Constraint-based Recommender Systems: Technologies and Research Issues

A. Felfernig
Intelligent Systems and Business Informatics
University of Klagenfurt, Austria
alexander.felfernig@uni-klu.ac.at

R. Burke
College of Computing and Digital Media
DePaul University Chicago, IL, USA
rburke@cs.depaul.edu

ABSTRACT
Recommender systems support users in identifying products and services in e-commerce and other information-rich environments. Recommendation problems have a long history as a successful AI application area, with substantial interest beginning in the mid-1990s, and increasing with the subsequent rise of e-commerce. Recommender systems research long focused on recommending only simple products such as movies or books; constraint-based recommendation now receives increasing attention due to the capability of recommending complex products and services. In this paper, we first introduce a taxonomy of recommendation knowledge sources and algorithmic approaches. We then go on to discuss the most prevalent techniques of constraint-based recommendation and outline open research issues.

Categories and Subject Descriptors
I.2.5. Expert system tools and techniques.

General Terms
Recommender Systems.

Keywords
Constraint-based Recommendation.

1. INTRODUCTION
Although e-commerce is more and more a dominant purchasing platform, buying complex products and services (e.g., financial services or computers) online is still a challenging task. Few organizations offer anything beyond simple query interfaces, under the assumption that customers know the technical details of the products and services they seek. Recommender technologies [2] [7] [14] [33] are an attempt to provide automated assistance for such decision tasks.

When originally developed, recommender systems were conceived primarily as social systems [21][31] through which users shared preferences over items (simple products such as books or movies). These collaborative recommendation techniques are widely used and actively researched, but the term *recommender systems* now refers to a much broader array of techniques that share a central focus on providing recommendations – personalized solution alternatives. We interpret a recommender system as *any system that guides a user in a personalized way to interesting or useful objects in a large space of possible options or that produces such objects as output.*

This paper highlights one particular technology for recommender systems: constraint-based recommendation. In this paradigm recommendation is viewed as a process of constraint satisfaction [40], some constraints come from users, other constraints come from the product domain. Products that satisfy the constraints are good recommendations. This paper situates constraint-based recommendation within the landscape of recommendation technologies by characterizing these technologies in terms of their knowledge requirements.

Constraint-based recommendation (see, e.g., [15]) like other knowledge-based recommendation techniques [6] becomes important when there are specific requirements that a solution must meet. Consider the following example:

> Joe would like to buy web hosting services for his business. He connects to a recommender system for such services and answers a few simple questions such as whether e-commerce services (e.g., shopping cart or advisory services) are required, what is the expected number of users per day, and what is the expected connection performance. Joe answers these questions and receives immediate feedback that due to the expected number of users hosting services should be based on a higher connection performance (an option that has as well been selected by many other users). Joe accepts this proposal and the system recommends the solutions *HostingA* and *HostingC*. Joe asks for more information on each of these, and learns that they both have similar pricing, but that although *HostingA* offers more storage, *HostingC* has a larger bandwidth. He ultimately decides that he would rather have the better bandwidth and chooses *HostingC*.

There are a number of ways in which this scenario goes beyond what could be expected of a purely collaborative system. For example, a collaborative system requires the accumulation of a history of choices or product preferences built up over time. It is these user profiles that are compared to find peer users with similar tastes, with most techniques requiring at least 20 ratings for reliability. Collaborative techniques would be unlikely to be successful for infrequently purchased items: a typical business owner might make web hosting decisions only once a year or even less frequently. Further, the kind of explanatory information (e.g., the repair proposal that is given) that Joe uses in making his

10th Int. Conf. on Electronic Commerce (ICEC) '08 Innsbruck, Austria

final decision would not likely be found in a purely collaborative recommender. However, even in this heavily knowledge-based scenario, collaborative information has its place: information about what other users have done in similar circumstances is extremely helpful.

To understand different recommendation approaches and the role that they can fulfill, it is best to have an overview of the various kinds of knowledge sources from which recommender systems can draw. This is the question that we turn to in Section 2. In Section 3, we look at the way that particular choices of knowledge sources have given rise to the well-recognized types of recommender systems. Section 4 examines in detail constraint-based recommendation technologies. In Section 5 we discuss research issues in constraint-based recommendation. With Section 6 we conclude the paper.

2. KNOWLEDGE SOURCES

Like all intelligent systems, recommender systems use different forms of knowledge. Sometimes this knowledge is *implicit*, such as the distribution of user opinions over some group of items or the knowledge encoded in an algorithm, in other cases it is *explicitly-encoded* inferential or ontological knowledge.

There are four sources where the knowledge needed to generate recommendations can be drawn: from the *user* herself, from *other peer users* of the system, from *data about the items* being recommended, and finally from the *domain of recommendation* itself, knowledge about how recommended items are used and what needs they satisfy.

Figure 1 shows a *taxonomy of recommendation knowledge* that builds on this commonsense distinction. We follow common usage in the field and describe as "Content" any type of knowledge that is not user-generated, although this might be considered overly broad.

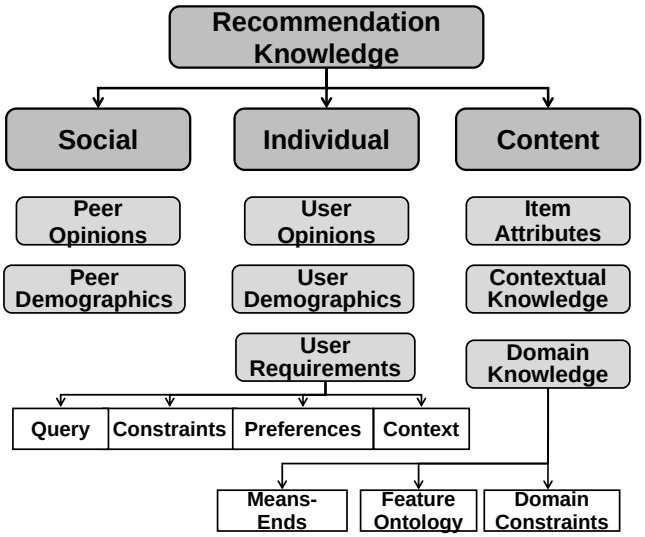

Figure 1: Knowledge Sources in Recommender Systems.

