Automatic Optimization of Web Recommendations Using Feedback and Ontology Graphs

Nick Golovin and Erhard Rahm

University of Leipzig, Augustusplatz 10-11, 04109, Leipzig, Germany
{golovin,rahm}@informatik.uni-leipzig.de
dbs.uni-leipzig.de

Abstract. Web recommendation systems have become a popular means to improve the usability of web sites. This paper describes the architecture of a rule-based recommendation system and presents its evaluation on two real-life applications. The architecture combines recommendations from different algorithms in a recommendation database and applies feedback-based machine learning to optimize the selection of the presented recommendations. The recommendations database also stores ontology graphs, which are used to semantically enrich the recommendations. We describe the general architecture of the system and the test setting, illustrate the application of several optimization approaches and present comparative results.

1 Introduction

Many modern websites use web recommendations to increase usability, customer satisfaction and commercial profit. A number of algorithms were developed, which generate recommendations by applying different statistical or data mining approaches to some available to them information, for example on characteristics of the current page, product, web user, buying history etc.[5, 9] However, so far no single algorithm uses the benefits of all the available knowledge sources and no single algorithm shows clear superiority over all others. Therefore, the need for hybrid approaches which combine the benefits of multiple algorithms has been recognized [5].

In this paper, we present a new approach, capable of combining many recommender algorithms (or shortly *recommenders*). Our approach utilizes a central recommendation database for storing the recommendations, coming from different recommender algorithms and applies machine learning techniques to continuously optimize the stored recommendations. Optimization of the recommendations is based on how "useful" they are to users and to the website, i.e. how willingly the users click them or how much profit they bring. The incentive for our optimization approach was the observation, that the popularity and perceived relevance of individual recommendations are not always well predicted by the recommenders.

The information about the website and the users is represented in the recommendation database in form of ontology graphs. This allows us to semantically enrich the recommendations and bring in the knowledge from additional sources. It is also practicable for the adaptation of the system to the different types of websites.

The preliminary version of the architecture was sketched in [7]. In the current paper, we describe the architecture of the prototype implementations of the system and present evaluation results.

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 375–386, 2005.

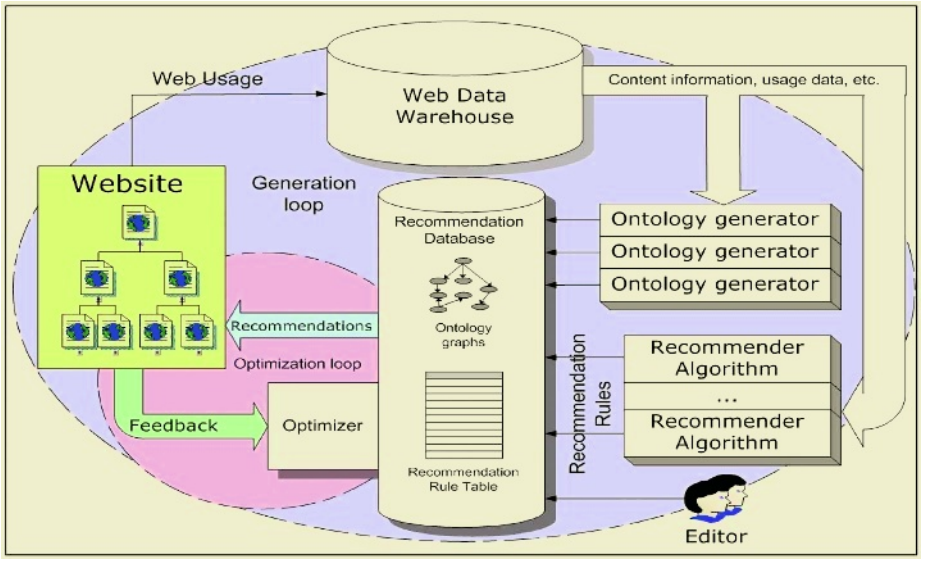

Fig. 1. Architecture of the recommendation system

We have implemented the prototype of our system on two websites: a website of the Database Group, University of Leipzig http://dbs.uni-leipzig.de and internet software shop http://www.softunity.com. To denote the origin of the examples and notions in the paper, we mark them either with EDU (educational) for the Database Group website or with EC (e-commerce) for www.softunity.com.

