2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

Scalable Dynamic User Preferences for Recommender Systems through the use of the Well-founded Semantics

Manoela Ilic
Universidade Nova de Lisboa
Portugal

João Leite
Universidade Nova de Lisboa
Portugal

Martin Slota
Universidade Nova de Lisboa
Portugal

Abstract

User modeling and personalisation are the key aspects of recommender systems in terms of recommendation quality. ERASP is an add-on to existing recommender systems which uses dynamic logic programming – an extension of answer set programming – as a means for users to specify and update their models and preferences, with the purpose of enhancing recommendations. While being an excellent solution in recommender systems limited to a few thousand products, ERASP does not scale well beyond that point. In this paper we present a major theoretical redesign of ERASP which entails a significant improvement in the performance of its implementation, making it usable in domains with hundreds of thousands of products.

1 Introduction

In this paper we propose a major change in the theory and implementation of ERASP [21, 19] – an add-on to existing recommender systems which uses dynamic logic programming as a means for users to specify and update their models with the purpose of enhancing recommendations – that significantly improves its time performance widening its usability from domains with a few thousand products to domains with several hundreds of thousand of products.

Currently, almost every e-commerce application provides a recommender system to suggest products or information that the user might want or need [23]. Recommender systems are employed to recommend products in online stores, news articles in news subscription sites or financial services, to mention only a few.

Common techniques for selecting the right item for recommendation are: collaborative filtering (e.g. [18]) where user ratings for objects are used to perform an inter-user comparison and then propose the best rated items; content-based recommendation (e.g.[9]) where descriptions of the content of items are matched against user profiles, employing techniques from the information retrieval field; knowledge-based recommendation (e.g. [10]) where knowledge about the user, the objects, and some distance measures between them are used to infer the right selections; and hybrid versions of these where two or more techniques (collaborative filtering being usually one of them) are used to overcome their individual limitations. For further details on this subject the reader is referred to [11].

The extent to which users find the recommendations satisfactory is the key feature of a recommendation system, and the accuracy of the user models that are employed is of significant importance to this goal. Such user models represent the user's taste and can be implicit (e.g. constructed from information about the user behavior), or explicit (e.g. constructed from direct feedback or input by the user, like ratings). The accuracy of a user model greatly depends on how well short-term and long-term interests are represented [8], making it a challenging task to include both sensibility to changes of taste and maintenance of permanent preferences. While implicit user modeling disburdens the user of providing direct feedback, explicit user modeling may be more confidence inspiring to the user since recommendations are based on a conscious assignment of preferences.

Though most recommender systems are very efficient from a large-scale perspective, the effort in user involvement and interaction is calling for more attention. Moreover, problems concerning trust and security in recommender systems could be approached with a better integration of the user and more control over the user model [20].

This calls for more expressive ways for users to express their wishes. The natural way to approach this is through the use of symbolic knowledge representation languages. They provide the necessary tools for representing and reasoning about users, while providing formal semantics that make it possible to reason about the system, thus facilitating trust and security management.

However, we want to keep the advantages of the more automated recommendation techniques such as collaborative filtering and statistical analysis, and the benefit of using large amounts of data collected over the years by existing recommender systems that use these techniques. Un-

978-0-7695-3496-1/08 $25.00 © 2008 IEEE
DOI 10.1109/WIIAT.2008.122

fortunately, the use of these methods makes it impossible to embed in them the explicit user models we seek. A tight combination between expressive (logic based) knowledge representation languages and sub-symbolic/statistical approaches is still the Holy Grail of Artificial Intelligence.

One solution to tackle this problem is through the use of a layered architecture, as proposed in [21], where expressive knowledge based user models, specified in *Dynamic Logic Programming* (DLP) [3, 22, 2], are used to enhance the recommendations provided by existing recommender systems.

In a nutshell, DLP is an extension of Answer-set Programming (ASP) [17] introduced to deal with knowledge updates. ASP is a form of declarative programming that is similar in syntax to traditional logic programming and close in semantics to non-monotonic logic, that is particularly suited for knowledge representation. Enormous progress concerning the theoretical foundations of ASP (c.f. [7] for more) have been made in recent years, and the existence of very efficient ASP solvers (e.g. DLV[1] and SMODELS[2] make it possible to investigate some serious applications. Whereas in ASP knowledge is specified in a single theory, in DLP knowledge is given by a sequence of theories, each representing an update to the previous ones. The declarative semantics of DLP ensures that any contradictions that arise due to the updates are properly handled. Intuitively, one can add newer rules to the end of the sequence and DLP automatically ensures that these rules are in force and that the older rules are kept for as long as they are not in conflict with the newly added ones (c.f. [22] for more).