2.1 Collaborative (Social)

Collaborative (social) knowledge is knowledge about other users. Its most familiar manifestation are numeric *rating* (*opinion*) *profiles* [17][21][32] but other kinds of knowledge can be considered collaborative as well. In one well-known example, web links are treated as collaborative votes in the PageRank algorithm [4]. *Demographic data* has also been used to draw similarities between users or to complement rating behavior. Current research has begun to explore other forms of user opinion such as social tagging behavior [26] and written reviews [1]. In the case of our example of web hosting services, collaborative knowledge might include the opinions of other users on such services and possibly information about the companies, such as annual sales or the number of employees.

2.2 User (Individual)

Of course, in order to personalize recommendations, we need knowledge of the individual user. The relationship between user and collaborative knowledge can be purely reciprocal, in the sense that, e.g., Joe's *opinions* or *demographic data* are individual when the system is giving him a recommendation, but social when another user relies on them.

Historical knowledge of user preferences may be sufficient for some recommendation tasks, but in many circumstances, the user will come to the system with a particular intent in mind, and the recommender system will need to respond to it. These *user requirements* may come in different forms.

- **Query:** Depending on the user interface, it may be possible for the user to pose a general query or the interface may present specific feature/attribute requirements to be specified. The example above describes such an interface, which gathers values for specific features, such as the expected performance of the ISPs Internet connectivity.
- **Constraints**: In some recommendation domains, there may be a variety of types of constraints that a recommended solution must meet. For example, in the area of rental apartments, the owner of a German Shepherd would have the constraint that the landlord must accept large pets. No solution that violates this constraint would likely be acceptable.
- **Preferences**: A preference is something that the user would prefer to be true of the solution, but a solution that violates it might be acceptable. For example, a buyer may have a preference for a bottle of wine under $10, but might be satisfied with a $12 bottle if it were a particularly good deal, highly appropriate, or if no other alternative existed that satisfied other, harder, constraints.
- **Context**: The user's context consists of the external circumstances associated with the recommendation or the user's situation. For example, the user's location might be an important contextual factor in a restaurant recommendation, with closer establishments being preferred. The deployment of context in recommender systems is an area of active research often borrowing concepts from ubiquitous computing [34].

2.3 Content

The area of content knowledge is very broad. In some cases, a system will have very little knowledge that can be used to match users' needs against products. In an area of personal taste, like music or fiction, it would be very difficult to reason about the connection between the user's mental state and the way in which

items are or are not satisfactory. In other cases, such as the web hosting example above, there may be considerable scope for reasoning about the connection between users' needs and the available products. The taxonomy distinguishes four different types of content knowledge.

- **Item attributes**: Obviously, the starting point for any type of reasoning about items must be knowledge about the items themselves. This may be a simple set of attribute value pairs, such as might be found associated with a product in a database, or the item description may itself be structured as in the case of complex products such as a computer [12].
- **Contextual knowledge**: Considerable inferential complexity may be involved in teasing out the consequences of particular contexts [2]. In the example above, the details of Joe's line of business and his customer base may in fact have significant bearing on the importance attached to his web site. Contextual factors have been addressed significantly in only a handful of research and fielded systems. Some mobile recommender systems for example use location as a contextual cue.
- **Domain knowledge**: Finally, a recommender may need to have deeper knowledge about the products it is recommending and the uses that they may serve. This knowledge comes in a variety of forms:

  Means-ends knowledge: Means-ends knowledge is perhaps the most complex component of content knowledge. This is the inferential knowledge that allows the system to reason about how particular items (the "means") satisfy particular needs or requirements of the user (the "ends"). Such knowledge could be arbitrarily complex involving physical reasoning for example, but there are also much more straightforward formulations, such as the knowledge that a person interested in a "family car" will typically have a certain bundle of preferences and constraints, such as passenger and cargo space [5].

  Feature ontology: Item attributes tell a recommender what features are associated with what products, but often a recommender will need to know the relationship between these features: to know, for example, that Ubuntu is a type of Linux operating system.

  Domain constraints: Users have constraints in terms of the requirements that recommended items must meet, as described above. The items themselves may also have constraints that govern their appropriateness in different circumstances. For example, a particular insurance policy may only be available to individuals whose are non-smokers, or as in the Internet services case, the deployment of multi-media content places a limit on the minimum bandwidth a site will need.

3. RECOMMENDATION TECHNIQUES

For the purposes of this paper, a *recommendation technique* is a set of knowledge sources and an algorithmic approach to generating recommendations using those sources. There are of course no apriori limitations on how these different sources of knowledge can be combined, and in fact, the field of recommender systems has seen great diversity in approaches. However, certain approaches have turned out to be practical and effective and are generally accepted techniques. Constraint-based techniques will be discussed in detail in the next section. The following discussion is a brief overview of the other types.

3.1 Collaborative Recommendation

The approach most closely associated with recommender systems since the field's inception is of course collaborative filtering or collaborative recommendation.[1] The knowledge sources here are collaborative opinion profiles, demographic profiles, and user opinions. Of course one of the strengths of this technique is that little else is required to implement it in its pure form.

Many algorithmic approaches have been applied to these knowledge sources, but in general, the problem can be seen as a form of multi-way classification task. It differs from classic classification tasks in that there is no specific dependent class variable being predicted in all situations, but rather the system may be called upon to make predictions about any item for any user. Thus collaborative recommendation is not amenable to a strict application of the tools of machine learning.

The most well-known approach is that of a *nearest neighbor*, in which ratings are extrapolated by comparing the user opinion knowledge against collaborative opinions and extrapolations made [32]. The popular item-based variant treats the collaborative information as features associated with items rather than with users [35]. Model-based techniques have been employed to compress the collaborative opinion data, including clustering, singular value decomposition, and others [35].

What all of these methods share is their sole reliance on opinions or rating data as the knowledge source for recommendations. There are well-known characteristics that result from this choice. First, there is positive benefit that no other information about users or items is required. This makes the approach attractive for hard to characterize items, like movies and music.

This technique can be said to be the most mature of the recommendation technologies and the characteristics of these algorithms are well established. Collaborative recommendation works best when there is a large amount of collaborative knowledge available and when there is a substantial history of user opinion. The density of the collaborative user-item matrix is a substantial consideration.