In the next section we explain the architecture of the system and its main components. Section 3 describes the selection of recommendations using ontology graphs and recommendation rules. In Section 4 we explain the recommenders and generation of the recommendations. The optimization techniques are presented in Section 5. Section 6 contains the evaluations of the real-life experimental data. In Section 7 we provide an overview of the related work. Section 8 summarizes the paper.

2 Architecture Overview

The architecture of our recommendation system is shown in Fig. 1. The *website* interacts with the web user, presents recommendations and gathers the feedback. The *web data warehouse* stores information about the content of the website (e.g., products and product catalog, HTML pages, etc.), users, and the usage logs generated by the web server or the application server. It serves as an information source for the recommender algorithms and ontology generators and allows OLAP evaluations of the usage data and the efficiency of recommendations. The *recommendation database* stores the semantic information in form of three directed acyclic ontology graphs (for the website content, web users and time) and the recommendations in form of recommendation rules, which are described in the next section. The set of *ontology generators* is responsible for generating the ontology graphs. The set of *recommender algorithms* generates recommendations using data from the web data warehouse.

Ontology graphs and recommendation rules can also be created and edited by a *human editor*. The *optimizer* refines the recommendation database based on the feedback obtained from the website using machine learning.

In our recommendation system we distinguish the generation loop and the optimization loop. The generation loop is executed at regular intervals of time. It involves generating/updating the ontology and the recommendation rules utilizing the information on the content and recent usage information from the web data warehouse. The optimization loop is continuously executed, and selects and presents the recommendations from the recommendation database. Furthermore, feedback is gathered, i.e. user reactions to presented recommendations. The optimizer uses this information to refine the recommendations in the database and to influence the selection of future recommendations.

3 Recommendation Selection Using Ontology Graphs

Fig. 2 shows the selection of recommendations using ontology graphs. To request recommendations to present, the website specifies the current *website context* and the desired number of recommendations. The website context is a set of parameters, which characterize the currently viewed website content, current web user and present point in time. An example of a website context is given below:

WebsiteContext{ ProductID="ECD00345"; UserCountry="DE";
UserOperatingSystem="Windows"; Date ="21.03.2005";...} (EC).

Obviously, the choice of suitable parameters in the website context depends on the specific website, especially with respect to the current content.

The recommendation system maps the provided website context into a *semantic context*, which consists of nodes of the three ontology graphs {ContentNodes, UserNodes, TimeNodes}. Using a selection policy, recommendations associated with the relevant nodes of the ontology graphs are selected and finally presented. This two-step selection process aims at supporting application-oriented recommendation strategies and high flexibility. Assigning recommendations to semantic concepts is expected to be more stable than directly using low-level website contexts whose values may change frequently (e.g. due to website restructuring).

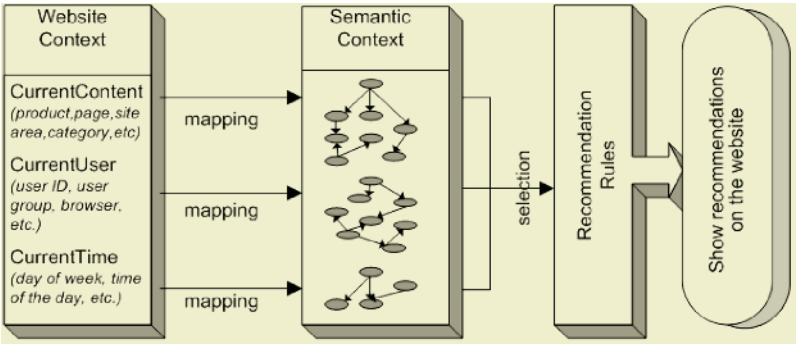

Fig. 2. Selecting recommendations using semantic context

Fig. 3 shows an example of an ontology graph for website content. Ontology graphs for web users and time are built in a similar way. We use directed edges to point from more specific concepts to more general concepts, from subcomponents to aggregated components, etc. Recommendations can be assigned to any node in such a graph. Highlighted with thick lines in Fig. 3 is an example of how the semantics stored in the ontology graph can be used to search for additional recommendations for Product4. We are able to retrieve the recommendations directly for Product4 as well the recommendations that are bound to some common property that Product4 possesses (in our case Hardcover) and the recommendations to some product catalog topics that Product4 is a part of (History, Books).

