- *Priorities Among Rules*: a predefined preference order among rules is used to determine which argument prevails. To do this, a partial order relation among rules must be defined. Then, when comparing two arguments, we consider the two rules that have conflicting conclusions, and the winner argument is the one that contains the rule that is preferred.
- *Generalized Specificity* [2]: those arguments that are based on more information or those that support their conclusions more directly are preferred.

Given these criteria, we must describe how they are combined to establish which arguments prevail in an attack situation. Basically, we give the *Priority Among Rules* criterion preponderance over *Generalized Specificity*. In a forthcoming section we will show how this order between the two criteria will enable us to establish a preference relation between the different aspects involved in a recommendation.

Definition 2 (Defeat) *Let $\mathscr{P} = (\Pi, \Delta)$ be a de.l.p. Let $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ be two arguments in $\mathscr{P}$. We say that $\langle A_2, h_2\rangle$* defeats *$\langle A_1, h_1\rangle$ if and only if there exists a subargument $\langle A, h\rangle$ of $\langle A_1, h_1\rangle$ such that $\langle A_2, h_2\rangle$ counter-argues $\langle A_1, h_1\rangle$ at literal h and it holds that:*

1. *$\langle A_2, h_2\rangle$ is preferred by priority to $\langle A, h\rangle$ (proper defeater), or*
2. *$\langle A, h\rangle$ is not preferred by priority to $\langle A_2, h_2\rangle$ and $\langle A_2, h_2\rangle$ is strictly more specific that $\langle A, h\rangle$ (proper defeater), or*
3. *$\langle A_2, h_2\rangle$ is unrelated to $\langle A, h\rangle$ (blocking defeater)*

In DeLP a literal h will be warranted if there exists a non-defeated argument structure $\langle A, h\rangle$. To establish whether $\langle A, h\rangle$ is non-defeated, the set of defeaters for A will be considered. Defeaters are arguments and may in turn be defeated. Thus, a complete dialectical analysis is required to determine which arguments are ultimately accepted. Such analysis results in a tree structure (*dialectical tree*), in which arguments are nodes labeled as undefeated (`U-nodes`) or defeated (`D-nodes`) according to a marking procedure.

Definition 3 Dialectical tree *[2] The* dialectical tree *for an argument $\langle A, h\rangle$, denoted $\mathscr{T}_{\langle A,h\rangle}$, is recursively defined as follows: (1) A single node labeled with an argument $\langle A, h\rangle$ with no defeaters (proper or blocking) is by itself the dialectical tree for $\langle A, h\rangle$; (2) Let $\langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \ldots, \langle A_n, h_n\rangle$ be all the defeaters (proper or blocking) for $\langle A, h\rangle$. The dialectical tree for $\langle A, h\rangle$, $\mathscr{T}_{\langle A,h\rangle}$, is obtained by labeling the root node with $\langle A, h\rangle$, and making this node the parent of the root nodes for the dialectical trees of $\langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \ldots, \langle A_n, h_n\rangle$.*

For the marking procedure we start labeling the leaves as `U-nodes`. Then, for any inner node $\langle A_2, Q_2\rangle$, it will be marked as `U-node` if and only if every child of $\langle A_2, Q_2\rangle$ is marked as a `D-node`. If $\langle A_2, Q_2\rangle$ has at least one child marked as `U-node` then it is marked as a `D-node`. This marking allows us to characterize the set of literals sanctioned by a given *de.l.p*, called *warranted literals*. A literal h is *warranted* if and only if there exists an argument structure $\langle A, h\rangle$ for h, such that the root of its marked dialectical tree $\mathscr{T}^{\star}_{\langle A,h\rangle}$ is a `U-node`.

3. Providing decision-relevant information to the argumentative process: The DBI-DeLP Framework

As we have stated in previous sections, we have used DeLP (and some of its extensions) as the basis to develop a music recommender system. However, if we want our system to make useful recommendations to our users, we cannot rely on information included in the program by manual codification. Instead, we have to feed it with real-world information both about the music tracks that can be recommended and the users for which recommendations are made. So, instead of using encoded facts, we will use relational databases as the source of argument support information. To do this, our recommender is developed using the *Database Integration for Defeasible Logic Programming* (DBI-DeLP) framework [9], which will enable us to use a relational database supported dataset. We will briefly introduce the framework in this section.

