Automated Repair of Scoring Rules in Constraint-Based Recommender Systems

Alexander Felfernig [a,*], Stefan Schippel [b], Gerhard Leitner [c], Florian Reinfrank [a], Klaus Isak [a], Monika Mandl [a], Paul Blazek [d], and Gerald Ninaus [a]

[a] *Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b, A-8010 Graz, Austria*
E-mail: {firstname.lastname}@ist.tugraz.at
[b] *Institute for Applied Informatics, Alpen-Adria University Klagenfurt, Universitaetsstrasse 65–67, A-9020 Klagenfurt, Austria*
E-mail: {Stefan.Schippel}@zh-tech.at
[c] *Institute for Informatics Systems, Alpen-Adria University Klagenfurt, Universitaetsstrasse 65–67, A-9020 Klagenfurt, Austria*
E-mail: {gerhard.leitner}@aau.at
[d] *cyLEDGE Media GmbH, Schottenfeldgasse 60, 1070 Vienna, Austria*
E-mail: {p.blazek}@cyledge.com

Constraint-based recommender systems support customers in preference construction processes related to complex products and services. In this context, utility constraints (scoring rules) play an important role. They determine the order in which items (products and services) are presented to customers. In many cases utility constraints are faulty, i.e., calculate rankings which are not expected and accepted by marketing and sales experts. The adaptation of these constraints is extremely time-consuming and often an error-prone process. We present an approach to the automated adaptation of utility constraint sets which is based on solutions for nonlinear optimization problems. This approach increases the applicability of constraint-based recommendation technologies by allowing the automated reproduction of example item rankings specified by marketing & sales experts.

Keywords: Constraint-based Recommender Systems, Knowledge Acquisition, Development Methods

[*] The work presented in this paper has been partially funded by the Austrian Research Promotion Agency (Projects: ICONE–827587 and Casa Vecchia–825889).

1. Introduction

Recommender systems support users in the identification of relevant products and services in situations where the amount and/or complexity of an offer outstrips the capability to survey it and to reach a decision [3,4,24].

The most well known recommendation approach is *collaborative filtering* [14][15] which is an implementation of word-of-mouth promotion where buying decisions are influenced by the opinion of friends. For example, if two customers bought similar books in the past and rated them in a similar way, the recommender system would propose books to one customer (books not known to him yet) that the other customer has rated positively.

Content-based filtering [21] is an information filtering approach which exploits item features a user has liked in the past with the goal of deriving new recommendations. For instance, if a customer has bought books about the Linux operating system, similar books (related to Linux) will be recommended in future advisory sessions. Content-based filtering does not exploit serendipity effects[1] which are in many cases required and welcome in recommendation contexts [2]. Typical applications of content-based filtering are the recommendation of Web content and news.

Collaborative filtering as well as content-based filtering are based on user profiles and do not exploit deep knowledge about the product domain. In contrast, *knowledge-based recommender systems* [2][12][29] exploit deep knowledge about the product domain in order to determine solutions fitting the wishes and needs of customers. Compared to customers purchasing simple products (e.g., books, compact disks, or movies), customers purchasing products such as digital cameras, computers, or financial services are much more in the need of intel-

[1] Identifying something which is useful without explicitly looking for it.

ligent interaction mechanisms supporting the retrieval and explanation of solutions. Such interaction mechanisms are supported by two basic types of knowledge-based recommender systems [2,12]. First, *critiquing-based recommender systems* [2][2] exploit the similarity between explicitly defined user requirements and items of the product assortment. Second, *constraint-based recommender systems* [12] exploit a set of explicitly defined constraints to calculate the set of products which are assumed to be of relevance for the customer. These constraints represent product, marketing, and sales knowledge in an explicit fashion and thus make it possible (1) to *derive* recommendations complying with existing marketing and sales strategies, (2) to *explain* these recommendations, and (3) to *propose* repair actions for inconsistent customer requirements [10].

In this paper we focus on a specific knowledge acquisition aspect in constraint-based recommender systems development [12] which is the development and maintenance of utility constraint sets (scoring rules). Products included in a recommendation have to be ranked according to their relevance for the customer [10][12]. In the line of *serial position effects* which induce customers to preferably take a look at and select items at the beginning of a list, the high-ranking of the most relevant items is extremely important [13][19]. For the determination of such rankings we apply the concepts of Multi-Attribute Utility Theory (MAUT) [17][25][30] where each product is evaluated according to a predefined set of *interest dimensions* which are abstract evaluation criteria for products. *Profit* and *availability* are examples for such interest dimensions in the domain of financial services. For example, if a customer is interested in *high return rates* and *longterm investments*, the dimension *profit* is very important. Consequently, customer requirements influence the importance of corresponding interest dimensions.

The consistency between utility constraints (scoring rules) and a company's marketing and sales strategy plays an important role for the successful application of recommender technologies. These constraints have to reflect marketing and sales strategies (in our example case those of financial service providers). Experiences from commercial projects [10] show a remarkable *need for a knowledge acquisition support* that alleviates the development and maintenance of utility constraint sets. *The manual adaptation of utility constraints is a time-consuming and error-prone task since such constraints are strongly interdependent.* Therefore, we developed techniques which support knowledge engineers in the identification and repair of faulty elements in utility constraint sets. We present adaptation concepts which automatically identify the sources of inconsistencies in utility constraint sets and propose corresponding repair actions. The presented approach has been implemented for a commercially available recommender environment [10] and is in the line of previous work [9] related to effective knowledge acquisition interfaces for recommender applications.