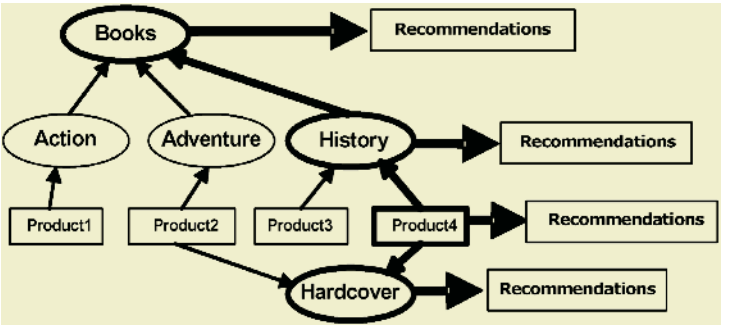

Fig. 3. A sample ontology graph for content dimension (EC)

The mapping between website and semantic contexts is specified by mapping clauses, which are statements written in a simple predicate language. The predicate language supports logical operators (AND, OR, NOT), comparison operators (<, >, =,<>,>=,<=) and operator LIKE, which does string matching with wildcard, similar to the SQL-operator with the same name. Some of the nodes in the ontology graphs immediately correspond to a certain set of parameters and can be mapped using mapping clauses. Others represent abstract notions and can be reached only by traversing the ontology graph and have no associated mapping clauses.

The ontology graphs are automatically generated by ontology generators and can be edited manually by the editors of the website. Each of the three ontology graphs is mapped separately. In our EC application, the ontology graphs are created by ontology generators using the product catalog, common properties of products and the business logic of the website. Application EDU uses the manually specified website content hierarchy and user groups determined by data mining (J48 decision tree algorithm). Most mapping clauses are automatically determined by the ontology generators together with the creation of the ontology and simply use an equality operator. Manually specified mapping clauses may be more complex. Examples of mapping clauses are:

ProductID=“ECD00345” -> ContentNode=1342 (EC)
UserCountry=“DE” -> UserNode=3 (EC)
UserDomain LIKE ‘%.edu’ OR UserDomain LIKE ‘%uni-%’ -> UserNode =2 (EDU)

The recommendations associated to nodes in the ontology graphs are represented by rules stored in the recommendation database. Recommendation rules have the form:

RuleContext{Content,User,Time} -> RecommendedContent, Weight

RuleContext refers to nodes in one or several of the three ontology graphs. These values can also be set to NULL, denoting that the rule does not depend on the corresponding dimension. RecommendedContent is the pointer to the content being recommended, e.g. recommended product or URL. The Weight is used as a criterion for the selection of the recommendation rules for presentations.

We have implemented several policies for selecting nodes of the ontology graphs and thus to select the associated recommendations. These policies include: "direct match", "direct + parents" and "combined". The "direct match" policy selects the recommendations using only the nodes matched using mapping clauses. The policy "direct + parents" uses these nodes as well as all their parents in the ontology graphs. In the "combined" policy the "direct match" policy is applied first. If this policy is unable to return the requested number of recommendations, the policy "direct + parents" is applied.

After the current semantic context is ascertained by selecting the nodes in the ontology graphs, we use the following general SQL query to select the recommendations from the rule table of the recommendation database:

```
SELECT TOP N RecomNode From Rules  WHERE
        (ContentNode in (CurrentContentNode1, CurrentContentNode2,…) OR
        ContentNode is NULL)  AND
        (UserNode in (CurrenUserNode1, CurrentUserNode2,…) OR
        UserNode is NULL)  AND
        (TimeNode in (CurrentTimeNode1, CurrentTimeNode2,…) OR
        TimeNode is NULL)
ORDER BY Weight DESC
```

We order the recommendations by weight and return the requested number of recommendations with the highest weight.

4 Creating Recommendation Rules

The recommendation rules are generated by the recommender algorithms and stored in the recommendation database. A recommender algorithm may also supply an initial weight for every generated recommendation rule from the interval [0 .. 1]. In the prototype implementation, we determine product recommendations with the following recommenders:

1. Content similarity. This recommender determines for each product (EC) or HTML page (EDU) (content node) the M most similar products using TF/IDF text similarity score.
2. Sequence patterns. Products (EC) or HTML Pages(EDU)most often succeeding other products/pages in the same user session are recommended to them.
3. Item-to-Item collaborative filtering.(EC) Products, which most often appear together in one user's basket, are recommended for each other.

4. Search Engine recommender (EDU). This recommender is applicable to the users coming from a search engine. It extracts search keywords from the HTTP Referrer field and uses the website's internal search engine to generate recommendations for each keyword. The recommender was first described and implemented in [17].