Basically, a DBI-DeLP program is a DeLP program extended with information obtained from a database. To do this, tuples in databases are represented as "defeasible

facts" called operative presumptions, which are literals in the form $func(q_1,\ldots,q_m)$–≺true . So a DBI-DeLP program accounts for a DeLP program along with a set Σ of operative presumptions, associated with a number of available databases $D_1 \ldots D_k$. Such operative presumptions are built on demand when the system is trying to solve a query. So, our system could have strict rules such as **child_restricted(Track)** ← **explicit_lyrics(Track)** and defeasible rules such as **track_genre(Track, progressive)** –≺ **performer(Track, genesis)** and dynamically added operative presumptions such as **has_listened(user X, stairway to Heaven)** or **has_listened(user X, whole lotta love)** if we found in our dataset that *User X* has listened to the tracks *Stairway to Heaven* and *Whole Lotta Love*. The dynamic search for operative presumptions in the dataset will be crucial if we want to obtain relevant data from the universe of available data in the dataset.

To start the process of obtaining from databases relevant data to a particular query, we first identify the literal in the query that is being solved. DeLP constructs arguments to solve queries by a backward chaining process. That is, when DeLP is trying to build an argument for a literal L it searches its knowledge base looking for strict rules, facts, presumptions and defeasible rules that have L as its Head, and when it finds them, it tries to prove the literals in the Body. We will call these literals in the Body part of a rule *Function Objectives (FO)*, as they will be the next *targets* of the inference procedure.

All FO are first order predicates in the form $func(p_1,\ldots,p_n)$, where *func* is the predicate's functor and $p_1,\ldots,p_n$ is a list of the predicate's parameters (*e.g.*, for genre(pride, Genre) its functor is genre while "pride" and "Genre" are its parameters). The SLD procedure will try to prove every FO using all available knowledge, *i.e.*, all the rules, facts and presumptions in the DBI-DeLP program; but for the purpose of this section we will focus on how presumptions can be obtained from databases. To do this, we first identify pertinent data-sources, *i.e.*, those databases (and those tables and fields) that are expected to have useful data for the FO. We will say that a data-source (more specifically some tables and fields in a database) is pertinent to a FO if the data-source has the potential to support the FO, *i.e.*, we *may* use some data stored in those tables and fields to prove the literal. Once we have identified the pertinent data-sources, we have to retrieve from them the relevant data for a recommendation. To do this we use a presumption retrieval function (PRF). Generally speaking, the goal of the function is to feed the argumentation process with relevant data obtained from the databases that are available to the system. That way we can discriminate useless information from the dataset based on the content, focusing only on data that is valuable for the particular recommendation under analysis.

4. An argument-based music recommender

The usual approaches for recommendation founded in the literature have a serious drawback [6]: although they are very effective, they often fail at giving users the reasons behind recommendations, thus affecting the users' trust in the results. In qualitative approaches such as the one we will describe in this paper, explanations can accompany recommendations, so the user will know which item is suggested and the reason behind the recommendation. This has a two-fold advantage: the user will have more confidence on the presented result, and also the user can give the system feedback about the process behind the obtained result, as it is a "white-box" approach.

The process of obtaining recommendations in an argument-based system is very different from the usual approach applied by quantitative systems. Nevertheless, they

share the same spirit: they try to somehow establish the similarity between objects or users, and then they use this similarity to make recommendations. The main difference is that while quantitative approaches establish similarity by giving it a numerical measure, in the argumentative-based qualitative approach proposed here we establish similarity using rules that state which characteristics have to be shared between objects or users to be considered similar.

For our recommender we have developed some intuitive postulates that state the conditions that need to be met so a certain track is recommended to a given user. These postulates, which can be easily translated into DeLP rules, are used to express aspects based on the principles of both the collaborative filtering and the content-based approaches, thus making our recommender a hybrid one. The postulates used by the system are:

Postulate	Description
1	A user may like a song if he likes another song by the same artist.
2	A user may dislike a song if he dislikes another song by the same artist.
3	A user may like a song if he likes another song performed by an artist related to this song.
4	A user may dislike a song if he dislikes another song performed by an artist related to this song.
5	A user may like a song if he likes another song with the same musical genre.
6	A user may dislike a song if he dislikes another song with the same musical genre.
7	A user may like a song if the song is liked by a similar user.
8	A user may dislike a song if the song is disliked by a similar user.
9	A user may like a song if the song is liked by a group of similar users.
10	A user may dislike a song if the song is disliked by a group of similar users.
11	A user may like a song if the song is often listened by the rest of the system's users.
12	A user may dislike a song if the song is not often listened by the rest of the system's users.