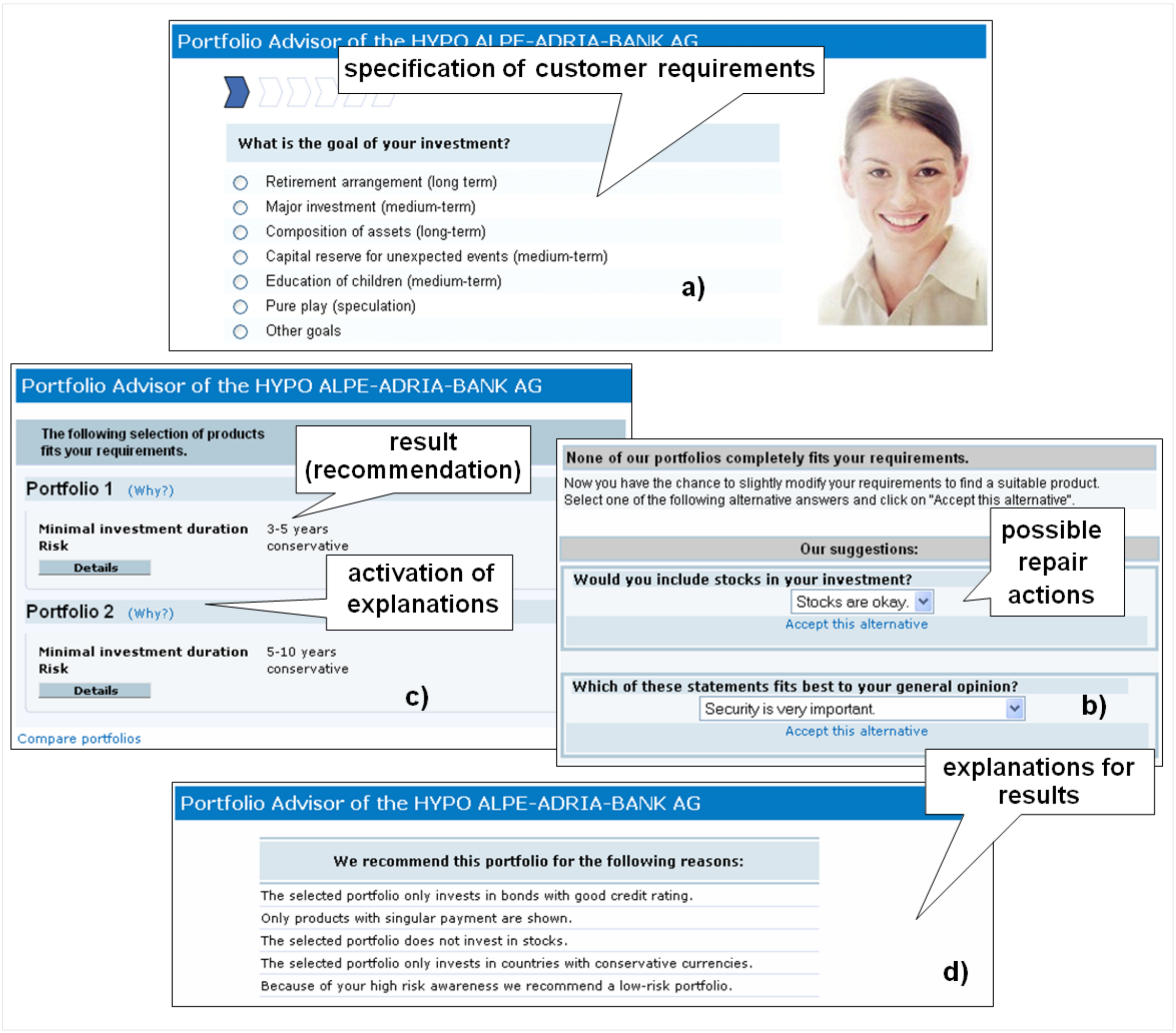
The remainder of this paper is organized as follows. In the following section we introduce the concepts of constraint-based recommendation. In Section 3 we present a simple set of utility constraints which serve as a working example throughout the paper. In Section 4 we show a CSP encoding for utility constraints. This encoding is then exploited for the automated determination of repairs for inconsistent utility constraint sets (Section 5). In Section 6 we discuss empirical findings related to the development of utility constraint sets for commercial applications. A discussion of related and future work is provided in Section 7. We conclude the paper with Section 8.

2. Constraint-based Recommendation

Figure 1 depicts example screenshots of an investment recommender application we have developed for a large financial service provider in Austria. The major task of this application is to support customers and sales representatives in the identification of appropriate investments (portfolios). In this context, users are involved in preference construction processes [6][22] where questions have to be answered and repair alternatives are proposed in situations where no solution can be found (see Figure 1, a. and b.). In the following, different investment alternatives are presented where each alternative has an assigned set of explanations as to why it has been recommended (see Figure 1, c. and d.).

Informally, a recommendation problem description consists of the following elements:

[2]Similarity-based approaches to the identification of solutions have been originally developed for case-based reasoning (CBR) systems [27] and later on adopted for the development of (critiquing-based) recommender systems [2].



Portfolio Advisor of the HYPO ALPE-ADRIA-BANK AG

What is the goal of your investment?

- Retirement arrangement (long term)
- Major investment (medium-term)
- Composition of assets (long-term)
- Capital reserve for unexpected events (medium-term)
- Education of children (medium-term)
- Pure play (speculation)
- Other goals

Portfolio Advisor of the HYPO ALPE-ADRIA-BANK AG

The following selection of products fits your requirements.

Portfolio 1 (Why?)

Minimal investment duration	3-5 years
Risk	conservative

Details

Portfolio 2 (Why?)

Minimal investment duration	5-10 years
Risk	conservative

Details

Compare portfolios

None of our portfolios completely fits your requirements.

Now you have the chance to slightly modify your requirements to find a suitable product. Select one of the following alternative answers and click on "Accept this alternative".

Our suggestions:

Would you include stocks in your investment? Stocks are okay.

Accept this alternative

Which of these statements fits best to your general opinion? Security is very important.

Accept this alternative

Portfolio Advisor of the HYPO ALPE-ADRIA-BANK AG

We recommend this portfolio for the following reasons:

The selected portfolio only invests in bonds with good credit rating.
Only products with singular payment are shown.
The selected portfolio does not invest in stocks.
The selected portfolio only invests in countries with conservative currencies.
Because of your high risk awareness we recommend a low-risk portfolio.

Fig. 1. Example financial service recommender application: users (customers and sales representatives) can specify requirements (a). In the case that no solution can be found by the recommender application, possible repair alternatives are presented (b). Solutions (results) are presented in the form of a recommendation list (c) where each entry in the recommendation list is associated with a set of explanations as to why this item has been recommended (d).

customer property $cp_i \in$ CP	dom(cp_i)
cp_1: investment period	{long term, medium term, short term}
cp_2: goal	{rainy days, stable growth, speculation}

Table 1

Example customer properties CP = $\{cp_1, cp_2\}$.