If the recommender algorithms generate a rule, which already exists in the recommendation rule table with different weight, the weight in the recommendation rule table takes preference over the weight supplied by the recommenders. We have explored two approaches to setting the initial weights of newly generated recommendation rules. In the first approach, we simply set all initial weights to zero (ZeroStart). The second approach uses normalized recommender-specific weights or relative priorities for the respective contexts. When several recommenders generate the same recommendation we use the maximum of their weights. The initial weights are expected to be relevant primarily for new recommendations since the weights for presented recommendations are continuously adapted.

5 Feedback-Based Optimization

The goal of our optimization is to adjust the weights of the recommendation rules in such a way, that the more useful recommendations are shown more often, than less useful. In our applications, we determine utility through the *acceptance rate* of the recommendations, which is:

$$AcceptanceRate = Nclicked / Npresented,$$

where *Nclicked* is number of times the recommendation was clicked and *Npres* is number of times the recommendation was presented.

However, our optimization algorithm is also able to work with utility determined otherwise, for example as sales turnover or profit brought by the recommendation.

Every time a web user requests a web page with recommendations, several recommendation rules from the recommendation database are selected and shown. We call such an event a *presentation*. The web user then may take some action in respect to the presented recommendations. For example, the user may click a recommendation, buy the recommended product, or ignore the recommendation. Each of these actions has a real value associated with it, called *feedback*.

The optimizer evaluates all presentations and adjusts the weights of the participating recommendation rules according to the obtained feedback. New recommendation rules can be added to the recommendation database at any time. Both online and offline optimization are possible.

There are two key aspects, which have to be addressed in our optimization algorithm. First, the utility of the individual recommendations may change over time, due to the "drift of interests" of the web users. The optimization algorithm must promptly react to the significant changes in user interests without overreacting to short-term fluctuations. Also, we are facing the "exploration vs. exploitation" dilemma. On one side, we would like to present the recommendations, which are the most "useful" according to our current knowledge, i.e. be "greedy". On the other side, we would like to learn, how good are the other recommendations, for which our current knowledge is insufficient [16, 8]. In the next subsections, we discuss these aspects in more detail.

5.1 "Drift of Interest"

We can handle the "drift of interest" in several ways:

- consider older feedback and newer feedback to be equally important (no aging). In this case, the weight of the recommendation rule is equal to the acceptance rate of the recommendation and the "drift of interest" is not taken into account.
- store the last n feedback values for each recommendation rule and generate the weights of the recommendation rules from them. This approach, however, requires additional memory in case of online optimization.
- use *aging by division* (also called *exponential smoothing*). Here, with every presentation the original weight of the recommendation rule is decreased by a fraction of its value:

$$Q(r) = (1\text{-}1/T)*Q(r) + Feedback(r) / T\,.$$

In this formula, Q(r) is the weight of the recommendation rule r, T is the aging parameter (T>1). Feedback is the numerical value which describes a user's response to the presentation of the given recommendation. Multiplying the weight by (1-1/T) implements the aging, since this way the latest presentations have the most impact on the resulting weight value while the contribution of past presentations decreases exponentially with each next presentation. The exact value of parameter T should be determined experimentally, depending on how dynamic the user interests for a website are.

5.2 Exploration Versus Exploitation

We have investigated two techniques of balancing between exploration and exploitation. In the *reward-only* technique, the tradeoff between exploration and exploitation is set statically through the parameter ε. With probability (1- ε) the technique selects the best recommendations according to their weights. With probability ε it selects random recommendations for presentation to give them a chance to be explored. This technique is also called ε-greedy in the literature [16]. The benefit of this technique is a very simple and understandable control over exploitation vs. exploration. The drawback is that it explores all recommendations with equal probability, not taking into account how promising they might be. The values of Feedback(r) for this technique are as follows:

1 if the recommendation r was clicked
0 if the recommendation r was not clicked

In the *reward-penalty* technique the balancing between exploration and exploitation is done dynamically using negative feedback. When some recommendation r in a presentation is clicked, r receives positive feedback, all other recommendations receive negative feedback. When no recommendation is clicked, after a predefined timeout all participating recommendations receive negative feedback. To prevent the weights from sliding into the extreme values, the feedback values should be chosen in such a way, that for any given context an approximate equilibrium is maintained throughout the process:

$$\Sigma(positive\ feedback) \approx -\Sigma(negative\ feedback)$$

For example:

1 if the recommendation r was clicked
$-p$ if the recommendation r was not clicked

where p is the probability, that a recommendation is clicked, averaged over all recommendation presentations on the web site. For both our applications. (EDU) and (EC), the value of $p \approx 0.01$.