Collaborative recommendation can theoretically be applied in any domain. It places very little restriction on the types of items that can be recommended and is highly suitable for items in which user tastes vary for reasons that might be difficult to represent explicitly, such as music and movies. However, there are considerations that may make a collaborative approach less attractive. One is opinion density. A user can listen to dozens of music tracks in a day, and might watch dozens of movies a year, but she would unlikely to live in dozens of apartments, own dozens of cars or have dozens of different pets. In some areas it is simply more difficult to accumulate a pattern of preference. It is possible to ask the user to venture a prospective opinion without having actually experienced the item in question, but such opinions will generally be quite speculative compared to those arrived at through experience.

[1] The "filtering" terminology is a legacy from an early application: filtering of interesting Usenet messages [21].

Also, we might expect that decisions regarding items less frequently purchased will be more sensitive to context. A 10-year-old rating for a car may be completely irrelevant if the user now has completely different driving patterns due to family and job requirements. Personal tastes in items such as music do change over time, but since they are experienced and rated more regularly, a collaborative system has the possibility of adapting.

The need for a substantial profile of the user gives rise to the "new user" problem. A new user cannot get much personalized help from a collaborative system until his profile is sufficiently large to be used reliably in prediction. A new item is similarly disadvantaged. It cannot be become a recommendation until some user rates it, and in general, the system will not have a good sense for a new item's appropriateness across the user base until a number of users have rated it. Together these issues are known as the "cold start" problem.

The frequency of item churn and the need for freshness are therefore important considerations. Domains in which items are rapidly appearing and disappearing will be difficult to handle via collaborative recommendation because an item might be nearing the end of its lifetime by the time it gets enough ratings to be recommended.

3.2 Content-based Recommendation

Content-based recommendation [28] is more like a pure classification task in the machine learning sense. The task is to learn a specific classification rule for each user on the basis of the user's rating information and the attributes of each item so that items can be classified as likely to be interesting or not. No social knowledge is used. The content-based problem has been tackled using a variety of machine learning techniques. However, in general, some of the more sophisticated techniques have been less successful, due to the sparsity problem. An individual user profile may not, in many cases, have enough data for a reliable profile. Simple techniques such as k-nearest neighbors and naive Bayes have often proved effective here.

Because a content-based recommender has access to item features (e.g., keywords or categories), it does not suffer from the *new item problem*: new items look just like old items. The *new user problem* remains since users must build up a sufficiently rich profile through the addition of multiple ratings. Depending on the feature set and learning algorithm, however, a small number of ratings may be sufficient.

Indeed, the quality of the data set becomes of primary importance in a content-based system. The learned profiles of each user will only be as good as the system's level of detail in representing the distinctions that matter in the domain. The creators of the music recommender Pandora (www.pandora.com) first developed a highly-detailed representation of musical form and expression before attempting to build a recommender system. Developers must make sure that all items in the catalog have a uniform, detailed and complete representation: a combination which can be difficult to achieve in many e-commerce contexts.

3.3 Knowledge-based Recommendation

Systems that rely on knowledge sources other than those discussed in Sections 3.1 and 3.2 above are by default known as knowledge-based recommender systems [6]. Such systems are characterized therefore by the two knowledge aspects not found in the other designs: namely user requirements and domain knowledge. Obviously, there are collaborative and content-based systems that allow users to pose queries or that have some forms of heuristics with respect to their content. What distinguishes a knowledge-based approach is that its emphasis: emphasis on the user's situation and how recommended items can meet that particular need.

Knowledge-based recommender systems are more difficult to briefly characterize than the other techniques discussed above. If we consider the taxonomy in Figure 1 and the knowledge sources used in collaborative and content-based recommendation, it is clear that the knowledge-based category itself is something of an accident of history. Systems that used additional knowledge sources came to be defined as "knowledge-based" because they relied more heavily on knowledge sources that were not being employed by the more widely-used techniques.

There are two well-known approaches to knowledge-based recommendation (*case-based recommendation* [6] [25] [34] [38] and *constraint-based recommendation* [15] [39]. Systems based on these approaches can exploit all of the knowledge sources depicted in Figure 1. In terms of used knowledge sources, both approaches are quite similar. Systems of both types must, for example, collect the requirements of the current user in order to derive new solutions, propose repairs in situations where no solution could be found and support explanations for the recommended items.

Case-based recommendation treats recommendation as primarily a similarity-assessment problem. How can the system find a product that is most similar to what the user has in mind, with the understanding that what counts as similar will often involve domain-specific knowledge and considerations. Constraint-based recommendation takes into account explicitly defined constraints (e.g., filter constraints or incompatibility constraints). If no item really fits the wishes of a customer (the calculated similarity value exceeds a certain threshold for all relevant products or the set of constraints is inconsistent with the given set of customer requirements) both knowledge-based approaches exploit mechanisms supporting the determination of minimal set of changes to the given set of customer requirements such that a solution can be found - see, for example, [15] [24].

The interaction with a knowledge-based recommender application is typically modeled in the form of a dialog (conversational recommender) where users can specify their requirements in the form of answers to questions [6] [15] [25] [34] [39]. This dialog can be modeled explicitly, for example, in the form of a finite state automaton (see, e.g., [15]) or in a way which allows the user to select interesting attributes (questions) on her own [25]. Furthermore, user interaction with a recommender application can be enriched with natural language interaction [39] which allows for more flexible interaction processes. For example, users can specify component properties on a textual level without being forced to answer a potentially larger number of questions. Another interesting aspect of additional textual interfaces is the flexibility to support queries which are not directly related to product search but to issues such as questions regarding the functionality or technical questions.

Many case-based recommender applications support the concept of critiquing (see, e.g., [6] [8] [38]), in which the user responds to

a recommended item by identifying how it differs from their ideal. For example, a user presented with a restaurant featuring a traditional style of food may apply the critique "More Creative" and obtain a more contemporary take on the same cuisine [5]. Such an interface has the advantage of allowing the user to formulate requirements on the fly, in response to examples. Critiquing interfaces promise a faster identification of interesting recommendations in terms of less decision effort, better decision accuracy, and increase of a user's confidence in a decision [8]. Recent research has moved from *unit critiques*, in which the user identifies specific item properties to critique (e.g., "I would prefer a camera with a lower price"), to more complex *compound critiques*, that move along several feature dimensions at once. Such critiques can be manually engineered (as in [5]) or can be automatically generated from the existing product assortment by mining association rules representing representative critique patterns in the available assortment [38].