- *Customer properties* (CP) define the set of possible requirements which can be articulated by a customer, for example, the intended *investment period* and the investment *goal* (see Table 1).
- *Product properties* (PP) define the set of possible product properties, for example, the *name*, the inclusion of *shares*, and the amount of *value fluctuation* (see Table 2).
- *Customer requirements* (CR) define the requirements of a concrete customer, for example, a *medium term investment period* and

product property $pp_i \in PP$	dom(pp_i)
pp_1: name	{balanced funds, bonds, bonds2, equity}
pp_2: shares	{0%, 50%, 100%}
pp_3: value fluctuation	{low, medium, high, very high}

Table 2

Example product properties PP = {pp_1, pp_2, pp_3}.

customer	investment period	goal
Robert	medium term (r_1)	rainy days (r_2)

Table 3

Example customer requirements CR = $\{r_1, r_2\}$.

filter constraint $fc_i \in FC$
fc_1: goal = rainy days $\rightarrow$ shares $\neq$ 100%
fc_2: investment period = shortterm $\rightarrow$ value fluctuation $\neq$ high

Table 4

Example filter constraints FC = {fc_1, fc_2}.

incompatibility constraint $ic_i \in IC$
ic_1: $\neg$(goal = stable growth $\wedge$ investment period = short term)
ic_2: $\neg$(goal = rainy days $\wedge$ investment period = long term)

Table 5

Example incompatibility constraints IC = {ic_1, ic_2}.

constraint	name	shares	value fluctuation
$prod_1$	balanced funds	50%	medium
$prod_2$	bonds	0%	medium
$prod_3$	bonds2	0%	high
$prod_4$	equity	100%	very high

Table 6

Example set of financial services represented by PC = $\{prod : prod_1 \vee prod_2 \vee prod_3 \vee prod_4\}$ where $prod_1$: *name = balanced funds* $\wedge$ *shares = 50%* $\wedge$ *value fluctuation = medium*, $prod_2$: *name = bonds* $\wedge$ *shares = 0%* $\wedge$ *value fluctuation = medium*, $prod_3$: *name = bonds2* $\wedge$ *shares = 0%* $\wedge$ *value fluctuation = high*, and $prod_4$: *name = equity* $\wedge$ *shares = 100%* $\wedge$ *value fluctuation = very high.*

money for rainy days (see Table 3).

– *Filter constraints* (FC) define which products should be recommended – see Table 4.
– *Incompatibility constraints* (IC) define restrictions on combinations of requirements – see Table 5.
– Finally, a *product catalog* (PC) defines the available product assortment (see Table 6).

More formally, a recommendation problem can be defined as a Constraint Satisfaction Problem CSP [32] as follows:

Definition 1 (Recommendation Problem). A recommendation problem can be defined as a triple (V, D, C) where V = CP $\cup$ PP and D represents the set of corresponding domain definitions. Furthermore, C={CR, FC, IC, PC} includes all constraints of the recommendation problem.

On the basis of this definition of a recommendation problem we can now introduce the definition of a corresponding recommendation result (a recommendation):

Definition 2 (Recommendation). A recommendation for a given recommendation problem (V, D, C) is the set of consistent and complete instantiations of all variables in V.

The recommendation result in our example is shown in Table 7. In this example, product $prod_4$ has been filtered out by the filter constraint fc_2.

variable in V	$prod_1$	$prod_2$	$prod_3$
cp_1: investment period	medium term	medium term	medium term
cp_2: goal	rainy days	rainy days	rainy days
pp_1: name	balanced funds	bonds	bonds2
pp_2: shares	50%	0%	0%
pp_3: value fluctuation	medium	medium	high

Table 7

Recommendation result for the recommendation problem (V, D, C).

3. Example Utility Constraint Set

We now present a simplified set of *utility constraints* (scoring rules) (UC) which is used as working example throughout the paper. Constraints in UC determine a ranking in which items (products and services) element of a recommendation result are presented to the customer (see Tables 8, 9, 11, and 12). We determine item orderings by using the concepts of *Multi-Attribute Utility Theory* (MAUT) [17][25][30]. The basic elements of MAUT are *interest dimensions* such as *profit* or *availability* describing interest focuses of a customer. For instance, *profit* denotes the performance of financial services in terms of, for example, high return rates. Furthermore, *availability* is related to aspects of accessibility of the invested sum within the targeted investment period.

The degree to which a customer is interested in such dimensions can be derived from the articulated requirements. Tables 8–9 include typical scoring rules (utility constraints $uc_i \in UC$) of reliable financial service providers. A customer interested in *long term investments* (*investment period* = *long term*) typically has a lower interest in *availability* than a customer who is interested in *short term investments* (*investment period* = *short term*). Similarly, customers interested in *speculations* (*goal* = *speculation*) have a lower interest in availability than those interested in *putting money aside for rainy days* (*goal* = *rainy days*). For the purposes of our example, we use the customer requirements specified in Table 3: customer *Robert* is interested in *medium term* investments with the goal of *putting money aside for rainy days.*

By interpreting the information of Tables 3, 8, and 9 we can figure out to which extent *Robert* has a focus on the interest dimensions *profit* and *availability. Robert* requires a *medium term investment* solution which contributes an importance of 6 to the interest dimension *profit* and an importance of 5 to the interest dimension *availability* (see Table 8). Furthermore, *Robert* is interested in putting money aside for rainy days which contributes an importance of 2 to the dimension *profit* and 6 to *availability.* Table 10 summarizes the preferences of *Robert.*

On the basis of such customer preferences we are able to evaluate which of a given set of alternative products (services) best suits a customer's wishes and needs. For the purpose of our simplified example we use the recommendation result shown in Table 7. We now define the dependencies between product attribute values and the interest dimensions *profit* and *availability.* For instance, financial services including shares support a higher *profit* (see Table 11). Furthermore financial services without shares have a higher *availability*, and those with a higher value fluctuation have a higher (potential) *profit.*

By interpreting the constraints of Tables 7, 11, and 12, we can derive product assortment specific scoring rules $up_i \in UC$ (see Tables 13 and 14).

On the basis of these scoring rules we can determine the extent to which our financial services contribute to the interest dimensions *profit* and *availability* (see Table 15). By exploiting the identified product utilities, we can determine the customer-specific utility of each product x which is contained in the recommendation result (see Table 15). The utility of a product or service can be determined on the basis of Formula (1)

$$utility(x) = \sum_{i=1}^{n} in_i \cdot con_i(x) \qquad (1)$$

where $utility(x)$ specifies the overall utility of a product/service x for a specific customer. The overall utility of x is defined as sum over the customer's interest in dimension i (in_i) times the contribution of product x (in our case financial service) to dimension i (con_i). In our example, *balanced funds* have a higher utility for *Robert* than *bonds* and *bonds2* (see Table 16).

investment period	profit	availability
short term	4 (uc_1)	9 (uc_2)
medium term	6 (uc_3)	5 (uc_4)
long term	8 (uc_5)	1 (uc_6)

Table 8

Scoring rules $\{uc_1, ..., uc_6\} \subseteq UC$ for customer property *investment period.*

goal	profit	availability
rainy days	2 (uc_7)	6 (uc_8)
stable growth	6 (uc_9)	4 (uc_{10})
speculation	9 (uc_{11})	2 (uc_{12})