Figure 1. Postulates

With the presented postulates it is possible to obtain arguments in favor or against the potential recommendations. A problem arising from this situation is how to decide whether to finally recommend the track or not. We have introduced two argument preference criteria that will help to solve this problem. We will use the priority among rules preference criterion to state which postulates prevail over others. This gives us the opportunity to easily find the best combination of postulates by empirical trials. So, we can go from a collaborative filtering based recommender which refines its answers on content-based aspects to the other way around effortlessly; or even go a step further and mix aspects in any arbitrary way we want, just by switching priorities between rules and running tests. The second preference criterion, generalized specificity, is used just for the sake of simplicity, as it will solve conflicts in those cases where we do not want to establish priorities as a rule is used to refine the conditions stated in another one.

The rules represent the cases they model in a very clear, colloquial way that is easy to understand and discuss. This leads to another advantage of our approach: we can model new cases by adding rules to the system at any time, without the need of examining the effects on other rules.

Basically, there are two cases when we want to expand the aspects that the recommender takes into account. On the one hand, we may want to add a rule that refines an existing one, if we discover that certain rule has marked counter-examples, and we know the characteristics that those counter-examples share. On the other hand, we can add an entirely new postulate modeling a relation between data that we have not been taking into account, or if we suddenly have access to data that was previously absent.

5. The role of argumentation in recommendations

As we will show through this section, the use of argumentation allows us to give preponderance to those characteristics we consider that better define similarity by simply stating them as rules in the program, and then establishing an adequate priority among them. Argumentation will also give us the possibility of easily offering explanations to given recommendations as arguments themselves can be presented as explanations, being arguments coherent sets of reasons in favor or against the recommendation.

We have already seen how we can obtain from our dataset the relevant data that will be used to build arguments. As stated in previous sections, we will recommend a certain user to listen to a given track if after considering all arguments in favor and against the recommendation, those arguments in favor prevail. In this section we will introduce how the dialectical process is carried out, and how it is used to make recommendations to users. For space reasons, we will show already computed arguments and state how they interact with each other, without focusing on how arguments have been obtained.

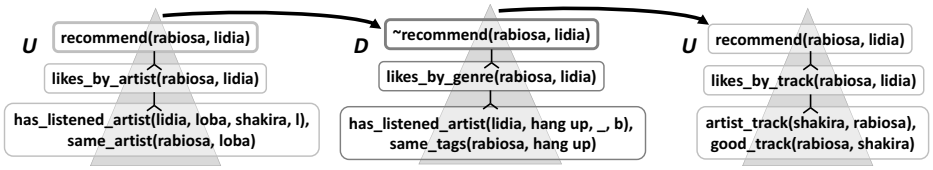

Figure 2. A marked dialectical tree with a warranted root

Figure 2 shows arguments that have been computed to provide a recommendation for the user Lidia. We can see how arguments can be built in favor or against the recommendation under consideration, and then perform an analysis of them. In this case, if we give preponderance to the tracks performed by the same artist over those that share a weaker link (*e.g.*, those that share the musical genre), the track "Rabiosa" is recommended to Lidia because she has listened to the track "Loba", and stated that she likes it, and this track is performed by the same artist as "Rabiosa" (Shakira). So, we have a reason to believe that the user Lidia would like the music track "Rabiosa", and according to our opinion this reason is better than the reason to believe otherwise, *i.e.*, we give that rule priority over the others. Thus, we finally recommend the track to the user.

Notice the way in which we have shown the reason behind the recommendation. We simply have expressed in a slightly more colloquial way the structure of the argument that has prevailed in the dialectical procedure, using the functors of predicates in the rule. This leads to another advantage of the argumentative approach: giving explanations to recommendations is almost straightforward when they are based on arguments, as they can be seen as self-explanatory set of reasons.