ERASP (Enhancing Recommendations with Answer-Set Programming) is the system that resulted from following this path. Specifically, ERASP takes the output of an existing recommender algorithm (we used collaborative filtering, but it could be another) and enhances it taking into account explicit models and preferences specified both by the user and the owner of the system, represented in DLP. The main features of ERASP include:

– Providing owner and user with a simple, expressive and extensible language to specify models and preferences, by means of rules and employing existing (e.g. product characteristics) as well as user defined (e.g. own qualitative classifications based on product characteristics) concepts.

– Facilitating the update of user models by automatically detecting and solving contradictions that arise due to the evolution of the user's tastes and needs, which otherwise would discourage system usage.

– Taking advantage of existing recommender systems which may encode large amounts of data that should not be disregarded, particularly useful in the absence of user specified knowledge, while giving precedence to user specifications which, if violated, would turn the user away from the recommendation system.

– Enjoying a formal semantics which allows to study the properties of the system and provides support for explanations, thus improving interaction with the user.

– Having a connection with relational databases (ASP can be seen as a query language, more expressive than SQL), easing integration with existing systems[3].

– Allowing the use of the combination of both strong and default negation to reason with the closed and open world assumptions, thus allowing for reasoning with incomplete information, and to encode non-deterministic choice, thus generating more than one set of recommendations, facilitating diversity each time the system is invoked;

In [19], benchmark results indicate that ERASP can perform well when the number of products is restricted to a few thousand. As the number of products increases, the time to compute the enhanced recommendations increases substantially. This is not surprising since ERASP is based on Answer-Set Programming which is in the NP class of complexity. Despite most current ASP solvers (e.g. SMODELS and DLV) being very efficient, the original system will never scale well. For those applications that do not employ more than a few thousand products ERASP is an excellent solution. But we need to find alternatives to widen the scope of applicability of ERASP by reducing complexity.

In this paper we explore one solution to this problem. Instead of using a semantics based on Answer-Set Programming, we employ one based on the Well-Founded Semantics for Dynamic Logic Programming [16, 5]. The Well Founded Semantics lies in the polynomial class of complexity making it more scalable. Like for ASP, there are very efficient systems to compute the WFS, notably XSB[4]. The price to pay for such change is a reduction in expressivity which, in our application, will mean that there will no longer be the possibility to encode non-deterministic choice. In other words, whereas in the original system each input could produce several sets of recommended products, the new version will produce only one i.e. if the system is twice invoked by some user without changes in the input (i.e. output from the initial recommender system, user preferences, owner preferences and database), the original system could output a different set of recommendations whereas the new version will produce the same. Given all qualities of ERASP, we feel that this is a small price to pay, especially when we realise that this dramatically increases scalability, making it cope with hundreds of thousands of products where the original could only cope with a few thousand products.

[1] http://www.dlvsystem.com

[2] http://www.tcs.hut.fi/Software/smodels

[3] Recent developments have introduced an extension of Logic Programming for Non-monotonic reasoning that allows for the interface with ontologies [14], making our ERASP easily extensible to the case where product information is available in the Semantic Web instead.

[4] http://xsb.sourceforge.net

The remainder of this paper is organised as follows: In Sect. 2, for self containment, we recap the notion of Dynamic Logic Programming, establishing the language and semantics used throughout. In Sect. 3 we present ERASP architecture, semantics (both the original and the new versions) and implementation. In Sect. 4 we present a short illustrative example used for benchmarking the system and comparing with the original. In Sect. 5 we discuss the results and conclude.