Table 9

Scoring rules $\{uc_7, ..., uc_{12}\} \subseteq UC$ for customer property *goal.*

customer	profit	availability
Robert	6+2=8	5+6=11

Table 10

Interests of the customer *Robert* derived from the constraints in Tables 3, 8, and 9.

shares	profit	availability
0%	2	7
50%	5	5

Table 11

Scoring rules for product property *shares.*

value fluctuation	profit	availability
medium	5	6
high	7	4

Table 12

Scoring rules for product property *value fluctuation.*

name	profit	availability
balanced funds	5 (up_1)	5 (up_2)
bonds	2 (up_3)	7 (up_4)
bonds2	2 (up_5)	7 (up_6)

Table 13

Product-specific scoring rules $\{up_1, ..., up_6\} \subseteq UC$ for product attribute *shares.*

name	profit	availability
balanced funds	5 (up_7)	6 (up_8)
bonds	5 (up_9)	6 (up_{10})
bonds2	7 (up_{11})	4 (up_{12})

Table 14

Product-specific scoring rules $\{up_7, ..., up_{12}\} \subseteq UC$ for product attribute *value fluctuation.*

name	profit	availability
balanced funds	5+5=10	5+6=11
bonds	2+5=7	7+6=13
bonds2	2+7=9	7+4=11

Table 15

Utilities of products regarding interest dimensions (result of interpreting Tables 13 and 14).

customer	product	profit	availability	utility
Robert	balanced funds	8*10	11*11	201
	bonds	8*7	11*13	199
	bonds2	8*9	11*11	193

Table 16

Utilities of products for customer *Robert*.

example	investment period	goal	ranking
e_1	medium term	for rainy days	utility(bonds) > utility(balanced funds)
e_2	medium term	for rainy days	utility(bonds2) > utility (balanced funds)
e_3	medium term	for rainy days	utility(bonds) > utility(bonds2)

Table 17

Examples E = $\{e_1, e_2, e_3\}$ of intended service orderings.

In order to test whether a given set of utility constraints calculates intended rankings, a corresponding set of *examples* (test cases) can be provided by marketing and sales experts (see, e.g., Table 17). In the case that the rankings calculated by the utility constraint set are in contradiction with the rankings of the given examples, we have to identify repairs such that the consistency with the examples is restored. In our scenario, the examples (E = $\{e_1, e_2, e_3\}$) are partially contradicting with the rankings (utilities) shown in Table 16 (e.g., the utility of *bonds* is lower than the utility of *balanced funds* if a customer is interested in medium term investments for rainy days, the contrary is specified in e_1: utility(*bonds*) > utility(*balancedfunds*)). Consequently, we have to identify an adaptation of our utility constraints. An approach to derive such adaptations automatically will be discussed in the following sections.

4. Utility Constraint Set (CSP Representation)

We now transform our utility constraint set (tabular representation of Section 3) into a corresponding constraint-based representation which is used as input for solving a non-linear optimization problem [31] (see Section 5). Following the definitions of Tables 8–9, we introduce the following set of utility constraints related to the required investment period and the personal goals of the customer. For instance, constraint uc_1 denotes the fact that for customers requiring financial services with short term investment periods, the dimension *profit* is of medium importance on value scale of [1..10], whereas *availability* aspects play a significantly more important role (uc_2). $\{uc_1, ..., uc_6\}$ represent the utility definitions of Table 8, $\{uc_7, ..., uc_{12}\}$ represent the definitions of Table 9.

uc_1: $profit_{(investmentperiod\_short)} = 4$
uc_2: $availability_{(investmentperiod\_short)} = 9$
uc_3: $profit_{(investmentperiod\_medium)} = 6$
uc_4: $availability_{(investmentperiod\_medium)} = 5$
uc_5: $profit_{(investmentperiod\_long)} = 8$
uc_6: $availability_{(investmentperiod\_long)} = 1$
uc_7: $profit_{(goal\_rainydays)} = 2$
uc_8: $availability_{(goal\_rainydays)} = 6$
uc_9: $profit_{(goal\_growth)} = 6$
uc_{10}: $availability_{(goal\_growth)} = 4$
uc_{11}: $profit_{(goal\_speculation)} = 9$
uc_{12}: $availability_{(goal\_speculation)} = 2$

We denote each constraint defining such utility values as utility constraint $uc_i \in$ UC. Since we are interested in a utility constraint set which is consistent with all the examples $e_i \in$ E, we have to check the consistency of the given set of utility constraints with $\bigcup e_i$. This type of consistency check requires a representation where each example is described by a separate set of finite domain variables. For instance, the contribution to *profit* provided by the customer attribute *investment period* in example e_1 is stored in the variable $\text{profit}_{(investmentperiod\_e1)}$. The following representation of examples can be directly interpreted by a non-linear optimization algorithm [31].

e_1: $profit_{(investmentperiod\_e1)} = profit_{(investmentperiod\_medium)} \wedge availability_{(investmentperiod\_e1)} = availability_{(investmentperiod\_medium)} \wedge profit_{(goal\_e1)} = profit_{(goal\_rainydays)} \wedge availability_{(goal\_e1)} = availability_{(goal\_rainydays)} \wedge utility_{(balancedfunds\_e1)} < utility_{(bonds\_e1)}$

e_2: $profit_{(investmentperiod\_e2)} = profit_{(investmentperiod\_medium)} \wedge availability_{(investmentperiod\_e2)} = availability_{(investmentperiod\_medium)} \wedge profit_{(goal\_e2)} = profit_{(goal\_rainydays)} \wedge availability_{(goal\_e2)} = availability_{(goal\_rainydays)} \wedge utility_{(balancedfunds\_e2)} < utility_{(bonds2\_e2)}$

e_3: $profit_{(investmentperiod\_e3)} = profit_{(investmentperiod\_medium)} \wedge availability_{(investmentperiod\_e3)} = availability_{(investmentperiod\_medium)} \wedge profit_{(goal\_e3)} = profit_{(goal\_rainydays)} \wedge availability_{(goal\_e3)} = availability_{(goal\_rainydays)} \wedge utility_{(bonds2\_e3)} < utility_{(bonds\_e3)}$

The overall customer interest in the dimension *profit* is stored in profit(e_i). The values of these variables represent the sum over all defined contributions of customer requirements of example e_i to the dimension *profit*. This approach is analogously applied to the dimension *availability* (availability(e_i)). We denote constraints summing up customer utilities as $s_i \in$ S. The following constraints implement the definitions for e_1, e_2, e_3.