6. Conclusions

We have introduced an argumentation-based approach with the capability to improve recommendation technologies. Argumentation can be used to perform a qualitative analysis on users and items. This allows us to go a step forward with respect to the classical quantitative approach to recommendation. In particular, this approach makes it easier to take

several aspects into consideration before we make the final recommendations to the user. These aspects can be applied to model some useful features that are hard to take into account in quantitative approaches. For instance, the defeasible nature of users' preferences in complex environments. The quantitative approaches to recommendation are typically based on the use of a number of parameters, which makes the interpretation of the results non intuitive. In our proposal, it is possible to express in a colloquial way the reason behind each recommendation. Also, we can model restrictions to recommendations based on heterogeneous contextual information (*e.g.*, weather or users' moods).

A useful feature of our formalism is that in order to include a new aspect (postulate) we simply have to state it in the form of one or more DeLP rules, and state the preference assigned to these rules. This rule-based modeling of aspects allows us to make recommendations based on quantitative criteria (if the rules model them), or qualitative criteria. In the meantime, the use of preference leads to another advantage: we can go from content-based approaches to collaborative filtering approaches fairly easily, just by switching priorities between rules. Finally, the use of coherent structures of reasons (arguments) makes giving explanations to recommendations a straightforward process.

Future work may be done in different directions. First, we plan to further test the presented recommendation model, and compare the results obtained by our hybrid recommender system against the ones obtained by a purely quantitative recommender system. Second, we are currently working on an extension of the formalism presented here for modeling context changes through time, to obtain different results for the same query when it is formulated at different time points.

References

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

[2] A. J. García and G. R. Simari. Defeasible Logic Programming: An Argumentative Approach, Theory Practice of Logic Programming, Vol. 4,Num. 1, 95-138, Cambridge University Press, 2004.

[3] P. Maes. Agents that reduce work and information overload. Communications of the ACM 37(7):30-40, 1994.

[4] F. Linton; D. Joy and H. P. Schaefer. Building user and expert models by long-term observation of application usage. In Proceedings of the seventh international conference on User modeling, 129-138. Springer-Verlag New York, Inc, 1999.

[5] J. Budzik; K. J. Hammond and L. Birnbaum. Information access in context. KBS 14(12):37-53, 2001.

[6] C. Chesñevar; A. Maguitman and P. Gonzalez. Empowering Recommendation Technologies through Argumentation. Argumentation in AI. Lecture Notes in Artificial Intelligence. Springer, 2010.

[7] M. J. Pazzani; D. Billsus. Content-Based Recommendation Systems. The Adaptive Web 325-341, 2007.

[8] J. J. Sandvig; B. Mobasher and R. D. Burke. A Survey of Collaborative Recommendation and the Robustness of Model-Based Algorithms. IEEE Data Eng. Bull. (DEBU) 31(2):3-13, 2008.

[9] C. A. D. Deagustini; S. E. Fulladoza Dalibón; S. Gottifredi; M. A. Falappa; C. I. Chesñevar and G. R. Simari. Supporting Defeasible Argumentation Processes over Relational Databases. In Proc. of ArgMAS 2012, Valencia, Spain, June 2012 (accepted).

[10] D. Goldberg; D. Nichols; B. M. Oki and D. Terry. Using collaborative filtering to weave an information tapestry. Communications of the ACM 35(12):61-70, 1992.

[11] M. Claypool; A. Gokhale; T. Miranda; P. Murkinov; D. Netes and M. Sartin. Combining content-based and collaborative filters in an online newspaper. Paper presented at the conference ACM SIGIR Workshop on Recommender Systems: Algorithms and Evaluation. Berkeley, CA, USA, 1999.

[12] M. C. D. Budán; M. Gómez Lucero; C. I. Chesñevar and G. R. Simari. Modeling Time and Reliability in Structured Argumentation Frameworks. In Proc. of KR 2012, Rome, Italy, June 2012 (accepted).

[13] J. S. Lee and J. C. Lee. Context awareness by case-based reasoning in a music recommendation system. In Proceedings of UCS'07, Springer-Verlag, Berlin, Heidelberg, 45-58, 2007.