3.4 Hybrid Recommendation

A hybrid recommender is one that uses recommendation components or logic of different types. See [7] for an overview. For example, a recommender might use both social knowledge and item features thereby combining collaborative and content-based approaches. In some sense, the notion of a hybrid is an artifact of the historical development of recommender system in which certain types of knowledge sources were exploited first, leading to well-established techniques that were later combined. If we view recommendation as a problem the solution to which may draw on multiple knowledge sources, then the only question to be answered is which sources are the most appropriate to a given task and how they can be most effectively used.

4. CONSTRAINT-BASED RECOMMENDATION

Let us reconsider the recommendation of web hosting services. The user of the recommender has to provide information about his personal preferences regarding, for example, the number of visitors, the need of e-commerce and advisory services, the maximum price and the required bandwidth for the connection.

On the basis of a given set of user preferences, the recommender proposes alternative solutions including explanations as to why those solutions have been proposed. Alternatively, no solution could have been found by the recommender. In this situation, a corresponding set of repair alternatives is derived which help the user to get out of the dead-end. Applications supporting such recommendation functionalities in many cases build upon constraint-based technologies [40], where specific product properties as well as the relationship between customer requirements and products are modeled in the form of constraints.

Constraint-based approaches typically help to successfully establish recommenders in domains where items are infrequently purchased. Furthermore, items are more complex and many customers do not know all the technical features in detail. Example product domains range from technical equipment, financial services, or e-government services to highly complex products such as production systems or telecommunication switches. Constraint-based recommenders support customers by explaining items, automatically proposing repair actions in situations where no solution can be found, and by proposing attribute settings based on the preferences of a customer community thus including collaborative recommendation as well.

The application of constraint-based recommenders in the financial services domain (recommendation of loans) is shown in [15]. Applications supporting the interactive selling of other types of financial services are reported in [12]. The core business model of all those applications is to improve the quality of service for customers, more specifically the quality of advisory processes. Those applications have an impact in the sense of, for example, time savings in advisory sessions due to automated explanation and repair processes, an increased number of sold products, a significant reduction of faulty offerings [15].

The authors of [18] present an approach to multimedia-enhanced recommendation where constraint-based technologies are additionally equipped with a component supporting item and component visualization. Thus standard constraint-based approaches are extended with visualization functionalities letting users directly interact with the virtual product. Visualization functionalities provide substantial contributions to user-friendly interfaces boosting the acceptance of recommenders.

4.1 Recommendation Knowledge

Constraint-based recommendation requires the explicit definition of questions, product properties and constraints. These elements constitute a *recommender knowledge base* which can be represented as a constraint network consisting of two sets of variables (U, P) and the corresponding constraints (*COMP*, *PROD*, *FILT*).

In this context, customer properties $u_i \in U$ describe all possible requirements which can be specified by customers. For example, *max-price = 15* or *commerce-services = yes* are potential requirements regarding a web hosting service. Furthermore, product properties $p_i \in P$ describe the offered product assortment, for example: *name*, *price*, or supported *bandwidth*. A simple example recommender knowledge base is the following:

U = { u_1:*commerce-services*(*yes*, *no*),
u_2:*connection-performance*(*low*, *medium*, *high*),
u_3:*advisory-services*(*yes*, *no*),
u_4:*number-users*(*integer*),
u_5:*maxprice*(*integer*)}

P = { p_1:*id*(*integer*),
p_2:*name*(*text*),
p_3:*commerce-services*(*yes*, *no*),
p_4:*price*(*integer*),
p_5:*available-storage*(*5GB*, *10GB*, *20GB*, *50GB*),
p_6:*supported-bandwidth*(*3Mbit*, *5Mbit*, *10Mbit*)}

$comp_i \in COMP$ are (*in*)*compatibility constraints* restricting the set of possible requirements. For example, *commerce-services* require a high *connection-performance*, i.e., a low or medium connection performance is incompatible with commerce services.

id	***constraint***
$comp_1$	*advisory-services* require *commerce-services*.
$comp_2$	*commerce-services* require a high *connection-performance*.
$comp_3$	a number of expected users (*number-users*) per day greater than 250 requires a high *connection-performance*.

Table 1: Example Compatibility Constraints.

Such constraints help to assure that customer requirements remain consistent and that customers learn about specific properties of the product domain. Furthermore, they help to assure the consistency of offers posed to customers and thus in many cases help to decrease costs related to correction processes.

$prod_i \in PROD$ are *product constraints* responsible for restricting the possible instantiations of variables in *P*. Such constraints can be seen as compatibility constraints specifically used to enumerate the offered set of products. Example constraints representing such products are shown in Table 2.

id	*name*	*commerce-services*	*advisory-services*	*price*	*storage*	*bandwidth* (*Mbit/s*)
$prod_1$	HostingA	yes	yes	20	20GB	5
$prod_2$	HostingB	no	no	10	5GB	3
$prod_3$	HostingC	yes	yes	20	10GB	10
$prod_4$	HostingD	no	yes	15	10GB	5
$prod_5$	HostingE	no	no	5	5GB	3
$prod_6$	HostingF	yes	yes	20	20GB	10

Table 2: Example Product Assortment.

$filt_i \in FILT$ (*filter constraints*) define the relationship between customer requirements and products. These constraints represent rules related to marketing and sales strategies. Example filter constraints are shown in Table 3 – they describe in detail under which conditions a certain product should be recommended.

id	*constraint*
$filt_1$	the *price* of the product has to be lower (equal) than (to) the maximum price (*maxprice*) imposed by the customer.
$filt_2$	more than 250 visitors per day (*number-users*) requires a *bandwidth* of more than 3Mbit/s.
$filt_3$	a high *connection-performance* is defined by a *bandwidth* of more than 3Mbit/s.

Table 3: Example Filter Constraints.

4.2 Dialog Structure

One frequently used approach to the representation of dialog structures is to explicitly define interaction sequences in which customers can answer questions [10] [15]. This type of design exploits the formalisms of finite state automata to explicitly design the possible states of a recommendation session [10]. A simple example for such a description is shown in Figure 2.