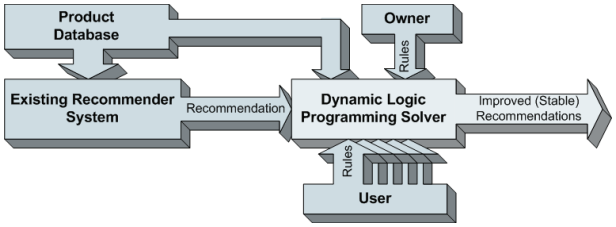

Figure 1. ERASP System Architecture

2 Dynamic Logic Programming

Let $\mathcal{A}$ be a set of propositional atoms. An **objective literal** is either an atom A or a strongly negated atom $\neg A$. A **default literal** is an objective literal preceded by not. A **literal** is either an objective literal or a default literal. A **rule** r is an ordered pair $H(r) \leftarrow B(r)$ where $H(r)$ (dubbed the head of the rule) is a literal and $B(r)$ (dubbed the body of the rule) is a finite set of literals. A rule with $H(r) = L_0$ and $B(r) = \{L_1, \ldots, L_n\}$ will simply be written as $L_0 \leftarrow L_1, \ldots, L_n$. A **generalised logic program** (GLP) P, in $\mathcal{A}$, is a finite or infinite set of rules. If $H(r) = \neg A$ (resp. $H(r) = not\, A$), then $\neg H(r) = A$ (resp. $not\, H(r) = A$). By the **expanded generalised logic program** corresponding to the GLP P, denoted by $\mathbf{P}$, we mean the GLP obtained by augmenting P with a rule of the form $not\, \neg H(r) \leftarrow B(r)$ for every rule, in P, of the form $H(r) \leftarrow B(r)$, where $H(r)$ is an objective literal. An **interpretation** M of $\mathcal{A}$ is a set of objective literals that is consistent, i.e. M does not contain both A and $\neg A$. An objective literal L is true in M, denoted by $M \models L$, iff $L \in M$, and false otherwise. A default literal $not\, L$ is true in M, denoted by $M \models not\, L$, iff $L \notin M$, and false otherwise. A set of literals B is true in M, denoted by $M \models B$, iff each literal in B is true in M. Let $least(.)$ denote the least model of the argument program while treating all default literals as new atoms. A **dynamic logic program** (DLP) is a sequence of generalised logic programs. Let $\mathcal{P} = (P_1, \ldots, P_n)$ be a DLP and P, P' be GLPs. We use $\rho(\mathcal{P})$ to denote the multiset of all rules appearing in the programs $\mathbf{P}_1, \ldots, \mathbf{P}_n$ and $(P, P', \mathcal{P})$ to denote the DLP $(P, P', P_1, \ldots, P_n)$. We can now set forth the definition of a semantics, *based on causal rejection of rules*, for DLPs. We start with the notion of conflicting rules: two rules r and r' are **conflicting**, denoted by $r \bowtie r'$, iff $H(r) = not\, H(r')$. Further we define:

$$Rej(\mathcal{P}, M) = \{r \mid r \in \mathbf{P}_i, \exists \overline{r} \in \mathbf{P}_j, i < j, r \bowtie \overline{r}, M \models B(\overline{r})\}$$
$$Rej^S(\mathcal{P}, M) = \{r \mid r \in \mathbf{P}_i, \exists \overline{r} \in \mathbf{P}_j, i \leq j, r \bowtie \overline{r}, M \models B(\overline{r})\}$$
$$Def(\mathcal{P}, M) = \{not\, A \mid \nexists r \in \rho(\mathcal{P}), H(r) = A, M \models B(r)\}$$

$$\Gamma_{\mathcal{P}} = least([\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M)] \cup Def(\mathcal{P}, M))$$
$$\Gamma^S_{\mathcal{P}} = least([\rho(\mathcal{P}) \setminus Rej^S(\mathcal{P}, M)] \cup Def(\mathcal{P}, M))$$

An interpretation M is a **(refined) dynamic stable model** of $\mathcal{P}$ iff it is a fixpoint of the $\Gamma^S_{\mathcal{P}}$ operator. An interpretation M is a **well-founded model** of $\mathcal{P}$ iff it is the (set inclusion) least fixpoint of the $\Gamma_{\mathcal{P}}\Gamma^S_{\mathcal{P}}$ operator[5].

3 Framework and its Implementation

In this Section, we introduce the architecture, its semantics and describe the implementation. ERASP's goal is to take the strengths of DLP as a framework for the representation of evolving knowledge, and put it at the service of both the user and owner of a recommender system, while at the same time ensuring some degree of integration with other recommendation modules, possibly based on distinct paradigms (e.g. statistical).

