

Preference reasoning with global soft constraints in constraint-based recommender systems

Markus Zanker · Markus Jessenitschnig ·
Wolfgang Schmid

Received: date / Accepted: date

Abstract A recommender system (RS) supports online users in e-commerce by proposing products that are assumed to be both useful and interesting. Knowledge-based recommendation systems form one branch of these online sales support systems that is particularly relevant for high-involvement product domains like consumer electronics, financial services or tourism. A constraint-based RS is a specific variant of a knowledge-based RS that builds on a CSP formalism for problem representation and solving. This article formalizes the different variants of a *constraint-based* recommendation problem based on consistency and the empirical part compares the performance of different constraint-based recommendation mechanisms in offline experiments on historical data.

Keywords Recommendation systems · Constraint-based recommendation · Evaluation · Application

1 Introduction

Product recommenders are a popular tool for web-based consumer platforms. Of the different paradigms for the computation of recommendations that exist [1], collaborative filtering is the most well-known recommendation mechanism that exploits clusters of similar users to recommend items to a user based on his/her most similar peers. Constraint-based recommendation is an alternate computation paradigm that employs an explicit representation of domain knowledge that conforms to the CSP formalism [2–4]. While collaborative filtering assumes that users can be clustered into different segments based on

Universität Klagenfurt
Universitätsstrasse 65-67, 9020 Klagenfurt, Austria
Tel.: +43-463-2700-3753
Fax: +43-463-2700-3799
E-mail: {markus.zanker,markus.jessenitschnig,wolfgang.schmid}@uni-klu.ac.at

their observed behavior, constraint-based recommendation provides expert advice in more complex domains that involve high-involvement products like, for instance, consumer electronics, financial services or tourism. Several studies have shown that systems should offer different elicitation styles for identifying users' preferences like feature-oriented or needs-oriented dialogues in order to be capable to deliver good search results and a satisfying user experience [5, 6].

The constraint-based RS requires explicit knowledge about the user's specific requirements (termed *SRS* in the following), a finite catalog of items from which recommendable items can be selected and additional domain knowledge representing sales expertise that encodes for instance which item features help to fulfill a specific user requirement. Note that the latter is necessary for supporting needs-oriented interaction styles where users are allowed to formulate more abstract requirements like - for instance in the digital camera domain - *'wants to use the camera mainly for travel'* and the system has to infer which lens is most suitable for this purpose. The goal is consequently to identify a number of catalog items that best match a user's requirements according to the encoded sales knowledge.

Two different conversational approaches may be used depending on the system's interaction paradigm: (1) requirements elicitation by questioning [2, 3] and (2) critiquing systems [7–10]. While the rationale behind (1) is to mimic an experienced sales clerk that initially identifies the customer's needs, the benefit of (2) is that it incorporates a tradeoff reasoning between the multiple objectives the customer might have. Obviously, both conversational approaches can be combined in real systems, where an initial asking phase that elicits the most obvious preferences is followed by several refinement cycles of the preference model where proposed items are critiqued by users in order to learn and revise their preference model [11]. Conversational constraint-based recommendation typically constitutes an over-constrained problem and, for instance, Pu and Faltings showed that soft constraints are very suitable for reasoning on such preference models [10].

In situations where the constraint-based RS system detects that the preference model is over-constrained it may choose between two possible reactions:

1. Weaken or relax some of the preferences: Finding a solution that satisfies a subset of the constraints comparable to partial constraint satisfaction and max-csp [12], finding maximally succeeding subqueries [13] or diagnosing conflict sets [3, 14, 15].
2. Revise the preference model: This is done, for instance, by providing trade-off scenarios [16], revising preferences and their weights [10] or proposing repair actions [17].

This article focuses solely on the first strategy - the automated relaxation of some preferences - in order to compute recommendable solutions. Its contribution is therefore a formalization of the commonly applied relaxation strategies of a constraint-based recommendation problem [4, 17] that is in-line with the recognized view of a RS being a function [18]. Empirical evaluations show im-

improvements in terms of accuracy that are documented by offline experiments on commercial datasets of historical user sessions.

The next section examines related work, while Section 3 introduces a motivating example. In Section 4 a computation scheme for constraint-based recommendation is defined. Finally, the evaluation methodology and the results are discussed in Section 6.