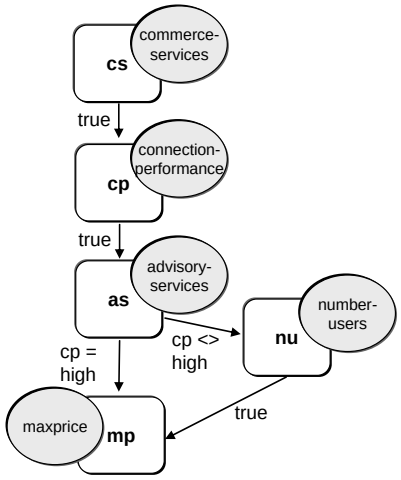

Figure 2: Example User Interface Description (simplified).

Typically, states correspond to input units of the recommender application, where specific questions can be posed to customers. Depending on the answers (preferences) of the customer, one of the set of possible following states is selected. In our example, the question regarding the number of users (*number-users* - *nu*) is only posed in situations where customers specify the requirement of a low or medium *connection-performance* (*cp*).

Explicitly defined interaction sequences are well-suited for domains where users do not have a clear opinion about which questions should be answered next. For example, new sales representatives in the financial services domain adopt such interaction mechanisms since those clearly help them to effectively conduct sales dialogs [15].

In other application domains, customers want to select relevant and interesting attributes on their own and appreciate personalized recommendations regarding interesting attributes which trigger significant time savings regarding the completion of recommendation sessions [25].

4.3 Calculating Recommendations

On the basis of the definition of a recommendation knowledge base, we are able to calculate a concrete recommendation for a customer. The task of deriving recommendations for a customer is denoted as *recommendation task*. Given a set of customer requirements, we can calculate a recommendation (result). A constraint-based recommender $\Re_{constr}$ is a system that computes solutions $\{prod_i \in PROD\}$ for a given recommendation task.

Definition (*Recommendation Task*). We define a recommendation task as a Constraint Satisfaction Problem (CSP) [40] (U, P, $CR \cup COMP \cup FILT \cup PROD$) where U is a finite set of variables representing potential requirements of customers and P is a set of variables defining the basic properties of the product assortment. Furthermore, CR is a set of customer requirements, $COMP$ represents a set of (incompatibility) constraints, $FILT$ is a set of filter constraints, and $PROD$ specifies the set of offered products.

A *solution* to a given recommendation task (U, P, $CR \cup COMP \cup FILT \cup PROD$) is a complete assignment to the variables of (U, P) such that this assignment is consistent with the constraints in ($CR \cup COMP \cup FILT \cup PROD$).

Table 4 depicts the requirements $\{req_1, req_2, req_3, req_4, req_5\} \in CR$ of our example customer (Joe) regarding his web hosting service solution. Two alternative solutions exist for those requirements: *HostingA* ($prod_1$) and *HostingC* ($prod_3$). On the basis of these requirements, Joe now for example could decide that he would rather have the candidate option $prod_3$, i.e., the one with the better bandwidth (compared to the alternative $prod_1$).

commerce-services (req_1)	*advisory-services* (req_2)	*connection-performance* (req_3)	*number-users* (req_4)	*max-price* (req_5)
yes	yes	high	500	20

Table 4: Example Customer Requirements.

4.4 Utilities of Recommendations

Let us assume our recommender for web hosting services has knowledge about the major strengths and weaknesses of the offered products. Those can be specified in terms of their contribution to the interest dimensions *reliability*, *economy*, and *performance*. Such dimensions are the basic elements of Multi-

Attribute Utility Theory (MAUT) [41] which is used for calculating the utility of product alternatives for a specific customer. In this context, the term *utility* denotes the degree of fit between an item the given set of customer requirements.

Table 5 depicts scoring rules that define the relationship between our candidate recommendations ($prod_1$ and $prod_3$) and the defined interest dimensions. For example, regarding *performance*, alternative $prod_3$ is better compared to alternative $prod_1$.

product	*reliability*	*economy*	*performance*
$prod_1$	7	7	7
$prod_3$	6	7	9

Table 5: Recommendations and Interest Dimensions.

We can apply the following MAUT utility function in order to determine the most interesting recommendation for Joe.

$$utility(x) = \Sigma_{(i=1..n)}\ e_i s_i(x) \qquad (1)$$

In this formula, n denotes the number of interest dimensions, $utility(x)$ represents the utility of an item x, e_i is the interest of the customer in dimension i, and $s_i(x)$ represents the contribution of item x to dimension i. The values for e_i can be derived directly from customer requirements defined within the scope of a recommendation session [15]. We now assume that Joe is highly interested in *reliability* (importance weight of 0.7) and less interested in *economy* (importance weight of 0.15) and *performance* (importance weight of 0.15). Evaluating the two product alternatives on the basis of Formula 1 results in the ranking *HostingA* (*utility*=6.8) > *HostingC* (*utility*=6.5).

4.5 Explaining Inconsistent Requirements

Table 6 depicts a slightly changed set of requirements {req_1', req_2', req_3', req_4', req_5'} (compared to the definitions in Table4).

commerce-services (req_1')	*advisory-services* (req_2')	*connection-performance* (req_3')	*number-users* (req_4')	*max-price* (req_5')
no	yes	medium	500	20

Table 6: Inconsistent Customer Requirements.

Obviously, those requirements are inconsistent with our example set of compatibility constraints (see Table 1) and we have to support the customer in getting out of this situation. We are interested in repair actions [15] which indicate *interesting and minimal changes* to the requirements (*CR*) s.t. the calculation of a recommendation becomes possible.

The calculation of such repairs is based on the concepts of Model-Based Diagnosis [30] which is used to resolve conflicts [19]. In this context, a conflict is defined as $c=\{req_\alpha, req_\beta, \ldots, req_\psi\} \subseteq CR$, s.t. $c \cup COMP \cup FILT \cup PROD \cup U \cup P$ is inconsistent. A conflict c is said to be minimal if not $\exists$ a conflict c': $c' \subset c$. In our example, alternative repairs resolving the conflicts {c_1: (req_1, req_2), c_2: (req_3, req_4)} for the requirements shown in Table 6 are the following (see Table 7).

commerce-services (**)	*advisory-services* (**)	*connection-performance* (**)	*number-users* (*)	*max-price* (*)
no⇒ yes	yes	medium⇒ high	500	20
no	yes⇒ no	medium⇒ high	500	20

Table 7: Repair Alternatives
(* no change possible, **change is possible)

The question now is which of the two alternative repairs should be proposed to the customer. For this purpose we can apply the following formula (Formula 2).

$$utility(r) = 1/\Sigma_{(i=1..m)}\ e_i c_i(r) \qquad (2)$$

In this context, m denotes the number of different requirements, $utility(r)$ represents the utility of a repair alternative r, e_i is the interest of the customer in requirement i (importance of requirement i), and $c_i(r)$ (can be 0 or 1) indicates whether requirement i is changed in the current repair alternative r. The values for e_i can be derived by directly asking customers within the scope of a recommendation session or by learning those preferences from customer surveys or previous interactions sequences [15]. We now assume that our customer has the following preferences regarding the specified requirements (req_i x *importance factor*): {(req_1', 0.1), (req_2', 0.4), (req_3', 0.3), (req_4', 0.1), (req_5', 0.1)}. Interpreting our two example repair alternatives using Formula 2 results in the selection of *repair alternative 1* (1/0.4 > 1/0.7) which should then be proposed to the customer.