Fig. 1 depicts the system architecture, representing the information flow. To facilitate presentation, we assume a layered system where the output of an existing recommendation module is simply used as input to our system. We are aware that allowing for feedback from our system to the existing module could benefit its output, but such process would greatly depend on the particular module and we want to keep our proposal as general as possible, and concentrate on other aspects of the framework. The output of the initial module is assumed to be an interpretation, i.e. a consistent set of atoms representing the recommendations. We assume that our language contains a reserved predicate of the form $rec/1$ where the items are the terms of the predicate[6]. The owner policy, possibly used to encode desired marketing strategies (e.g. introduce some bias towards some products), is encoded as a generalised logic program. The user model (including its updates) is encoded as a dynamic logic

[5] We use the notation $\Gamma_{\mathcal{P}}\Gamma^S_{\mathcal{P}}$ to denote the operator obtained by first applying $\Gamma^S_{\mathcal{P}}$ and then $\Gamma_{\mathcal{P}}$.

[6] It would be straightforward to also have some value associated with each recommendation, e.g. by using a predicate of the form $rec(item, value)$. However, to get our point across, we will keep to the simplified version.

program. The Product Database is a relational database that can easily be represented by a set of facts in a logic program. For simplicity, we assume such database to be part of the generalised logic program representing the owner's policy. A formalization of the system is given by the concept of Dynamic Recommender Frame:

Definition 1 (Dynamic Recommender Frame) *Let $\mathcal{A}$ be a set of propositional atoms. A Dynamic Recommender Frame (DRF), over $\mathcal{A}$, is a triple $\langle M, P_0, \mathcal{P}\rangle$ where M is an interpretation of $\mathcal{A}$, P_0 a generalised logic program over $\mathcal{A}$, and $\mathcal{P}$ a DLP over $\mathcal{A}$.*

The semantics of a Dynamic Recommender Frame is given by the set of dynamic stable models of its transformation into a DLP. This transformation is based on two natural principles: – the user's specification should prevail over both the initial recommendations and the owner's rules, since users would not accept a recommendation system that explicitly violates their rules; – the owner should be able to override the recommendations in the initial interpretation, e.g. to specify preference among products according to the profit. Intuitively, we construct a DLP with the initial program obtained from the initial recommendations, which is then updated with the owner's policy specification (P_0) and the user's specification ($\mathcal{P}$). A formal definition follows:

Definition 2 (Recommendation Semantics) *Let $\mathcal{R} = \langle M, P_0, \mathcal{P}\rangle$ be a Dynamic Recommender Frame, $\mathcal{P}_R$ be the dynamic logic program $(P_M, P_0, \mathcal{P})$ where $P_M = \{A \leftarrow | \ A \in M\}$ and M_R be an interpretation. M_R is a **Stable Recommendation** of $\mathcal{R}$ iff M_R is a dynamic stable model of $\mathcal{P}_R$. M_R is a **Well-founded Recommendation** of $\mathcal{R}$ iff M_R is a well-founded model of $\mathcal{P}_R$.*

According to this semantics, each Dynamic Recommender Frame can have several Stable Recommendations but exactly one Well-founded Recommendation. The advantage of Stable Recommendations is that it allows the system to present different recommendations each time the user invokes the system, adding diversity, while the advantage of the Well-founded Recommendation is that it always exists.

ERASP is implemented as an online application[7] using a PHP-based initial collaborative filtering recommender system. The product database consists of the complete MovieLens (http://www.grouplens.org/) dataset (3883 movies with title, genre and year). After rating some movies and receiving some initial recommendations (using the collaborative filtering algorithm), the user can edit his preferences encoded as a dynamic logic program using an interface which provides help in rule creation. This program $\mathcal{P}$, the initial recommendations M, the product database and the owner specifications P_0 are given to a DLP solver which computes

[7] Available at `http://centria.di.fct.unl.pt/erasp/`

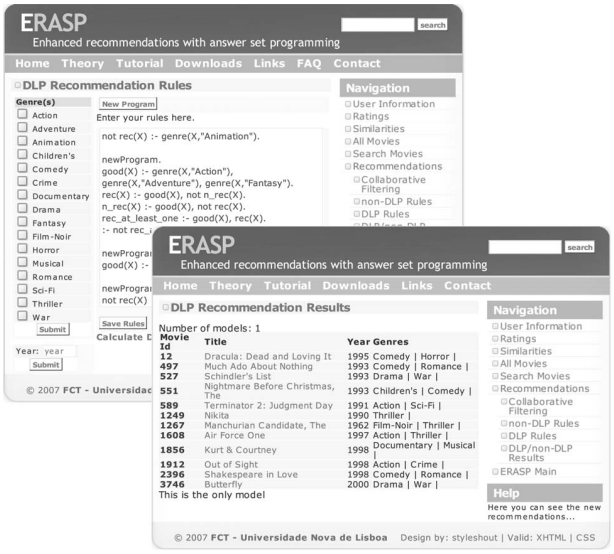

Figure 2. ERASP User Model Interface and Recommendations

the recommendations. Our program then writes the recommendations into an SQL database and presents them to the user. Figure 2 shows two screen-shots of ERASP.