2 Related work

A variety of different knowledge-based conversational RS have been proposed. In order to categorize them, Figure 1 sketches a simple reference model that is a slightly modified version of the one developed by Pu & Faltings [10]. Initially, the system possesses some domain knowledge that would support a needs-based requirements elicitation from users, a detailed description of the product catalog (both depicted by the data store) and the specific requirements entered when users start the interaction. In case the problem is already overconstrained at this point a strategy for constraint relaxation is needed in order to compute a list of recommendations. In a second phase the user can iteratively critique these solution candidates, make trade-off decisions and revise her/his preferences which will lead to an updated preference model or accept one of the solutions (or simply exit the conversation).

One of the first systems based on such a candidate/critique interaction style was the *Automated Travel Assistant* presented by [7]. An additional early example is the Findme system [19] that offers a conversational interface that

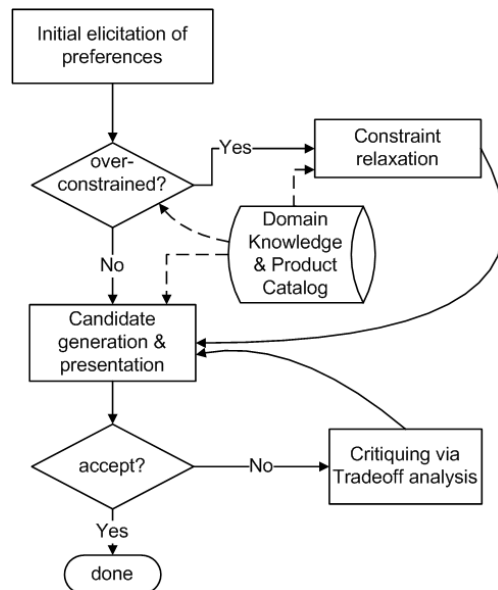


Fig. 1 Reference model for constraint-based conversational RS

allows users to formulate a critique on a specific item and retrieves more appropriate alternatives based on similarity measures. The Smartclient approach [20] is based on the Java Constraint Library that allows efficient client-side exploration of the search space for, e.g., travel planning when constraints like flight and hotel availabilities can be retrieved in bulk from different servers. In [10] Pu & Faltings proposed an example critiquing interaction and empirically showed that it supports users in making trade-off decisions better than ranked product lists. In addition, the study of [21] confirmed that critiquing leads to more accurate matches of user preferences than a sole question-based interaction. Therefore, such critique systems have been further developed to consider users' critiquing history [22] or to learn compound critiques [23] that critique different product features frequently occurring together in the catalog in one round of interaction. Furthermore, [24] introduce adaptive suggestions to elicit critiques that will help to improve the preference model and learn about the user's underlying utility model.

However, in practice when identifying user preferences a variety of influence factors that derive from the configuration of the recommendation set itself can occur that have to be controlled. For instance, within a set of different choices different psychological effects like asymmetric dominance or the existence of Decoy products influence users in their decision making [25]. Consequently, soft constraints have their limitations in modeling preferences as they are building on the assumption that preference weights can be linearly added and neglect interactions between preferences for different features. Rossi and Sperduti [26], for instance, exploited local or solution preferences in a critiquing-based system. These *qualitative* preferences denote that a user prefers *ceteris paribus* item A to B (i.e. with everything else remaining equal) and can therefore model conditional dependencies.

However, in the first phase of the interaction scenario as depicted in Figure 1 no qualitative preferences are yet known and therefore global soft constraints remain to be the natural choice. The relaxation of constraints when confronted with an overconstrained CSP is a well studied problem. Freuder and Wallace [12] proposed a general model for partial constraint satisfaction that they seek to solve by satisfying a maximal number of constraints. This is comparable to the work of McSherry [13] who analogously to Freuder and Wallace formalized the process of identifying all maximally succeeding sub queries (XSS) in order to repair a retrieval failure and find solutions that fulfill only a subset of the initial criteria. An alternate third approach towards relaxing an overconstrained retrieval problem has been proposed by Jannach in [2,27] and Felfernig et al. [3]. They use a linear weighting scheme that guides the search for XSS. In [14] an optimized version for determining maximally succeeding sub-queries is proposed that avoids an exponential number of database queries by explicitly enumerating an item/constraint matrix in a bitmap structure in memory and computes minimal diagnoses that constitute sets of constraints that have to be relaxed. Jannach has described a system that implements this approach in [28].