s_1: $profit(e_1) = profit_{(investmentperiod\_e1)} + profit_{(goal\_e1)}$
s_2: $availability(e_1) = availability_{(investmentperiod\_e1)} + availability_{(goal\_e1)}$
s_3: $profit(e_2) = profit_{(investmentperiod\_e2)} + profit_{(goal\_e2)}$
s_4: $\text{availability}(e_2) = \text{availability}_{(investmentperiod\_e2)} + availability_{(goal\_e2)}$
s_5: $profit(e_3) = profit_{(investmentperiod\_e3)} + profit_{(goal\_e3)}$
s_6: $availability(e_3) = availability_{(investmentperiod\_e3)} + availability_{(goal\_e3)}$

For each service part of our example assortment, we specify its contribution to the given interest dimensions. For instance, the shares percentage specified for the service *balancedfunds* defines an average interest in the dimension *profit*. In our CSP, we define this fact as

$$profitshares_{(balancedfunds)} = 5.$$

Analogously, we define the relationship between the interest dimension *availability* and *shares* percentage as

$$availabilityshares_{(balancedfunds)} = 5.$$

We denote each constraint defining a utility value for a certain product (service) as utility constraint $up_i \in$ UC. The following constraints implement the definitions of Tables 13–14.

up_1: $profitshares_{(balancedfunds)} = 5$
up_2: $availabilityshares_{(balancedfunds)} = 5$
up_3: $profitshares_{(bonds)} = 2$
up_4: $availabilityshares_{(bonds)} = 7$
up_5: $profitshares_{(bonds2)} = 2$
up_6: $availabilityshares_{(bonds2)} = 7$
up_7: $profitfluctuation_{(balancedfunds)} = 5$
up_8: $availabilityfluctuation_{(balancedfunds)} = 6$
up_9: $profitfluctuation_{(bonds)} = 5$
up_{10}: $availabilityfluctuation_{(bonds)} = 6$
up_{11}: $profitfluctuation_{(bonds2)} = 7$
up_{12}: $availabilityfluctuation_{(bonds2)} = 4$

For each $uc_i \in$ UC (and each $up_i \in$ UC) we add a corresponding *repair constraint* cr_i (pr_i) which specifies possible repairs for uc_i (up_i). The idea behind repair constraints is that if the utility constraint set is inconsistent with the examples, a non-linear optimization process can identify minimal repairs for uc_i (up_i) which are within the boundaries defined by repair constraints. These repairs should change the original uc_i (up_i) as little as possible.[3] Therefore, we define an interval for the accepted changes for each $uc_i \in$ C and each $up_i \in$ P. Each of the following example repair constraints allows changes of the given evaluations by at most one unit. We denote $\bigcup cr_i \cup \bigcup pr_i$ as set of repair constraints R.

cr_1:$profit_{(investmentperiod\_short)} \in \{3,4,5\}$
cr_2:$availability_{(investmentperiod\_short)} \in \{8,9,10\}$
cr_3:$profit_{(investmentperiod\_medium)} \in \{5,6,7\}$
cr_4:$availability_{(investmentperiod\_medium)} \in \{4,5,6\}$
cr_5:$profit_{(investmentperiod\_long)} \in \{7,8,9\}$
cr_6:$availability_{(investmentperiod\_long)} \in \{0,1,2\}$
cr_7:$profit_{(goal\_rainydays)} \in \{1,2,3\}$
cr_8 $availability_{(goal\_rainydays)} \in \{5,6,7\}$
cr_9:$profit_{(goal\_growth)} \in \{5,6,7\}$
cr_{10}:$availability_{(goal\_growth)} \in \{4,5,6\}$
cr_{11} $profit_{(goal\_speculation)} \in \{8,9,10\}$
cr_{12}:$availability_{(goal\_speculation)} \in \{1,2,3\}$
pr_1:$profitshares_{(balancedfunds)} \in \{4,5,6\}$
pr_2:$availabilityshares_{(balancedfunds)} \in \{4,5,6\}$
pr_3:$profitshares_{(bonds)} \in \{1,2,3\}$
pr_4:$availabilityshares_{(bonds)} \in \{6,7,8\}$
pr_5:$profitshares_{(bonds2)} \in \{1,2,3\}$
pr_6:$availabilityshares_{(bonds2)} \in \{6,7,8\}$
pr_7:$profitfluctuation_{(balancedfunds)} \in \{4,5,6\}$
pr_8:$availabilityfluctuation_{(balancedfunds)} \in \{5,6,7\}$
pr_9:$profitfluctuation_{(bonds)} \in \{4,5,6\}$
pr_{10}:$availabilityfluctuation_{(bonds)} \in \{5,6,7\}$
pr_{11}:$profitfluctuation_{(bonds2)} \in \{6,7,8\}$
pr_{12}:$availabilityfluctuation_{(bonds2)} \in \{3,4,5\}$

The *profit* of a financial service is defined by the sum of contributions of the values of *shares* and *value fluctuation*. *Availability* of a service is as well defined by the sum of related contributions. We denote each rule summing up service utility values as $s_i \in$ S. The following constraints implement the definitions of Table 15.

s_7: $profit_{(balancedfunds)} = profitshares_{(balancedfunds)} + profitfluctuation_{(balancedfunds)}$
s_8: $profit_{(bonds)} = profitshares_{(bonds)} + profitfluctuation_{(bonds)}$
s_9: $profit_{(bonds2)} = profitshares_{(bonds2)} + profitfluctuation_{(bonds2)}$
s_{10}: $availability_{(balancedfunds)} = availabilityshares_{(balancedfunds)} + availabilityfluctuation_{(balancedfunds)}$
s_{11}: $availability_{(bonds)} = availabilityshares_{(bonds)} + availabilityfluctuation_{(bonds)}$
s_{12}: $availability_{(bonds2)} = availabilityshares_{(bonds2)} + availabilityfluctuation_{(bonds2)}$

The following constraints specify the calculation of product utilities, where utility(x_{ei}) specifies the utility of product x in the context of example e_i.

s_{13}: $utility_{(balancedfunds\_e1)} = profit_{(balancedfunds)} \cdot profit(e_1) + availability_{(balancedfunds)} \cdot availability(e_1)$
s_{14}: $utility_{(bonds\_e1)} = profit_{(bonds)} \cdot profit(e_1) + availability_{(bonds)} \cdot availability(e_1)$
s_{15}: $utility_{(bonds2\_e1)} = profit_{(bonds2)} \cdot profit(e_1) + availability_{(bonds2)} \cdot availability(e_1)$
s_{16}: $utility_{(balancedfunds\_e2)} = profit_{(balancedfunds)} \cdot profit(e_2) + availability_{(balancedfunds)} \cdot availability(e_2)$
s_{17}: $utility_{(bonds\_e2)} = profit_{(bonds)} \cdot profit(e_2) + availability_{(bonds)} \cdot availability(e_2)$
s_{18}: $utility_{(bonds2\_e2)} = profit_{(bonds2)} \cdot profit(e_2) + availability_{(bonds2)} \cdot availability(e_2)$
s_{19}: $utility_{(balancedfunds\_e3)} = profit_{(balancedfunds)} \cdot profit(e_3) + availability_{(balancedfunds)} \cdot availability(e_3)$
s_{20}: $utility_{(bonds\_e3)} = profit_{(bonds)} \cdot profit(e_3) + availability_{(bonds)} \cdot availability(e_3)$
s_{21}: $utility_{(bonds2\_e3)} = profit_{(bonds2)} \cdot profit(e_3) + availability_{(bonds2)} \cdot availability(e_3)$

5. Automated Repair of Scoring Rules

All the mentioned constraints are constituting elements of a corresponding nonlinear optimization problem [31] which represents a *Constraint Set Adaptation Problem* (see Definition 3).