We implemented both the Stable Recommendation semantics and Well-founded Recommendation semantics. Both solvers are based on a transformation that convert a dynamic logic program into an equivalent normal logic program [6]. The input is parsed and the dynamic logic program $\mathcal{P}_R$ is created. In case of the Stable Recommendation semantics, Lparse is used to produce an equivalent DLP $\mathcal{P}_G$ without variables which is further transformed into an equivalent normal logic program $\mathcal{P}_G^R$. Smodels is then used to compute the stable models of $\mathcal{P}_G^R$ which directly correspond to the desired Stable Recommendations.

In case of the Well-founded Recommendation semantics, the grounding phase is not required and the transformation can be performed directly. The well-founded model of the transformed program can then be queried using XSB in order to find the Well-founded Recommendations. The database part of the input is treated separately in order to minimize its loading time. In particular, the database facts are not subject to the transformation. This is only possible because the program doesn't need to be grounded in order to compute its well-founded model.

4 Illustrative Example and Benchmarks

In this Section, we show an example that illustrates some features of our proposal, and present the results of bench-

mark tests based on the example.

Let's consider a typical on-line movie recommender. Its product database contains information about a number of movies and its recommendations are based on some kind of statistical analysis performed over the years. The owner of the recommender system may want to explicitly influence the recommendations of the system in a certain way. She may also want to give the users the ability to specify some explicit information about their tastes in order to correct or refine the recommendations of the existing system. Below we will illustrate how our framework can help achieve both these goals in a simple way. A list of the movies involved in the example together with their relevant properties can be found in Table 1. We will also consider the initial interpretation M obtained from the statistical system to be constant throughout the example:

$$M = \{rec(497), rec(527), rec(551), rec(589), rec(1249), rec(1267), rec(1580), rec(1608), rec(1912), rec(2396)\}$$

We first consider the following owner specification P_0:

$$rec(12) \leftarrow not\, rec(15). \quad (1)$$
$$rec(15) \leftarrow not\, rec(12). \quad (2)$$
$$rec(X) \leftarrow rec(Y), year(Y, 1998), year(X, 1998), genre(Y, \text{“}Romance\text{”}), genre(X, \text{“}Musical\text{”}). \quad (3)$$

Rules (1) and (2) specify that the system should non-deterministically recommend either movie 12 or 15[8]. Rule (3) encodes that the system should recommend all movies with the genre *Musical* from 1998 if any movie with the genre *Romance* from the same year is recommended. Adding an empty set of user specifications $\mathcal{P}_0 = ()$, the recommender frame $\langle M, P_0, \mathcal{P}_0\rangle$ has two stable recommendations: $M_{R1} = M \cup \{rec(12), rec(1856), rec(2394)\}$ and $M_{R2} = M \cup \{rec(15), rec(1856), rec(2394)\}$. The reader can easily check that each of these two stable recommendations extend the results from the initial recommendation to reflect the wishes of the owner. The well-founded recommendation is a skeptical approximation of the stable recommendations: $M_R = M \cup \{rec(1856), rec(2394)\}$. We now turn to the user specifications, assuming four updates:

$$P_1: \quad not\, rec(X) \leftarrow genre(X, \text{“}Animation\text{”}). \quad (4)$$
$$P_2: \quad good(X) \leftarrow genre(X, \text{“}Action\text{”}), genre(X, \text{“}Adventure\text{”}), genre(X, \text{“}Fantasy\text{”}). \quad (5)$$
$$rec(X) \leftarrow good(X). \quad (6)$$
$$P_3: \quad good(X) \leftarrow genre(X, \text{“}War\text{”}), year(X, 2000). \quad (7)$$
$$P_4: \quad not\, rec(X) \leftarrow genre(X, \text{“}Adventure\text{”}). \quad (8)$$

[8]The even loop through default negation is used in ASP to generate two models. In the well-founded semantics, atoms simply become undefined if there isn't any other rule for them