With the reward-penalty technique we do not have to always sacrifice a fixed share of presentations for exploration. However, the drawback of this technique is the need for careful selection of the feedback values, because otherwise the learning process degenerates. It is also possible to combine the reward-penalty technique with the ε-random selection of the recommendations for exploration.

6 Prototype and Evaluation

In this section we describe the implementations of our prototype and evaluate the obtained results. In subsection 6.1 we present technical details of the implementation and effects of the recommendations on the buying behavior of the users. In subsection 6.2 we compare different optimization algorithms and recommendation rule selection policies.

6.1 Prototype Implementations, Click Rates and Economic Efficiency

A prototype of the system was implemented and applied at two websites. The first one is the website of the Database Group, University of Leipzig (http://dbs.uni-leipzig.de, approximately 2000 page views per day). It shows two (N=2) recommendations on all html-pages of the site. The second application is a small commercial online software store (http://www.softunity.com, approximately 5000 page views per day). Here, our approach is used to automatically select and present five (N=5) recommendations on product detail pages. Both websites have around 2500 content pages. The recommendation database contains about 60000 rules for (EDU) and 35000 rules for (EC).

The prototype uses a MySQL database server for the recommendation database and Microsoft SQL Server for the web data warehouse. All recommenders and the optimizer, as well as the websites themselves, are implemented using the PHP scripting language.

Fig. 4 shows the that the number of clicks per recommendation rule is distributed according to a Zipfian-like law (in the figure, only the recommendations with at least 100 presentations are considered.). The data shows that a relatively small percentage of the recommendation rules brings the majority of clicks. This supports our optimization heuristic, since it shows that we may achieve overall improvement of the acceptance rate by presenting the most successful recommendations more often.

In general, 2.07 % web users of www.softunity.com are becoming customers (this metric is usually regarded as CCR – Customer Conversion Rate). For web users who clicked a recommendation this value is 8.55 %, i.e. more than four times higher.

The analysis of the customer and purchase data has also shown, that 3.04 % of all purchased products were bought immediately after clicking the recommendation, and 3.43 % of all purchased products were recommended in the same session.

Fig. 5 shows the effects of our optimization algorithm in terms of buying behavior, in contrast to the non-optimized selection of the recommendations. The non-optimized algorithm uses the initial weights supplied by the recommenders to select the recommendations, and does no feedback-based optimization. The figure shows that the optimized approach results in a noticeable increase of the number of additions to shopping carts.

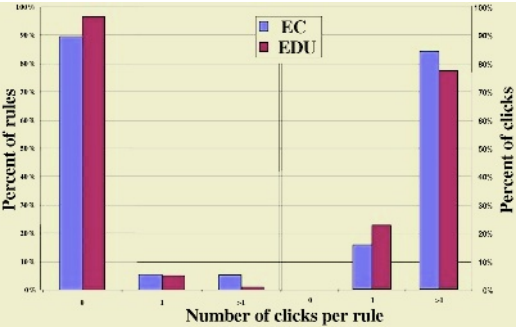

Fig. 4. A small number of recommendations brings the majority of clicks (EC, EDU)

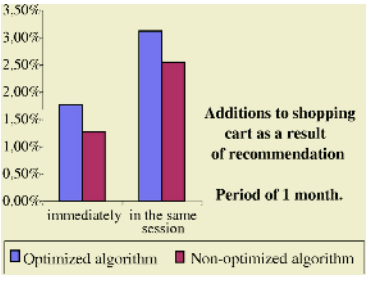

Fig. 5. Additions to basket as a result of recommendation (EC)

6.2 Optimization Algorithms and Recommendation Selection Policies

To evaluate the effectiveness of the different optimization algorithms, we have compared the performance of the reward-only and reward-penalty optimization algorithms with the selection of recommendations based on the initial weight supplied by recommender. For an evaluation period of several months the selection algorithm was chosen with equal probability from one of the following:

- reward-only, ε=0.2, no aging
- reward-only, ε=0.05, no aging
- reward-penalty, aging by division with T=200
- reward-penalty, aging by division with T=500
- without optimization