[3]Note that changes of at most one unit are only introduced for this example, the range of possible changes is more flexible. In the current implementation it can be specified by knowledge engineers.

Definition 3 (Constraint Set Adaptation Problem). A constraint set adaptation problem is defined by the tuple (V, D, UC, Con, Opt), where V is a set of variables referred to by the constraints in Con = R ∪ S ∪ E, D contains the domain definitions of the variables in V, and UC represents the set of scoring roles inconsistent with the examples in E. Opt is the optimization function of the underlying nonlinear optimization problem.

The constraints Con defined in Section 4 are the basic elements of an optimization problem of minimizing repair distances between original scoring values and corresponding repair values using the following minimization function – see Formula (2).

$$Minimize: \sum_{i=1}^{m} |val(uc_i) - val(cr_i)| + \sum_{j=1}^{n} |val(up_j) - val(pr_j)| \quad (2)$$

In this formula, $|val(uc_i) - val(cr_i)|$ denotes the degree to which the original value of the utility constraint (scoring value for customer requirements) has been changed. Furthermore, $|val(up_j) - val(pr_j)|$ denotes the degree to which the original value of the (product) utility constraint has been changed. Now the Minos solver [31] can calculate a solution to a constraint set adaptation problem (see Definition 4).

Definition 4 (Constraint Set Adaptation). A constraint set adaptation for a given constraint set adaptation problem (V, D, UC, Con, Opt) is an assignment of the variables in V s.t. all constraints in Con are satisfied.

If we want to restrict the proposed repair actions to the utility constraints $up_i \in$ UC ($uc_i \in$ UC), we have to include $\cup uc_i$ ($\cup up_i$) into the set of constraint definitions (Con). The following adaptations (repairs) to the original scoring rules (utility constraints) in UC represent a constraint set adaptation for our example constraint set adaptation problem.

$profit_{(investmentperiod\_short)} = 4$
$availability_{(investmentperiod\_short)} = 9$
$profit_{(investmentperiod\_medium)} = 5$
$availability_{(investmentperiod\_medium)} = 4$
$profit_{(investmentperiod\_long)} = 8$
$availability_{(investmentperiod\_long)} = 1$
$profit_{(goal\_rainydays)} = 2$
$availability_{(goal\_rainydays)} = 5$
$profit_{(goal\_growth)} = 6$
$availability_{(goal\_growth)} = 4$
$profit_{(goal\_speculation)} = 9$
$availability_{(goal\_speculation)} = 2$
$profitshares_{(balancedfunds)} = 5$
$availabilityshares_{(balancedfunds)} = 4.99$
$profitshares_{(bonds)} = 2$
$availabilityshares_{(bonds)} = 6.33$
$profitshares_{(bonds2)} = 2$
$availabilityshares_{(bonds2)} = 7$
$profitfluctuation_{(balancedfunds)} = 5$
$availabilityfluctuation_{(balancedfunds)} = 5$
$profitfluctuation_{(bonds)} = 5$
$availabilityfluctuation_{(bonds)} = 6.22$
$profitfluctuation_{(bonds2)} = 6$
$availabilityfluctuation_{(bonds2)} = 4.67$

The application of these repairs results in the new rankings depicted in Table 18. These rankings are now consistent with $e_i \in$ E.

6. Evaluation

Experiences from Commercial Projects. On the basis of our experiences from commercial recommender projects (see, e.g., [10]) we identified a clear need for more effective engineering techniques in the context of utility constraint development and maintenance. The investment recommender of an Austrian financial service provider (see Figure 1) has been implemented without the repair functionalities presented in this paper. The system comprises 15 parameters for specifying customer requirements, 10 item properties and about 150 scoring rules (interest dimensions: *availability*, *profit*, *risk*). The recommender application has been designed, developed, and deployed with an overall effort of about 12 man months. Before deploying the first version of the application, new versions of the utility constraint set have been released every third week and tested by domain experts. About 15 adaptation cycles were needed before deploying the utility constraint set in the productive environment. Adaptation efforts related to the utility constraint set consumed about 12 hours per adaptation cycle. This results in 180 hours of development and maintenance efforts specifically related to the adaptation of the utility constraint set. In each adaptation cycle, the knowledge engineer tried to adapt the current utility constraint set to be consistent with the example rankings provided by domain experts. The process was error-prone and time-consuming and triggered require-

customer	item	utility	ranking after repair	ranking before repair
Robert	balanced funds	160.004	3	1
	bonds	162.004	1	2
	bonds2	161.004	2	3

Table 18

Utilities of items for customer *Robert* (before and after the repair process). The utilities are now consistent with the examples shown in Table 17.

ments to automate the adaptation process. The major problem was the task of manually detecting a set of repair actions that make a utility constraint set consistent with the set of examples. By exploiting the presented repair functionalities, a reduction of the overall development and maintenance efforts related to utility constraint sets by about 60% (effort directly related to the adaption of the utility constraints) can be expected which means more than 100 hours of time savings in projects similar to the described case.

Optimality and Performance. Optimality properties of solutions to optimization problems are depending on the used optimization approach [20]. We had to deal with a non-linear optimization problem since non-linear constraints are part of the constraint set adaptation problem. Non-linear optimization solvers can not guarantee the optimality of an identified solution [20]. For this reason we had to evaluate the quality (degree of optimality) of results calculated by the Minos Solver [31] which we used for calculating solutions to a constraint set adaptation problem. For our evaluations we used commercial utility constraint sets from the product domains of refrigerators (refrig1-4), financial services (finserv1-4), and computer monitors (mon1-4) (see Table 19). For each of the above mentioned application domains we have defined four different settings which differed in the number of examples (#e) and the number of products (#p). For example, in finserv4 #e=20 examples were defined for #p=71 products. The corresponding utility constraint set comprised #su=503 constraints (scoring rules). In order to make the 20 examples consistent with the given set of scoring rules, #so=340 rules have been adapted with an average change distance avg(d)=0.056 where each scoring rule is defined over the domain [0..10]. The time needed by the Minos solver [31] to calculate the adaptations for finserv4 was t=9464 milliseconds. The Minos solver is capable of calculating adaptations for faulty utility constraint sets within a reasonable time span acceptable for utility constraints engineering scenarios. In such scenarios, the system uses either examples defined by marketing and sales experts or examples automatically derived from existing user interaction logs. In our test settings, we used examples which have been specified manually – the automated derivation of examples is the goal for future work. For a detailed discussion of the handling of constraint set adaptation problems without a corresponding solution (e.g., the set of provided examples is inconsistent) the reader is referred to the work of [8,16].