ID	Title	Year	Genres
12	Dracula: Dead and Loving It	1995	Comedy, Horror
15	Cutthroat Island	1995	Action, Adventure
497	Much Ado About Nothing	1993	Comedy, Romance
527	Schindler's List	1993	Drama, War
551	Nightmare Before Christmas	1993	Children's, Comedy
558	Pagemaster, The	1994	Action, Adventure, Fantasy
589	Terminator 2: Judgment Day	1991	Action, Sci-Fi
1249	Nikita	1990	Thriller
1267	Manchurian Candidate, The	1962	Film-Noir, Thriller
1580	Men in Black	1997	Action, Adventure
1608	Air Force One	1997	Action, Thriller
1856	Kurt & Courtney	1998	Documentary, Musical
1912	Out of Sight	1998	Action, Crime
2394	Prince of Egypt, The	1998	Animation, Musical
2396	Shakespeare in Love	1998	Comedy, Romance
3746	Butterfly	2000	Drama, War

Table 1. Some movies used in the example with some of their properties

# of movies	# of facts (S)	Size (S)	# of facts (WF)	Size (WF)
500	1750	38 kB	1251	33 kB
1000	3421	75 kB	2438	65 kB
3883	13523	307 kB	9640	264 kB
200000			700551	21 MB
400000			1400163	42 MB
600000			2100592	63 MB

Table 2. Databases used in the tests

In the single rule (4) of program P_1, the user simply overrides any previous rule that would recommend an *Animation* movie. In program P_2, he introduces the notion of a good movie in rule (5) and rule (6) makes sure all good movies get recommended. Program P_3 extends the definition of a good movie and program P_4 avoids recommendations of *Adventure* movies.

With $\mathcal{P}_1 = (P_1)$, $\mathcal{P}_2 = (P_1, P_2)$, $\mathcal{P}_3 = (P_1, P_2, P_3)$, $\mathcal{P}_4 = (P_1, P_2, P_3, P_4)$, the recommender frames $\langle M, P_0, \mathcal{P}_1\rangle$, $\langle M, P_0, \mathcal{P}_2\rangle$, $\langle M, P_0, \mathcal{P}_3\rangle$ and $\langle M, P_0, \mathcal{P}_4\rangle$ have 2, 2, 2 and 1 stable recommendations, respectively, which, for lack of space we cannot list here. Instead, we list only the final one and invite the reader to see how it complies with the user rules, how contradictory rules are solved (e.g. adventure movies that are good, such as movie 558), as well as those between user and owner rules (e.g. movie 15 is no longer recommended).

$$M = \{rec(12), rec(497), rec(527), rec(551), rec(589), rec(1249), rec(1267), rec(1608), rec(1856), rec(1912), rec(2396), rec(3746)\}$$

We now turn our attention to time performance. To investigate the importance of the size of the input, we tested using the programs specified above with the databases listed in Table 2. The first three databases were used to compute both the stable recommendations and well-founded recommendations in order to compare the two implementations. The table also contains the number of facts in those

DB	DLP	PA	GR	TR	SM	Total
500	$\mathcal{P}_1$	0.0895	0.3077	0.0897	0.2198	0.7067
500	$\mathcal{P}_2$	0.0810	0.3050	0.0890	0.2232	0.6982
500	$\mathcal{P}_3$	0.0836	0.2996	0.0888	0.2259	0.6978
500	$\mathcal{P}_4$	0.0774	0.2948	0.0900	0.2205	0.6827
1000	$\mathcal{P}_1$	0.0973	0.5047	0.1311	0.3311	1.0643
1000	$\mathcal{P}_2$	0.1082	0.5075	0.1325	0.3273	1.0756
1000	$\mathcal{P}_3$	0.1020	0.5120	0.1315	0.3448	1.0902
1000	$\mathcal{P}_4$	0.1008	0.5214	0.1355	0.3398	1.0974
3883	$\mathcal{P}_1$	0.2042	1.9126	0.5022	1.4106	4.0296
3883	$\mathcal{P}_2$	0.2074	1.8990	0.4997	1.4199	4.0260
3883	$\mathcal{P}_3$	0.2080	1.8816	0.5037	1.4564	4.0496
3883	$\mathcal{P}_4$	0.2096	1.8995	0.5039	1.4746	4.0876