When applying the aforementioned approaches to recommendation they are implicitly building on the assumption that all recommended items have to fulfill an identical set of constraints, i.e. all recommended items are different solutions for a CSP with the same set of its constraints relaxed. However, we propose to remove this assumption and to treat the relaxation and the generation of recommendations as a model checking problem that requires only a linear number of consistency checks. By an experimental evaluation it will be further demonstrated that this improves the ranking function of the recommended items which in turn leads to an increased accuracy. The next section sketches a motivating example.

3 Motivating example

Constraint-based recommender systems reason on user preferences and explicit domain knowledge to make personalized product suggestions. They rely on heuristics and business rules often formulated as implication constraints that have been acquired from domain experts during the setup phase of a system. The basic idea is to model and automate the behavior of experienced sales agents in routine sales advisory tasks and build cost-efficient and easily accessible applications for customers [29]. For instance in domains like financial services, where agents may be held liable for the advice they are giving, automated and quality-assured sales dialogues have been successfully deployed [3].

Continuing with the example of digital camera recommendation, an experienced sales agent would typically not propose a camera model merely because it is on sale but instead would try to ascertain the customer’s needs first. Typical questions in this example domain include *What type of scenes do you want to photograph?* or *How much can you afford to spend?* An example user

Variables	Value
scenes:	<i>landscape</i>
brand_preference:	<i>Canon</i>
max_cost:	<i>350 EUR</i>
min_cost:	<i>200 EUR</i>
storage_media:	<i>don't care</i>
Codes (identifiers) of rated items (S):	{318}

Table 1 Example user model

model from an historical user session containing the answers to these questions is depicted in Table 1. These explicit user requirements can be utilized by a constraint-based algorithm and the subsequently positively rated items (S) serve as success criteria for the evaluation described in Section 6. Fur-

thermore, the fictitious product catalog (see Table 2) lists two products with identifiers 296 and 318; Table 3 contains three exemplary soft constraints.

ID	Variables	Value
296	name:	<i>Powershot XY</i>
	brand:	<i>Canon</i>
	lower_focal_length:	<i>35</i>
	upper_focal_length:	<i>140</i>
	price:	<i>420</i>
318	name:	<i>Lumix</i>
	brand:	<i>Panasonic</i>
	lower_focal_length:	<i>28</i>
	upper_focal_length:	<i>112</i>
	price:	<i>319</i>

Table 2 Example product catalog

The first and third soft constraints denote that the user’s brand and price preferences should be observed. The second preference rule bridges the gap between the abstract user requirement of taking pictures of landscapes and quite technical product characteristics like a 35mm camera equivalent minimal focal length of 28mm or below (a wide-angle lens). Such constraints stem directly from the recommendation knowledge of sales experts.

When trying to fulfill all user requirements shown in Table 1 within the limitations of the given catalog of items (see Table 2) and the set of constraints (see Table 3), an empty result set has to be returned. However, in online sales situations best-practice suggests that empty result sets should be avoided and instead the original query should be refined. For instance, the recommender presented in [3] determines maximally succeeding subqueries by relaxing constraints with lowest weights in a stepwise fashion until at least a single item can be retrieved. In our example the preference rules on price and focal length will have to be discarded in order to recommend at least one item, namely number 296 that fulfills the brand requirement.

Weight	Preference rule
25	If TRUE then brand = brand_preference.
20	If scenes CONTAINS <i>landscape</i> then lower_focal_length \leq 28.
15	If TRUE then price \leq max_cost.

Table 3 Example constraints

In contrast, a preference reasoning mechanism that determines the sum of weights of the constraints that an item satisfies on an individual basis produces a ranked list of two items: item 318 fulfills two constraints with an aggregated weight score of 35 and item 296 satisfies a single preference rule with weight 25. Over the next sections, we will formally define this scheme for preference reasoning and compare it with other approaches.