7. Related and Future Work

Recommendation Approaches. There are three basic recommendation approaches [2]. Collaborative Filtering [14][15] is based on the assumption of the correlation of preferences, i.e., similar products are recommended to customers with similar purchasing behavior in the past. Content-based filtering [21] focuses on the analysis of a given set of products already purchased by the customer. By exploiting historical purchasing data, products are recommended which resemble those already purchased. One of the major advantages of both approaches lies in the simple set-up procedure which does not require complex knowledge acquisition and maintenance. In contrast, knowledge-based approaches [2][6][10] rely on deep knowledge about the offered product and service assortments as well as on deep knowledge about the company's marketing and sales strategy. In this context, the relationship between customer requirements and offered products has to be explicitly modeled in a recommender knowledge base. Deep knowledge representations are the basis for knowledge-based recommendation technologies. They are the major precondition for deriving recommendations, explaining recommendations, and for determining repair actions for inconsistent requirements [18][23]. The repair approach presented in this paper has

rec.	#e	#p	#su	#so	avg(d)	t(msec)
refrig1	5	16	39	39	<0.001	761
refrig2	10	30	69	55	0.085	1221
refrig3	15	43	80	62	0.082	1998
refrig4	20	55	100	74	0.077	2252
finserv1	5	19	266	160	0.070	1622
finserv2	10	37	396	244	0.054	5599
finserv3	15	53	439	288	0.055	6389
finserv4	20	71	503	340	0.056	9464
mon1	5	22	109	40	0.046	731
mon2	10	42	177	75	0.041	1813
mon3	15	61	214	105	0.044	1542
mon4	20	80	246	125	0.047	2063

Table 19

Performance of the Minos solver [31] for the calculation of solutions for our example test settings in three different domains (*refrigerators*, *financial services*, and *computer monitors*) where *#e=#examples*, *#p=#products*, *#su=#scoring rules* (utility constraints), *#so=#scoring rules adapted by the non-linear optimization process*, *avg(d)=average distance to the original scoring values* (before the repair process has been started), *t(msec) = calculation time in milliseconds.*

been designed for the application in constraint-based recommendation scenarios.

Utility-based Ranking of Repair Alternatives. The utility-based evaluation of different decision alternatives is not restricted to the ranking of product alternatives (which was the focus of discussion in this paper). If no solution can be found for a given set of customer requirements, *repair actions* represent minimal changes to those requirements such that the calculation of a corresponding recommendation is possible [10,28]. If we assign a priority (represented as utility value) to each customer requirement we can apply MAUT concepts for evaluating which of the alternative repairs should be first proposed to the customer [10]. Such personalized rankings can alleviate the retrieval of products and contribute to a significantly lower number of interaction cycles with the recommender. Utility values can be either predefined or directly specified by the customer. In contrast to the ranking of products, the importance of customer requirements negatively influences the ranking of a repair alternative, i.e., the utility of a repair alternative decreases if the corresponding requirements are of importance to the customer. At this point we also want to emphasize that ethical aspects play an important role when applying the concepts presented in this paper. Ranking examples can also be misused for pushing the sales of specific items – a detailed discussion of related biasing effects can be found, for example, in [11].

Utility-based Ranking of Explanations. When presenting recommendation results, each product part of the result set has a corresponding list of explanations [12] (e.g., argumentations as to why this product fits to the wishes and needs of a customer). Similar to products, explanations can be as well ordered conform to a predefined set of interest dimensions, i.e., the most interesting explanations are presented first. Such rankings can increase the trust in explanations and the corresponding product recommendations [5]. Rankings can be easily calculated since each recommended product is directly connected to filter constraints responsible for the selection of this product. The higher the overall importance of the customer requirements referenced in a filter constraint, the higher is the utility of the explanation for the customer. Thus the ranking of explanations plays an important role in the configuration of argumentation chains where the ordering of sub-arguments has a major influence on whether the main argument is accepted by the customer or not (see, e.g., [5]). The application of the concepts presented in this paper to the automated repair of scoring rules used for the ranking of *explanations* is within the focus of future work.

Repairing Inconsistent Constraint Sets. The repair concepts presented in this paper are completely different from those presented in [9]. The work of [9] focuses on the automated identification of faults in recommender user interface descriptions which requires a different knowledge representation in terms of finite state models and a diagnosis algorithm based on conflicts induced by navigation patterns in finite state automata. Compared to the work of [9], the approach presented in this paper determines repair actions on the basis of the results of a non-linear optimization process. Such a type of optimization requires specific constraint representations which allow to take into account the set of utility constraints as well as all examples provided by knowledge engineers and domain experts. The work presented in [7] follows a pure learning-based approach where preference functions are learned on the basis of examples. These functions are subsequently used to order a set of new solutions. In contrast to pure learning-based approaches, our approach defines a kind of ranking seed knowledge (the initial utility constraint set) which is subsequently automatically repaired to conform with new situations. Thus we avoid cold start problems and effectively support continuous updates of utility constraint sets. The work of [1] deals with modeling constraint problems using soft constraints. In this context, solution preferences are treated as examples which are exploited for the learning of constraint preferences. Compared to our work, [1] apply a learning algorithm that does not require the consistency between the given set of solution preferences and the learned constraint preferences. In this case, the optimality of a solution strongly depends on the accepted/supported solution preferences. Finally, [8] present an approach to the identification of *minimal sets* of scoring rules which should be adapted in order to recover consistency between the examples $e_i \in$ E and the defined set of scoring rules. The major difference between the work presented in this paper and the approach of [8] is the focus on the determination of *minimal-cardinality sets of scoring rules* which have to be adapted in order to achieve consistency between examples and utility constraint set. This is accomplished using the concepts of model-based diagnosis that calculates minimal sources of inconsistencies on the basis of pre-calculated conflict sets [8]. Minimality in the sense of [8] is interpreted in terms of the number of identified faulty utility constraints (scoring rules) whereas minimality in the sense of this paper is interpreted as minimal distance between the original scoring definitions and the scoring definitions in the adapted set of scoring rules – see Formula (2). Both approaches have their advantages. The approach presented in [8] can be applied to identify typical modeling faults and the approach of this paper can be used to support periodical update processes which keep the new set of scoring rules as near as possible to the original version.