Fig. 6 shows that the optimized algorithms achieve higher acceptance rates than the algorithms without optimization. The algorithm, which uses penalty as well as reward, was able to achieve somewhat higher acceptance rates than the algorithm which uses only reward and with some probability selects random recommendation rules for exploration. Too quick aging (in our case T=200) can adversely affect the optimization. The relatively small improvement of the reward-penalty algorithm can be attributed to the fact, that in our applications the successful and unsuccessful recommendations can be distinctly separated even by the simpler algorithms. The algorithm which used zero as initial weights for the recommendation rules was tested on the EDU website. Its acceptance rate was only 4% lower than that of the algorithm which used recommender-specific initial weights.

Fig. 7 shows the session acceptance rates (number of sessions where at least one recommendation was accepted divided through total number of sessions) for different

recommendation selection policies introduced in Section 3. Five policies were tested. The random policy was used for comparison. In addition to the policies described in section 3, we have tested policies "only parents" and "only siblings" which were used to simulate the scenarios when no directly matching recommendations can be found. The policy "only parents" ignores the direct matching recommendations and takes only recommendations from the higher hierarchy levels. The policy "only siblings" searches for recommendations among the hierarchy siblings (nodes having a common parent with the current node), also ignoring the direct matches. According to the test results, the "direct match" policy performs better then the policy "direct+parents". However, the "direct+parent" policy is able to find recommendations even in cases, when no directly matching recommendations are available.

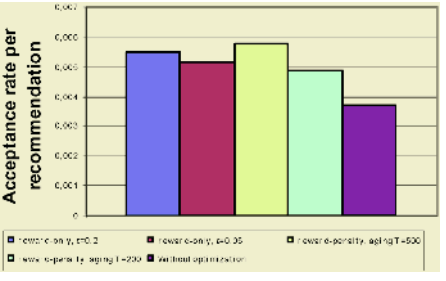

Fig. 6. Acceptance rate of different optimization algorithms(EC)

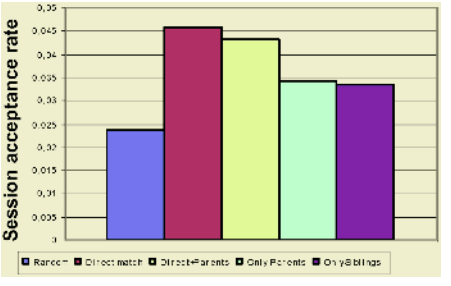

Fig. 7. Session Acceptance rate of the rule selection policies(EDU)

6.3 User Groups

Fig. 8 shows the comparison of the acceptance rates for different user groups. The user groups were built using a decision tree algorithm J48 over the usage data of several months from the EDU website. The DBS website is structured in several areas of interest, most important of which are Study and Research: For the decision tree algorithm, the area of interest (Research/Study) visited by a web user, has served as a classification attribute; other attributes were country, browser and operating system of the web user. However, after the tree was pruned, only the attribute country appeared to be of importance in addition to the area of interest. The resulting tree was transformed into ontology graph nodes with mapping clauses. Fig. 8 indicates that the acceptance rates differ substantially for the user groups and that for the considered website research-oriented users accept presented recommendations almost twice as much than study-oriented users.

Fig. 9 shows how good our user groups are in predicting the user interests. Here, differently colored bars show the acceptance for recommendations pointing to content of different interest areas. The user group Research appears to be quite effective, since its users have only clicked the recommendations leading to the research area of the website. Users of group Study preferred study-related recommendations but the corresponding acceptance rate is not much higher than for users not belonging to any of the two specific user groups.

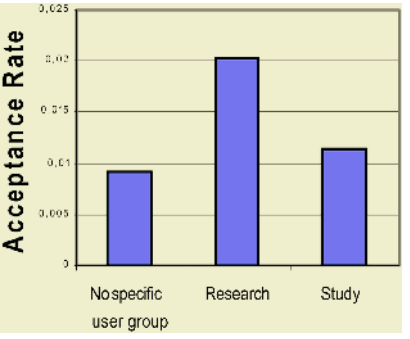

Fig. 8. Acceptance rates of different user groups (EDU)

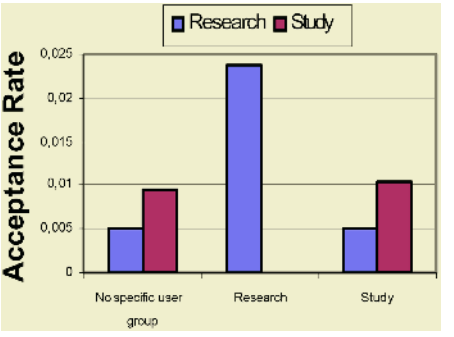

Fig. 9. Acceptance rates of user group based recommendation rules (EDU)

7 Related Work

A survey of hybrid recommender systems including a list of strengths and weaknesses of different recommender algorithms and a classification of the hybridization methods can be found in the [5].