Table 3. Stable Recommendations

DB	DLP	Load	TR	Query	Total
500	$\mathcal{P}_1$	0.0130	0.0043	0.0006	0.0179
500	$\mathcal{P}_2$	0.0130	0.0057	0.0011	0.0198
500	$\mathcal{P}_3$	0.0130	0.0060	0.0012	0.0202
500	$\mathcal{P}_4$	0.0130	0.0064	0.0012	0.0206
1000	$\mathcal{P}_1$	0.0240	0.0041	0.0008	0.0289
1000	$\mathcal{P}_2$	0.0240	0.0056	0.0015	0.0311
1000	$\mathcal{P}_3$	0.0240	0.0061	0.0018	0.0318
1000	$\mathcal{P}_4$	0.0240	0.0066	0.0014	0.0320
3883	$\mathcal{P}_1$	0.0950	0.0043	0.0013	0.1005
3883	$\mathcal{P}_2$	0.0950	0.0057	0.0039	0.1046
3883	$\mathcal{P}_3$	0.0950	0.0061	0.0045	0.1055
3883	$\mathcal{P}_4$	0.0950	0.0064	0.0045	0.1059

Table 4. Well-founded Recommendations

databases and the size of the file with the facts. For the well-founded computation, we chose a less redundant encoding of the product database – instead of having the predicates $title(Id, Title)$ and $year(Id, Year)$, we have a single predicate $movie(Id, Title, Year)$ and the original two are added by adding the rules:

$$title(Id, Title) \leftarrow movie(Id, Title, \_).$$
$$year(Id, Year) \leftarrow movid(Id, \_, Year).$$

The same database format change for stable recommendations would have a negative effect since the program is grounded in a later stage of execution and hence the number of atoms processed by the answer set solver would increase instead of decreasing. We also tested the well-founded recommender for larger databases, generated for test purposes, also listed in Table 2. For each database and each recommender frame we computed all recommendations 20 times using an Intel Pentium D 3.4 GHz processor with 2 MB cache and 1 GB of RAM. The average times (in seconds) for each step can be seen in Tables 3, 4 and 5. Table 3 lists the parsing time (PA), grounding time (GR), transformation time (TR), stable model computation time (SM) and total time. Tables 4 and 5 list the database loading time (Load), transformation time (TR), query time (Query) needed to extract the well-founded recommendations and the total time. However, since in the well-founded case, it is possible to load the database separately and then work with it for different queries, and the transformation time is very low, the relevant time for computing the recommendation for one user is the query time, not the total time.

DB	DLP	Load	TR	Query	Total
200000	$\mathcal{P}_1$	6.3570	0.0045	0.5739	6.9354
200000	$\mathcal{P}_2$	6.3570	0.0058	0.8766	7.2394
200000	$\mathcal{P}_3$	6.3570	0.0063	1.1934	7.5566
200000	$\mathcal{P}_4$	6.3570	0.0066	0.9768	7.3403
400000	$\mathcal{P}_1$	12.7370	0.0046	2.0610	14.8025
400000	$\mathcal{P}_2$	12.7370	0.0059	2.8576	15.6005
400000	$\mathcal{P}_3$	12.7370	0.0063	3.8791	16.6225
400000	$\mathcal{P}_4$	12.7370	0.0068	3.0066	15.7504
600000	$\mathcal{P}_1$	19.1010	0.0046	4.8464	23.9520
600000	$\mathcal{P}_2$	19.1010	0.0060	6.5242	25.6312
600000	$\mathcal{P}_3$	19.1010	0.0065	9.2211	28.3633
600000	$\mathcal{P}_4$	19.1010	0.0070	6.4636	25.5716

Table 5. Well-founded Recommendations for larger databases

As can be seen from Table 3, the stable recommendations can be computed in reasonable time for a database up to a few thousand products. However, with these databases, the well founded recommendations can be computed 30-40 times faster as seen in Table 4. Furthermore, query time in Table 5 shows that the well founded version scales well to databases with hundreds of thousands of products. It is also worth noting that the effect of addition of common user rules to the user specification can have both a positive and a negative effect on the query time, depending from the size of product database and the number of recommendations generated by those rules.

5 Concluding Remarks

In this paper we took a closer look at ERASP – a system that can work as an add-on for existing recommender systems – and proposed a major change in its supporting theory, accompanied by a new implementation, to address the lack of scalability to systems that deal with more than a few thousand items. The new system, based on the well founded semantics which rests in the polynomial class of complexity, is able to cope with databases with several hundreds of thousands of products, thus increasing the scope of application to most existing recommender systems.