4 Constraint-based recommendation

A standard constraint satisfaction problem (CSP) is described by a tuple (X, D, C) where X is a set of variables, D a set of finite domains for the variables in X and $C = \{c_1, \dots, c_m\}$ a set of constraint restrictions representing a knowledge base that defines which combinations of values can be simultaneously assigned to variables [30]. Let $vars(c)$ be a function returning the set of variables that are restricted by c and $vars(c) \subseteq X$. The commonly agreed task of a recommender system [18] is to identify the k most suitable items from a given catalog for the targeted user. The CSP formalism may be used to represent a constraint-based recommendation task (*CSPRecTask*) [2–4], where X is a set of variables modeling item features $X_I \subset X$ such as *name*, *brand* or *price* (see the motivating example in Section 3) as well as user and session characteristics $X_U \subset X$ like *brand-preference* or the *scenes* the user would like to photograph. Note that contextual parameters like *season* or *day of week* can also be utilized depending on the application context. The variables’ domain D can be defined as a function $dom(x)$ that returns a finite number of valid variable assignments for each $x \in X$, e.g. $dom(scenes) = \{landscape, people, unknown\}$.

Finally, a constraint $c \in C$ formulates a restriction on the variable assignments that can be evaluated to be either true or false. Constraints usually stem directly from the user, representing his or her needs and preferences, typically in the form of unary constraints on variables like *brand-preference* = *Canon* or *scenes* = *landscape*, but can in principle be arbitrarily complex, limited only by the capabilities of the constraint solver and the user interface. In the following, we denote the set of constraints that has been entered by the user as specific requirements $SRS \subset C$. Furthermore, additional domain knowledge that represents expertise and heuristics for matching items with prospective customers in a specific product domain can be considered. It is typically acquired with the help of domain experts like for instance in [31, 3, 32]. We denote the set of these additional constraints as $KB \subset C$. Constraints in KB can define rather obvious restrictions such as ensuring that proposals only include items of the brand the user prefers (*brand* = *brand-preference*) or deeper domain insights like *scenes* = *landscape* \rightarrow *low-focal.length* \leq 28.

The third set of constraints $I \subset C$ only contains a single restriction that encodes the product catalog in disjunctive normal form, i.e. $I = \{(name = Powershot \wedge brand = Canon \wedge lower_focal_length = 35 \wedge upper_focal_length =$

$140 \wedge price = 420) \vee \dots \vee (name = \dots)$. Thus, every item is represented by a conjunction of its assigned feature values.

Summarizing, the set of constraints is partitioned into specific requirements, domain knowledge and a constraint representing the catalog of items, i.e. $C = SRS \cup KB \cup I$ and $\forall S_1, S_2 \in \{SRS, KB, I\} (S_1 \neq S_2 \rightarrow S_1 \cap S_2 = \emptyset)$.

Definition 1 Solution to CSPRecTask A solution to a CSPRecTask represented by the CSP $(X_I \cup X_U, D, SRS \cup KB \cup I)$ with $C = SRS \cup KB \cup I$ requires finding an assignment tuple θ for all variables in X_I , such that for all $x \in X_I (x = v) \in \theta \wedge v \in dom(x)$ and every constraint $c \in C$ is satisfied under θ .

Definition 2 Satisfiable CSPRecTask A CSPRecTask represented by the CSP (X, D, C) is satisfiable iff a solution exists.

As a prerequisite we assume that the CSP $(X, D, KB \cup I)$ without any specific requirements deriving from the user is satisfiable, otherwise the knowledge base represented by KB needs to be debugged and corrected as proposed in [33, 3].

In situations where the CSPRecTask turns out to be overconstrained several alternatives exist to simply doing nothing and returning an empty set of recommended items:

1. Request revisions of SRS
2. Propose/Select repair alternatives
3. Relax restrictions in $SRS \cup KB$

Requesting users to reformulate their initial inquiry (1.) has been proposed for instance in Mirzadeh et al. [16], who support users in such situations by stating which requirements should be modified to produce a non-empty result list. This case-based reasoning approach computes similarities between the initial inquiry and cases in the catalog that could be retrieved if the inquiry is revised.

(2.) Felfernig et al. [17] apply Model-Based Diagnosis [34] to identify minimal sets of unfulfillable requirements from SRS . This is particularly of interest in situations where already the CSP $(X, D, SRS \cup KB)$ without considering the constraints that derive from the catalog of items (I) is not satisfiable.