The work in [17] also employs data warehouse to store usage information and implicit user feedback. However, in [17] the feedback is used to learn how to switch different recommender algorithms, which work independently, whereas in our approach the feedback influences the weights of individual recommendations ("switched" approach vs. "weighted" approach according to the classification in [5]). [17] describes several strategies, according to which the best recommender can be chosen.

Combining the collaborative filtering with content-based algorithms is addressed in [6], [13] and [3]. Their approaches strive to combine both algorithms in one algorithm in an algorithm-specific way. Our approach, in contrast, views these algorithms as independent, but dynamically combines their results in a way, optimized for the given website.

8 Summary

We described the architecture, implementation and use of a novel recommendation system. It uses multiple techniques to generate recommendations, stores them in a semantically enabled recommendation database and then refines them using online optimization. Our results for two sample websites showed that feedback-based optimization can significantly increase the acceptance rate of the recommendations. Even the simple optimization techniques could substantially improve acceptance of recommendations compared to the non-optimized algorithm. We have also shown that web recommendations and our optimization approach have considerable impact on the buying behavior of the customers of an e-commerce web site.

References

1. S. Acharyya, J. Ghosh: Context-Sensitive Modeling of Web-Surfing Behavior using Concept Trees. Proc. WebKDD, 2003

2. M. Balabanovic: An Adaptive Web Page Recommendation Service. CACM, 1997
3. J.Basilico, T.Hofmann.: Unifying collaborative and content-based filtering. Proc. 21th ICML Conference. Banff, Canada, 2004
4. S. Baron, M. Spiliopoulou: Monitoring the Evolution of Web Usage Patterns. Proc. ECML/PKDD, 2003
5. R. Burke: Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, 2002
6. Claypool, M., Gokhale, A., Miranda, T.: Combining Content-Based and Collaborative Filters in an Online Newspaper. In: Proc. ACM SIGIR Workshop on Recommender Systems, 1999
7. N. Golovin, E. Rahm: Reinforcement Learning Architecture for Web Recommendations. Proc. ITCC2004, IEEE, 2004
8. S. ten Hagen, M. van Someren and V. Hollink: Exploration/exploitation in adaptive recommender systems. Proc. European Symposium on Intelligent Technologies, Hybrid Systems and their Implementation in Smart Adaptive Systems, Oulu, Finland. 2003
9. A. Jameson, J. Konstan, J. Riedl: AI Techniques for Personalized Recommendation. Tutorial presented at AAAI, 2002
10. G. Linden, B. Smith, and J. York: Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Internet Computing. Jan. 2003
11. B. Mobasher, X. Jin, Y. Zhou. Semantically Enhanced Collaborative Filtering on the Web. Proc. European Web Mining Forum, LNAI, Springer 2004
12. M. Nakagawa, B. Mobasher: A Hybrid Web Personalization Model Based on Site Connectivity. Proc. 5th WEBKDD workshop, Washington, DC, USA, Aug. 2003
13. P. Paulson, A. Tzanavari: Combining Collaborative and Content-Based Filtering Using Conceptual Graphs. Lecture Notes in Computer Science 2873 Springer 2003
14. E. Reategui, J. Campbell, R. Torres, R. Using Item Descriptors in Recommender Systems, AAAI Workshop on Semantic Web Personalization, San Jose, USA, 2004
15. B. Sarwar, G. Karypis, J. Konstan, J. Riedl: Analysis of Recommendation Algorithms for E-Commerce. Proc. ACM E-Commerce, 2000
16. R.S. Sutton, A.G. Barto: Reinforcement Learning: An Introduction. MIT Press,1998.
17. A. Thor, E. Rahm: AWESOME - A Data Warehouse-based System for Adaptive Website Recommendations. Proc. 30th Intl. Conf. on Very Large Databases (VLDB), Toronto, Aug. 2004