ERASP will allow owners of existing recommender systems be able to plug in the application as a recommendation enhancer, for offering users the possibility of explicit preference creation, for defining specific system rules, or both. A rule-based language like DLP can empower owners of recommender systems with the necessary tools to employ marketing strategies with precision, while keeping the diversity of recommendations and following user's preferences. Moreover, it can be used as a query tool to extract information from the database using a more sophisticated language than for example SQL, allowing the system to be used on its

own without the need of a previously existing recommender system. Users of ERASP can find a rich language to interact with the recommender system, gaining control, if they so desire, over the recommendations provided.

The new ERASP, like its predecessor, still enjoys a formal declarative semantics thus inheriting many of their theoretical properties. ERASP also enjoys other formally provable properties, e.g. the property of both positive and negative supportiveness which insures there always exists an explanation for each recommended item.

One issue that needs to be tackled is that of the user interface, namely the task of writing rules, burdening for the user [13]. Without addressing this issue, ERASP can still provide added value for experts of specific domains that are willing to learn how to write such rules to satisfy their demand of higher accuracy and, more important, reliability of recommendations. Even for less demanding users, there are still some easy to write rules that provide some basic interaction with the recommender system that is of great help. Dealing with this issue includes transforming natural language sentences into logic programs [15], creating natural language interfaces for databases [4], creating rules by tagging and suggestion and learning rules by induction [1].

As for related work, in [12] Defeasible Logic Programming is employed in the ArgueNet website recommender system. Lack of space prevents us from comparing both architectures. To the best of our knowledge, ArgueNet has not been implemented yet. However, the complexity of Defeasible Logic Programming (NP-hard at least) makes it impossible to compete with the improved ERASP in terms of scalability.

Acknowledgements

We would like thank Paulo Lopes for providing dedicated hardware for running the benchmark tests, and for his reliable technical support and advice. Martin Slota is partially supported by FCT grant SFRH/BD/38214/2007.

References

[1] J. Aitken. Learning information extraction rules: An inductive logic programming approach. In *ECAI'02*. IOS Press, 2002.

[2] J. Alferes, F. Banti, A. Brogi, and J. A. Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1), 2005.

[3] J. Alferes, J. Leite, L. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *J. Logic Programming*, 45(1-3), 2000.

[4] I. Androutsopoulos, G. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.

[5] F. Banti, J. Alferes, and A. Brogi. Well founded semantics for logic program updates. In *IBERAMIA'04*, volume 3315 of *LNCS*. Springer, 2004.

[6] F. Banti, J. Alferes, and A. Brogi. Operational semantics for DyLPs. In *EPIA'05*, volume 3808 of *LNAI*. Springer, 2005.

[7] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge U. Press, 2003.

[8] D. Billsus and M. J. Pazzani. User modeling for adaptive news access. *User Model. User-Adapt. Interact*, 10(2-3):147–180, 2000.

[9] D. Billsus and M. J. Pazzani. Content-based recommendation systems. In *The Adaptive Web*, volume 4321 of *LNCS*, pages 325–341. Springer, 2007.

[10] R. Burke. Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems*, volume 69. M. Dekker, 2000.

[11] R. D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User-Adapt. Interact*, 12(4):331–370, 2002.

[12] C. Chesñevar and A. Maguitman. ArgueNet: An argument-based recommender system for solving web search queries. In *the 2nd. International IEEE Conference on Intelligent Systems*, pages 282–287. IEEE Press, June 2004.

[13] M. Claypool, P. Le, M. Wased, and D. Brown. Implicit interest indicators. In *IUI'01*, 2001.

[14] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In *KR'04*. AAAI Press, 2004.

[15] N. E. Fuchs and R. Schwitter. Specifying logic programs in controlled natural language. *CoRR*, abs/cmp-lg/9507009, 1995.

[16] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.

[17] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *ICLP'90*. MIT Press, 1990.

[18] 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, Dec. 1992. Special Issue on Information Filtering.

[19] M. Ilic, J. Leite, and M. Slota. Explicit dynamic user profiles for a collaborative filtering recommender system. In *IBERAMIA'08*, volume 5290 of *LNAI*. Springer, 2008.

[20] S. K. Lam, D. Frankowski, and J. Riedl. Do you trust your recommendations? An exploration of security and privacy issues in recommender systems. In *ETRICS'06*, 2006.

[21] J. Leite and M. Ilic. Answer-set programming based dynamic user modeling for recommender systems. In *EPIA'07*, volume 4874 of *LNAI*. Springer, 2007.

[22] J. A. Leite. *Evolving Knowledge Bases*. IOS press, 2003.

[23] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Min. Knowl. Discov*, 5(1/2):115–153, 2001.