4.6 Predicting Attribute Settings

In situations where users do not have detailed technical knowledge about the product domain and are not able to specify all relevant requirements, prediction mechanisms support the identification of potentially useful and interesting attribute settings. Such predictions could be either calculated on the basis of explicitly defined rules for deriving default values [15] or calculated on the basis of already existing interaction sequences with the recommender application [9].

Using the interaction sequences (history of customer requirements) of Table 8 we can predict interesting values for those attributes not already specified by the customer. For this purpose we can, for example, apply a *weighted majority voter* (*wmv*) [9] (see Formula 3). For each value s of the domain of attribute $u_\alpha \in \{u_{k+1}, u_{k+2}, \ldots, u_m\}$ we determine its potential degree of interest for the customer. $\{u_1, u_2, \ldots, u_k\}$ is the set of attributes already instantiated by the customer and $\{u_{k+1}, u_{k+2}, \ldots, u_m\}$ is the set of those attributes not instantiated by the customer. Furthermore, $\{inst(u_1), inst(u_2), \ldots, inst(u_k)\}$ is the set of instantiations of the attributes in $\{u_1, u_2, \ldots, u_k\}$ and $[u_{1i}, u_{2i}, \ldots, u_{mi}]$ is a sequence (vector) of valuations of the attributes $\{u_1, u_2, \ldots, u_m\}$ in already existing interaction sequences $seq_i \in SEQ$ (i=1..n). The potentially most interesting value $s \in dom(u_\alpha)$ can be calculated on the basis of the following formula:

$$wmv(s \in dom(u_\alpha)) = \Sigma_{(i=1..n)}\ (\Sigma_{(j=1..k)}\ [inst(u_j)=u_{ji}])*[u_{\alpha i}=s]. \qquad (3)$$

In this context, i iterates over interaction sequences and j over already instantiated attributes. We assume that the customer has already specified the following requirements: {req_1: *commerce-services=yes*, req_3: *connection-performance=high*}. The set *SEQ* of already existing interaction sequences is shown in Table 8.

We are now interested in a prediction for the attribute *advisory-services* (u_α=*advisory-services*). Applying Formula 3 to the given setting, results in the recommendation of value *yes* for the attribute u_α= *advisory-services*.

Further approaches to the prediction of attribute values take into account the recommendation of attribute sets not already specified by the customer and are based on the application of Naïve Bayes voters. Further details on such approaches can be found in [9].

seq_i	*commerce-services* (u_1)	*connection-performance* (u_2)	*advisory-services* (u_3)	*number-users* (u_4)	*max-price* (u_5)
1	yes (u_{11})	medium (u_{31})	no (u_{21})	100 (u_{41})	15 (u_{51})
2	yes (u_{12})	low (u_{32})	yes (u_{22})	50 (u_{42})	20 (u_{52})
3	no (u_{13})	medium (u_{33})	no (u_{23})	150 (u_{43})	25 (u_{53})
4	yes (u_{14})	high (u_{34})	yes (u_{24})	200 (u_{44})	25 (u_{54})
5	yes (u_{15})	high (u_{35})	yes (u_{25})	500 (u_{45})	20 (u_{55})

Table 8: History of Customer Requirements.

s	$[u_{\alpha i}=s]$	$(\Sigma_{(j=1..k)}\ [inst(u_j)=u_{ji}])$	$\Sigma_{(i=1..n)}$
yes	0,1,0,1,1	1,1,0,2,2	5
no	1,0,1,0,0	1,1,0,2,2	1

Table 9: Calculation of attribute values.

4.7 Selecting Attributes

Another question strongly related to the recommendation of attribute values is the recommendation of interesting questions, i.e., questions which a user would like to reply to. Where traditional entropy-based methods strongly focus on the reduction of the size of the result set, the aspect of user-centered question ordering is in many cases not taken into account [25]. In order to support such functionalities, we need a user interface which supports the selection of interesting attributes (see, e.g., [25]) and a corresponding history of user interaction sequences.

A simple approach to the prediction of interesting attributes is presented in [25]. This approach is based on the measurement of the frequency of attribute usage (popularity). The popularity of an attribute can be calculated using Formula 4.

$$popularity(u_\alpha, pos) = \#(selections\ of\ u_\alpha\ in\ pos)/\#(sessions) \quad (4)$$

Taking into account the example attribute selection sequences shown in Table 10, $popularity(u_1{:}commerce\text{-}services, 1)$ has the highest evaluation (0.6). Another approach to select interesting attributes within the scope of recommendation sessions is to apply the weighted majority voter (Formula 3) [9] for determining the next interesting attribute. Thus, the determination of interesting attribute values is substituted by the selection of interesting attributes, i.e., in this case the domain under consideration is represented by a set of different attribute identifiers $\{u_1, u_2, \ldots, u_n\}$. In our example, u_2 would be selected as the next interesting attribute (see Table 11) (assuming that $\{u_1, u_3\}$ have already been selected).

i	*attribute1*	*attribute2*	*attribute3*	*attribute4*	*attribute5*
1	u_1	u_3	u_2	u_5	u_4
2	u_1	u_3	u_2	u_5	u_4
3	u_1	u_2	u_3	u_4	u_5
4	u_2	u_1	u_3	u_4	u_5
5	u_2	u_1	u_4	u_3	u_5

Table 10: History of Selected Attributes.

s	$[u_{\alpha i}=s]$	$(\Sigma_{(j=1..k)}\ [inst(u_j)=u_{ji}])$	$\Sigma_{(i=1..n)}$
u2	0,1,0,0,0	2,2,1,0,0	4
u4	0,0,0,0,1	2,2,1,0,0	0
u5	0,0,0,0,0	2,2,1,0,0	0

Table 11: Calculation of Interesting Attributes.