Definition 3 SRS Diagnosis A Diagnosis Δ with $\Delta \subseteq SRS$ for a CSPRecTask represented by the CSP $(X, D, SRS \cup KB \cup I)$ is a subset of requirements s.t. the CSP $(X, D, SRS \setminus \Delta \cup KB \cup I)$ is satisfiable. Diagnosis Δ is minimal, iff no diagnosis Δ' exists such that $\Delta' \subset \Delta$.

Diagnoses can be computed by applying Reiter's hitting set algorithm [34] in order to resolve conflict sets. Conflict Sets are defined according to [35, 17].

Definition 4 Conflict Sets CS A Conflict Set $CS = \{c_1, \dots, c_k\}$ with $CS \subseteq SRS$ for a CSPRecTask represented by the CSP $(X, D, SRS \cup KB \cup I)$, where CSP $(X, D, CS \cup KB \cup I)$ is not satisfiable. The Conflict Set CS is minimal, iff no Conflict Set $CS' \subset CS$ exists.

Minimal conflict sets can be computed efficiently for instance by using Junker’s recent QUICKXPLAIN algorithm [35]. Ultimately, the purpose of computing diagnoses of faulty specific user requirements can be to identify repair alternatives. Repair actions are a set of constraints Δ_{repair} that substitute a diagnosis Δ such that the CSP $(X, D, SRS \setminus \Delta \cup \Delta_{repair} \cup KB \cup I)$ is satisfiable. The system could either propose several repair alternatives to the user or automatically select one. For instance, Felfernig et al. [17] identify plausible repair alternatives based on past user interaction records.

(3.) Relaxing some restrictions from $SRS \cup KB \subseteq C$ actually constitutes a partial CSP [12] where the maximal number of constraints $C_{max} \subseteq SRS \cup KB$ that still do not overconstrain the CSPRecTask is determined.

Definition 5 Solution to MAX CSPRecTask A MAX CSPRecTask represented by the CSP $(X, D, SRS \cup KB \cup I)$ finds a set of constraints C_{max} s.t. an assignment tuple θ for all variables in X_I exists with for all $x \in X_I$ $(x = v) \in \theta \wedge v \in dom(x)$ and every constraint $c \in C_{max} \cup I$ is satisfied under θ . Furthermore, there must not exist a $C'_{max} \supset C_{max}$ such that every constraint $c \in C'_{max} \cup I$ is also satisfied under θ .

The same search problem is also known as identifying a maximal set of succeeding sub-queries (XSS) [13] or finding minimal diagnoses [3, 14]. In [12, 13, 3] branch and bound algorithms are proposed that explore the search space in a depth-first manner and require an exponential number of consistency checks in the worst case due to the nature of the problem.

However, the problem of constraint-based recommendation is to identify items from a fully specified catalog of items (in contrast to configuration problems where evaluations of product features have to be computed). Therefore, we propose that the problem is reformulated as a model checking task [36] that can be sequentially performed for every item in the catalog. All possible solutions to the recommendation problem have to be an assignment tuple θ for all variables in X_I (see Definition 1). We introduce Θ as the set of all possible models $\theta_1, \dots, \theta_p$ according to the given catalog of p items, where each $\theta_i \in \Theta$ is an assignment tuple that assigns to the variables in X_I the feature values for the i -th item from the catalog. We define a constraint-based recommendation task based on model checking (MCRRecTask) as follows:

Definition 6 Solution to MCRRecTask A MCRRecTask is represented by the Theory $T = SRS \cup KB$ and the assignment tuple (Model) $\theta_i \in \Theta$ that contains assignments for all variables in X_I , s.t. for all $x_j \in X_I$ an assignment $(x_j = v_{i,j}) \in \theta_i$ and $v_{i,j}$ is the evaluation of attribute x_j for item i from the product catalog. If and only if θ_i is a valid model for T , i.e. $\theta_i \models T$, then θ_i is a solution for the *MCRRecTask*.

Next, Theorem 1 states that a solution to a CSPRecTask is equivalent to a solution of the MCRRecTask and vice versa.