Future Work. We will evaluate our repair approach in *further product and service domains.* In this context we will extend the application of our approach to the *repair of faulty rankings of explanations* [12] and *faulty rankings of repair actions for inconsistent customer requirements* [10]. Furthermore, we will focus on the development of concepts that enable the *automated generation of ranking examples* from user interaction logs (e.g., on the basis of association rule mining). Finally, we will *extend the expressivity of ranking examples* by allowing to define further criteria on the ranking of items in a result set, such as the minimal and maximal utility distance between specific items, the minimal and maximal allowed ranking of an item, and ranking alternatives.

8. Conclusion

We have presented knowledge engineering techniques which support knowledge engineers and domain experts in the development of utility constraint sets used for the calculation of item rankings (the ranking of products and services on the result page of a recommender). We have proposed an approach to the identification and repair of faulty elements in utility constraint sets which is based on an application of non-linear optimization methods. Summarizing, the presented concepts are an important step towards more effective knowledge-based systems development processes.

Acknowledgements

The work presented in this paper has been conducted within the scope of the research projects ICONE (827587) and Casa Vecchia (825889) funded by the Austrian Research Promotion Agency.

References

[1] A. Biso, F. Rossi, and A. Sperduti. Experimental Results on Learning Soft Constraints, 7th Intl. Conf. on the Principles of Knowledge Representation and Reasoning (KR02), Breckenridge, CO, pp. 435–444, 2000.

[2] R. Burke. Knowledge-based Recommender Systems. Encyclopedia of Library and Information Systems, 69(32):180–200, 2000.

[3] R. Burke. Hybrid Web Recommender Systems. in: The adaptive web: methods and strategies of web personalization, P. Bruzsilovsky, A. Kobsa, and W. Nijdl, eds., pp. 377–408, 2007.

[4] R. Burke. A. Felfernig, and M. Goeker. Recommender Systems: An Overview. AI Magazine, AAAI, 32(3):13–18, 2011.

[5] G. Carenini and J. Moore. Generating and evaluating evaluative arguments. AI Journal, 170:925–952, 2006.

[6] L. Chen, and P. Pu. Survey of Preference Elicitation Methods, EPFL. Technical Report, 2004.

[7] W. Cohen, R. Schapire, and Y. Singer. Learning to order things. Journal of AI Research, 10:243–270, 1999.

[8] A. Felfernig, E. Teppan, G. Friedrich, and K. Isak. Intelligent Debugging and Repair of Utility Constraint Sets in Knowledge-based Recommender Applications, 13th ACM International Conference on Intelligent User Interfaces, Canary Islands, Spain, Jan. 13-16, pp. 218–226, 2008.

[9] A. Felfernig and K. Shchekotykhin. Debugging User Interface Descriptions of Knowledge-based Recommender Applications. ACM Conference on Intelligent User Interfaces, IUI06, pp. 234–241, 2006.

[10] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. An Environment for the Development of Knowledge-based Recommender Applications, International Journal of Electronic Commerce (IJEC), 11(2):11–34, 2006.

[11] A. Felfernig, G. Friedrich, B. Gula, M. Hitz, T. Kruggel, R. Melcher, D. Riepan, S. Strauss, E. Teppan, and O. Vitouch. Persuasive Recommendation: Exploring Serial Position Effects in Knowledge-based Recommender Systems, Persuasive 2007, Stanford, California, Springer Lecture Notes in Computer Science, pp. 283–294, 2007.

[12] A. Felfernig and R. Burke. Constraint-based Recommender Systems: Technologies and Research Issues, ACM International Conference on Electronic Commerce (ICEC'08), Innsbruck, Austria, Aug. 19-22, pp. 17-26, 2008.

[13] F. Gershberg and A. Shimamura. Serial position effects in implicit and explicit tests of memory. Journal of Experimental Psychology: Learning, Memory, and Cognition, 20:1370–1378, 1994.

[14] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems, 22(1):5–53, 2004.

[15] Z. Huang, D. Zeng, and H. Chen. A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce. IEEE Intelligent Systems, 22(5):68–78, 2007.

[16] U. Junker. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. 19th National Conference on AI, pp. 167–172, 2004.

[17] R. Keeney, H. Raiffa. Decisions with Multiple Objectives: Preferences and Value Tradeoffs, John Wiley and Sons, 1976.

[18] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems, Artificial Intelligence 56 (2-3) pp. 197–222, 1992.

[19] K.S. Lashley. The problem of serial order in behavior. In L. A. Jeffress (Ed.), Cerebral mechanisms in behaviour (pp. 112–136). New York: Wiley, 1951.

[20] A. Neumaier, O. Shcherbina, W. Huyer and T. Vinko. A comparison of complete global optimization solvers, Mathematical Programming, 103(2):335–356, 2005.

[21] M. Pazzani and D. Billsus. Content-based Recommender Systems. in: The adaptive web: methods and strategies of web personalization, P. Bruzsilovsky, A. Kobsa, and W. Nijdl, eds., pp. 325–341, 2007.

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

[23] R. Reiter. 1987. A theory of diagnosis from first principles. AI Journal, 23, 1, 57–95, 1987.

[24] P. Resnick and H. Varian. 1997. Special Issue on Recommender Systems. Communications of the ACM, 40, 1997.

[25] C. Schmitt, D. Dengler, and M. Bauer. Multivariate Preference Models and Decision Making with the MAUT Machine. User Modeling, 297–302, 2003.

[26] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. in: The adaptive web: methods and strategies of web personalization, P. Bruzsilovsky, A. Kobsa, and W. Nijdl, eds., pp. 291–324, 2007.

[27] R. Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Maher, M. Cox, K. Forbus, M. Keane, A. Aamodt, and I. Watson. Retrieval, reuse, revision, and retention in case-based reasoning, Knowledge Engineering Review, 20(3):215–240, 2005.

[28] B. O'Sullivan, A. Papadopoulos, B. Faltings, P. Pu: Representative Explanations for Over-Constrained Problems. 22nd National Conference on AI, 323–328, 2007.

[29] C. Thompson, M. Goeker, and P. Langley. A Personalized System for Conversational Recommendations. Journal of AI Research 21:393–428, 2004.

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

[31] R. Fourer, D. Gay, and B. Kernighan. AMPL: A Modeling Language for Mathematical Programming, Cole Publishing Company, 2002.

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