4.8 Knowledge Acquisition

In contrast to collaborative and content-based filtering ones, constraint-based recommenders rely on an explicit knowledge representation. Changes in the marketing and sales knowledge have to be immediately updated in the underlying recommender knowledge base in order to be visible and operable for customers as well as for sales representatives. Such updates hold the risk of introducing erroneous definitions which causes faulty recommendations and explanations. In order to support effective development and maintenance processes, we have to provide effective knowledge acquisition functionalities which support automated testing and debugging processes.

An approach to the automated generation of test cases for recommender knowledge bases is documented in [13] where test cases (examples for correct and faulty recommendations) are systematically generated from a given user interface description (represented as a finite state automaton). In this context, heuristics are presented which help to reduce the number of test cases to be validated by a domain expert. Concepts from Model-Based Diagnosis (MBD) [30] are applied in order to identify minimal sets of changes in the recommender knowledge base such that all recommendations fulfill the predefined examples. Related empirical studies presented in [13] report significant time savings in the identification and repair of erroneous constraints in recommender knowledge bases.

A similar approach has been developed for the identification of faulty transition conditions in recommender user interface descriptions [10] and the identification of faulty scoring rules in utility constraint sets used for the calculation of item utilities (which item in the recommendation set has the highest utility for the customer) [11]. Those debugging and repair concepts have as well been evaluated within the scope of empirical studies which clearly show up the potentials for significant time savings in development and maintenance processes for recommenders.

5. RESEARCH ISSUES

Constraint-based recommendation technologies have successfully shown their applicability in real-world environments. However, there exist a number of research issues which have to be tackled in order to further increase the impact of those technologies.

Consumer Decision Making and Recommender Applications. Psychological studies [27] have shown that customers do not exactly know their preferences when confronted with a set of product alternatives. Typically, preferences are constructed while learning about the choices offered. In order to successfully deploy recommender applications in commercial environments we have to understand the limiting factors customers are subject to when interacting with a recommender application. On the one hand *need for cognition* [23] is a property indicating the extent to which a user wants to invest time in cognitive efforts related to the finding of products and services. On the other hand, people have *limited cognitive resources* and use heuristics to find optimal products and services with as little effort as it is possible. This well known *effort-accuracy tradeoff* [27] has been observed as well in online decision situations such as web content search [22]. Both aspects, a persons need for cognition and the effort-accuracy tradeoff influence the way in which persons try to solve a given decision making task. Consequently, the design of recommenders can have a huge impact on how decisions tasks are solved by customers. In order to take into account such aspects, psychological theories from the areas of decision theory and cognitive psychology have to be investigated in detail regarding

their impact on online buying situations. First steps in this direction are documented in [16].

Complexity Metrics for Recommender Knowledge Bases. The automated testing and debugging of recommendation knowledge bases is documented in a number of publications – see, for example, [10][11][13]. Those techniques help to accelerate error identification and repair tasks, however, in many cases knowledge engineers are confronted with different repair alternatives which help to restore consistency. The question which has to be answered in this context is which repair alternative should be recommended. Software engineering research has developed a couple of complexity metrics which help to effectively identify faulty parts in given software components. In this context, knowledge-based systems research should focus on the development of complexity metrics for knowledge bases (in our case recommender knowledge bases) which help to focus error identification/repair on the most complex parts of a knowledge base.

Community-based Recommendation. Existing recommender applications are based on knowledge bases created by a small number of domain experts and knowledge engineers. Future environments for the development of recommender knowledge bases will focus on a distributed approach where whole communities of interest will contribute to the development of a knowledge base. Such an approach has high potential for improving knowledge base quality and the effectiveness of related development and maintenance processes. Such a distributed paradigm requires changes in underlying development and maintenance processes (e.g., distributed diagnosis and repair processes or social network analysis) as well intuitive knowledge acquisition interfaces supporting end-user development of knowledge bases.

Recommendation of Configurable Products and Services. Mass Customization [29] is a paradigm which aims at supporting the provision of highly variant products and services under the pricing conditions of Mass Production. In parallel, the increasing size and complexity of product and service assortments outstrips the capability of customers to survey it. Especially for highly variant products and services, the integration of existing configuration technologies with recommendation approaches is crucial in order to effectively support customers in their preference construction processes. The application of recommendation technologies in this context is manifold, for example, the recommendation of interesting attribute and component settings, the recommendation of repair actions, the recommendation of complete configurations or sub-configurations, or the recommendation of explanations.

6. CONCLUSIONS

Recommender systems employ a wide variety of AI techniques. These disparate strands of research can be brought together through an understanding of the knowledge sources on which they draw. In this paper, we have presented a taxonomy of these knowledge sources in order to clarify the relationships between different recommendation techniques.

We have also summarized the most prevalent techniques of constraint-based recommendation, a recommendation technique drawing heavily on domain knowledge. This technique is particularly applicable in complex product- and service domains. Unlike collaborative and content-based methods, constraint-based recommendation does not suffer from cold start problems, and can provide recommendations even for products that are not purchased or experienced frequently enough to generate a meaningful history of preferences.

7. REFERENCES

[1] Aciar S., Zhang, D., Simoff, S., and Debenham, J. 2007. Informed Recommender: Basing Recommendations on Consumer Product Reviews, *IEEE Intelligent Systems Special Issue: Recommender Systems*, 22(3):39-47.

[2] Adomavicius, G. and Tuzhilin, A. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6): 734-749.

[3] Belkin, N., Kelly, D., Kim, G., Kim, J., Lee, H. Muresan, G., Tang, M., Yuan, X., Cool, C. 2003. Query length in interactive information retrieval, 26th *ACM SIGIR Conference on Research and Development in Information Retrieval*, Toronto, Canada, pp. 205-212.

[4] Brin, S. and Page, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *7th International WWW Conference*, Brisbane, Australia, pp.107-117.

[5] Burke, R., Hammond, K., and Young, B. 1997. The FindMe Approach to Assisted Browsing. *IEEE Expert*, 12(4), pages 32-40.