Theorem 1 *Let MCRRecTask with Theory $T = SRS \cup KB$ and possible Models $\theta_i \in \Theta$ and CSPRecTask represented by the CSP $(X_I \cup X_U, D, SRS \cup KB \cup I)$*

be constraint-based recommendation tasks. θ is a solution to the CSPRecTask iff there exists a $\theta_i \in \Theta$ s.t. $\theta = \theta_i$ and θ_i is a solution to MCRRecTask.

The idea of the proof is that constraint I that models the product catalog is in disjunctive normal form where each disjuncted sentence restricts the solution space to a single model representing a specific item from the product catalog. Since every item from the catalog is a possible model and I restricts the solution space to only items from the catalog both MCRRecTask and CSPRecTask have to find identical solutions.

Proof (\implies) $\theta \models T$ therefore $\theta \cup T$ has to be satisfiable. Since I has to be satisfiable by definition and is in disjunctive normal form, i.e. $I = \{\bigvee_{i \in \{1 \dots |\Theta|\}} \text{sentence}_i\}$, where each sentence_i restricts the solution space to exactly one model θ_i from Θ and $\forall i \in \{1 \dots p\} \theta_i \models \text{sentence}_i$, it follows that $I \cup \theta$ is satisfiable and consequently $T \cup I \cup \theta$ must be satisfiable.

(\impliedby) Since $\theta \cup SRS \cup KB \cup I$ is satisfiable also $\theta \cup SRS \cup KB$ has to be satisfiable. Since CSPRecTask assigns all variables in X_I and θ contains assignments for all $x \in X_I$ it follows that $\exists^1 \theta' \in \Theta$ for which $\theta' = \theta$. From this it follows that $\theta \models SRS \cup KB$.

The goal of the MAX MCRRecTask is to identify a maximal set of constraints T_{max} from a theory such that an arbitrary item from the catalog $\theta_i \in \Theta$ becomes a model for T_{max} .

Definition 7 Solution to MAX MCRRecTask A MAX MCRRecTask represented by the Theory $T = SRS \cup KB$ and the assignment tuple (Model) $\theta_i \in \Theta$ finds a set of constraints $T_{max} \subseteq T$ s.t. θ_i contains assignments for all variables in X_I with for all $x_j \in X_I$ an assignment $(x_j = v_{i,j}) \in \theta_i$ and $v_{i,j}$ is the evaluation of attribute x_j for item i from the product catalog. Furthermore, θ_i must be a valid model for T_{max} , i.e. $\theta_i \models T_{max}$, and no $T'_{max} \supset T_{max}$ and $T'_{max} \subseteq T$ exists s.t. $\theta_i \models T'_{max}$. θ_i is a solution for the MAX MCRRecTask that relaxes T to T_{max} .

In analogy to Theorem 1, Theorem 2 states that a solution to a MAX CSPRecTask that relaxes constraints $SRS \cup KB$ to C_{max} is also a solution to a MAX MCRRecTask that relaxes T to T_{max} and $T_{max} = C_{max}$.

Theorem 2 Let MAX MCRRecTask with Theory $T = SRS \cup KB$, subset $T_{max} \subset T$ and Model $\theta_i \in \Theta$ and MAX CSPRecTask represented by the CSP $(X_I \cup X_U, D, SRS \cup KB \cup I)$, subset $C_{max} \subseteq SRS \cup KB$ and solution θ be both constraint-based recommendation tasks with constraint relaxations. θ is a solution to the MAX CSPRecTask iff there exists a $\theta_i \in \Theta$ s.t. $\theta = \theta_i$, $C_{max} = T_{max}$ and θ_i is a solution to MAX MCRRecTask that relaxes T to T_{max} .

The proof idea is that both C_{max} and T_{max} are subsets of $SRS \cup KB$. As according to Definitions 5 and 7 both are maximal they also need to be equivalent.

Proof In the proof of Theorem 1, T has to be replaced by T_{max} . Then it remains to be shown that $T_{max} \subseteq C_{max}$ and $T_{max} \supseteq C_{max}$.

Since $C_{max} \cup I \cup \theta$ has to be satisfiable and there is no $C'_{max} \supset C_{max} \wedge C'_{max} \subseteq C$ where CSP $(X, D, C'_{max} \cup I)$ is satisfiable by definition, it follows $C_{max} \supseteq T_{max}$. Furthermore, since T_{max} is also maximal by definition it follows that $T_{max} \supseteq C_{max}$ and consequently $C_{max} = T_{max}$.