[6] Burke, R. 2000. Knowledge-based recommender systems. *Encyclopedia of Library & InformationSystems*, 69(32).

[7] Burke, R. 2002. Hybrid recommender systems: survey and experiments. *User Modeling and User-Adapted Interaction* 12(4):331–370.

[8] Chen, L., and Pu, P. 2006. Evaluating Critiquing-based Recommender Agents. 21st *National Conference on Artificial Intelligence (AAAI'06)*, pp. 157-162, Boston, USA.

[9] Coester, R., Gustavsson, A., Olsson, R., and Rudstroem, A. 2002. Enhancing web-based configuration with recommendations and cluster-based help', in *AH'02 Workshop on Recommendation and Personalization in E-Commerce*, Malaga, Spain.

[10] Felfernig, A., Friedrich, G., Isak, K., Shchekotykhin, K., Teppan, E, and Jannach, D. 2008. Automated Debugging of Recommender User Interface Descriptions, to appear: *Journal of Applied Intelligence*.

[11] Felfernig, A., Friedrich, G., Teppan, E., and Isak, K 2008. Intelligent Debugging and Repair of Utility Constraint Sets in Knowledge-based Recommender Applications, to appear *13th ACM International Conference on Intelligent User Interfaces*, Canary Islands, Spain.

[12] Felfernig, A. 2007. Standardized Configuration Knowledge Representations as Technological Foundation for Mass Customization, *IEEE Transactions on Engineering Management*, 54(1):41-56.

[13] Felfernig, A. 2007. Reducing Development and Maintenance Efforts for Web-based Recommender Applications,

International Journal of Web Engineering and Technology, 3(3):329-351.

[14] Felfernig, A., Friedrich, G., Schmidt-Thieme, L. 2007. *Introduction to the IEEE Intelligent Systems Special Issue: Recommender Systems*, 22(3):18-21.

[15] Felfernig, A., Isak, K., Szabo, K., and Zachar, P. 2007. The VITA Financial Services Sales Support Environment, *AAAI/IAAI 2007*, pp. 1692-1699, Vancouver, Canada.

[16] Felfernig, A., Friedrich, G., Gula, B., Hitz, M., Kruggel, T., Melcher, R., Riepan, D., Strauss, S., Teppan, E., and Vitouch, O. 2007. Persuasive Recommendation: Exploring Serial Position Effects in Knowledge-based Recommender Systems, Persuasive 2007, LNCS, 4744.

[17] Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53.

[18] Jiang, B., Wang, W., and Benbasat, I. 2005. Multimedia-Based Interactive Advising Technology for Online Consumer Decision Support, Communications of the ACM. 48 (9), 93–98.

[19] Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. 19th *National Conference on AI (AAAI'04)*, pp. 167-172.

[20] Kelleher, J. and Bridge, D. 2004. An Accurate and Scalable Collaborative Recommender, *Artificial Intelligence Review*, 21(3-4):193-213.

[21] Konstan, J., Miller, B., Herlocker, J., Gordon, L., and Riedl, J. 1997. GroupLens: Applying Collaborative Filtering to Usenet News, *Communications of the ACM*, 40(3): 77-87.

[22] Kuoa, F., Chub, T., Hsuc, M., Hsieha, H. 2004. An investigation of effort-accuracy trade-off and the impact of self-efficacy on Web searching behaviors, Decision Support Systems, 37: 331-342, Elsevier.

[23] Martin, B., Sherrard, M., Wentzel, D. 2005.The Role of Sensation Seeking and Need for Cognition on Web-Site Evaluations: A Resource-Matching Perspective, Psychology and Marketing, 22(2): 109-126, Wiley.

[24] McSherry, D. 2004. Maximally Successful Relaxations of Unsuccessful Queries. *15th Conference on Artificial Intelligence and Cognitive Science*, Galway, Ireland, pp. 127–136.

[25] Mirzadeh, N., Ricci, F., Bansal, M. 2005. Feature Selection Methods for Conversational Recommender Systems, *IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE'05)*, Hongkong, 772-777.

[26] Niwa, S., Doi, T., and Honiden, S. 2006. Web Page Recommender System based on Folksonomy Mining for ITNG '06 Submissions, *3rd International Conf. on Information Technology*, Las Vegas, Nevada, pp. 388-393.

[27] Payne, J., Bettman, J., and Johnson, E. 1993. The Adaptive Decision Maker. Cambridge University Press.

[28] Pazzani, M., and Billsus, D. 1997. Learning and Revising User Profiles: The Identification of Interesting Web Sites, Machine Learning. (27), 313–331, (1997).

[29] Pine, B., and Davis, S. 1999. Mass Customization: The New Frontier in Business Competition. Harvard Business School Press, 1999.

[30] Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence*, 23(1):57-95, 1987.

[31] Resnick, P. and Varian, H. R. 1997. Recommender Systems, *Communications of the ACM*, 40(8):56-58.

[32] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, NC, pp. 175–186.

[33] Ricci, F., and Werthner, H. 2006. Special Issue of International Journal of Electronic Commerce, on Recommender Systems, 11(2).

[34] Ricci, F. and Q. Nguyen. 2007. Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System, *IEEE Intelligent Systems Special Issue: Recommender Systems*, 22(3):22-29.

[35] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. *10th International WWW Conference*, Hongkong, pp. 285-295.

[36] Schafer, J., Konstan, J., and Riedl, J. 2000. Electronic Commerce recommender applications. *Journal of Data Mining and Knowledge Discovery*, 5(1/2):115–152.

[37] Simon, H. 1955. A Behavioral Model of Choice. Quarterly Journal of Economics, 69(1): 99-118.

[38] Smyth, B., McGinty, L., Reilly, J., McCarthy, K. 2004. Compound Critiques for Conversational Recommender Systems, IEEE/WIC/ACM *International Conference on Web Intelligence (WI'04)*, Beijing, China, pp. 145 – 151.

[39] Thompson, C., Göker, M., and Langley, P. 2004. A Personalized System for Conversational Recommendations. *Journal of Artificial Intelligence Research* 21:393–428.

[40] Tsang, E. 1993. Foundations of Constraint Satisfaction, Academic Press, London and San Diego.

[41] Winterfeldt, D., and Edwards, W. Decision Analysis and Behavioral Research, Cambridge University Press, Cambridge, England, 1986.