From Theorem 1 it follows that the CSPRecTask and the MCRecTask will find the same solutions if they exist and Theorem 2 shows that in case of relaxation all solutions identified by MAX CSPRecTask are also solutions for MAX MCRecTask. However, most constraint-based recommendation systems based on the MAX CSPRecTask paradigm compute only a single (preferred) constraint relaxation and present the solution candidates [13,14,3]. Thus the constraint-based recommendation system itself can only assign binary scores to the items in the catalog: 1 if an item satisfies the query conditions and 0 otherwise. As a result, these systems have to employ a cascading hybridization scheme by using a second algorithm to refine the recommendation list and perform an additional ranking [1]. For instance, the CWAdvisor system [2,37] uses a scheme based on multi-attribute utility theory (MAUT) [38]. Zanker and Jessenitschnig [39] provided evidence that cascading knowledge-based and collaborative recommendation algorithms leads to superior results compared to a weighting hybridization strategy partly due to the binary scores of the knowledge-based recommender based on a MAX CSPRecTask.

However, the hybridization of constraint-based recommendation with another approach comes at the cost of additional engineering as well as knowledge acquisition and maintenance effort. Therefore, we propose to invoke the MAX MCRecTask for all items in the catalog and utilize a function that scores items based on the set of satisfied or relaxed constraints T_{max} and $T \setminus T_{max}$ respectively. This way a constraint-based RS can emulate a scoring function as postulated by Adomavicius and Tuzhilin [18], who defined a recommender system as a function $rec(i, u)$ that computes a recommendation score for a given item i and user u , expressing how useful or interesting the item will be. The RS will then typically return the k highest scored items. Note, that this behavior is not always supported if the system only computes a single (preferred) constraint relaxation and therefore returns possibly less than k recommendations.

One primitive scoring approach is to use the cardinality of the satisfied/relaxed constraint set. Assigning weights to constraints as proposed by [10] is the most popular approach as mentioned in Section 3. We assume a function $w(c) \in [0, \dots, 100]$ returning a preference weight for every constraint c that constitutes a partial ordering, where $w(c_1) > w(c_2)$ means relaxing c_2 is preferred over relaxing c_1 . Calculating with preference weights of soft constraints builds on the basic assumption that these weights are mutually independent and may thus be combined through addition. This assumption may be challenged, particularly in circumstances where users have conditional preferences. However, the presented MAX MCRecTask approach can easily be extended with more complex and dynamic weighting strategies for preferences.

In the following we present an algorithm for invoking the MAX MCRecTask for each item from the catalog that returns the k highest scoring items. For this purpose we present which premises we assume:

1. Every constraint c that restricts variables from X_I (i.e. $vars(c) \cap X_I \neq \emptyset$) has to be formulated as an implication $cond(c) \rightarrow cons(c)$, where $cond(c)$ forms the condition and $cons(c)$ the consequent part.
2. For all $c \in SRS \cup KB$ $cond(c)$ only formulates restrictions on variables in X_U , i.e. $vars(cond(c)) \subseteq X_U$.
3. Constraint c is applicable ($app(c)$), *iff* its condition part ($cond(c)$) evaluates to true, i.e. $T \cup cond(c)$ is satisfiable.
4. Constraint c is satisfied ($sat(c)$), *iff* it is either not applicable or its consequent part ($cons(c)$) evaluates to true, i.e. $\forall \theta_i \in \Theta \neg(app(c))$ or $((\theta_i \models T) \rightarrow (\theta_i \models T \cup cons(c)))$.
5. In case an assignment tuple θ_i satisfies all constraints, i.e. $\theta_i \models T$, then item i should be assigned the highest possible recommendation score.
6. However, if the given model θ_i does not satisfy any applicable constraints, i.e. $\forall c \in T_{max} \neg app(c)$, then item i should be assigned the lowest possible recommendation score.
7. In cases other than (5) and (6) where $\theta_i \models T_{max}$, the recommendation score should represent the relative share of weights of applicable and satisfied constraints compared to the cumulated weights of the applicable constraints. For instance a linear combination would score item i on the interval $[0 \dots 1]$ as follows:

$$score(i, T) = \left[\sum_{\substack{c \in T_{max} \\ \wedge app(c)}} w(c) \right]^{-1} \times \sum_{\substack{c \in T \wedge \\ app(c) \wedge sat(c)}} w(c)$$

Premise (1) conforms to the representation of association rules in general that are popular for personalization. Furthermore, any constraint c can obviously be reformulated as $TRUE \rightarrow c$. Furthermore, the applicability of a constraint can be determined independently from the model θ_i of an item i due to premise (2). The rationale for (5)-(7) is that it is sensible only to use applicable constraints for scoring an item because for instance the entailment of a constraint $a = 1 \rightarrow b = 2$ by a model $\theta = \{a = 2 \wedge b = 3\}$ is trivial. Thus, a $score(i, T) = 0.8$ can be interpreted as *item i satisfies 80% of all constraint weights of constraints in T that are applicable*.

In most practical situations the consequent part of a constraint c fulfills the condition $vars(cons(c)) \subseteq X_I$. For instance, the consequent part of the example constraint on the minimal focal length of a camera depicted in Table 3 needs to be evaluated for each item in the catalog only once and the binary result ($cons(c)$ is either satisfied by the item or not) can be used in all subsequent recommendation requests. Thus, checking if a $\theta_i \in \Theta$ is a model of any constraint whose consequent part restricts only on variables in X_I can moreover be decided independent of each other. Therefore, Theorem 3 states that for all constraints from the subset $T_S \subseteq T$, that all restrict only variables de-

scribing item features, their satisfiability with respect to a specific assignment tuple can be checked separately from each other.

Theorem 3 *Let MAX MCRcTask be represented by Theory T , $T_{max} \subseteq T$, Model θ_i and Theory $T_S \subseteq T$ with $T_S = \{c_1, \dots, c_n\}$ and $\bigwedge_k^{1, \dots, n} app(c_k) \wedge vars(cons(c_k)) \subseteq X_I$. Then $\theta_i \models T$ iff for all $c \in T_S$ $\theta_i \models \{c\}$ and $\theta_i \models T \setminus T_S$. Furthermore, for all $c \in T_S$ $\theta_i \models T_{max} \wedge c \in T_{max}$ iff $\theta_i \models \{c\}$ and $\theta_i \models T_{max} \setminus T_S$.*

Proof Sketch. Let $T_1 \subseteq T$, $T_2 \subseteq T_S$, $T_1 \cup T_2$ be consistent by definition and $\theta_i \models T_1 \cup T_2$. It follows $\theta_i \models T_1$ and $\theta_i \models T_2$ due to the monotonicity of logic. Since $\theta_i \models T_1$, $\theta_i \models T_2$ and θ_i fully specifies exactly one product item, it follows that $\theta_i \models T_1 \cup T_2$

In practice, we query a database table to check if a specific constraint $c \in T$ is entailed by a specific item from the catalog. Therefore, we introduce a relational database table I_T that contains all item tuples from Θ , i.e. $I_T = \{\langle x_1 : v_{1,1}, \dots, x_n : v_{1,n} \rangle, \langle x_1 : v_{2,1}, \dots, x_n : v_{2,n} \rangle, \dots, \langle x_1 : v_{p,1}, \dots, x_n : v_{p,n} \rangle\}$ with $|X_I| = n$ and $|\Theta| = p$. Furthermore, we use the projection π and the selection operator σ from relational algebra to query the catalog I_T . For instance, the projection of the table I_T to the id-attribute returns the ids for all p item instances $\pi_{[id]}(I_T) = \{i_1, \dots, i_p\}$.

Algorithm 1 sketches the function TopkMCRcTask that computes the k most highly recommendable items according to a given theory T . In lines 4-9 a binary constraint/item matrix is precomputed that stores the satisfiability for constraint/item pairs in memory by executing a query for every $c \in T$ comparable to the algorithm in Jannach [14]. Note, that we only need to query the database once for each constraint whose consequent part restricts solely on variables describing the item catalog at application startup, i.e. $vars(cons(c)) \subseteq X_I$. This task that can be done in polynomial time proportional to the size of the product table [40]. In lines 10-18 the applicability of each constraint is determined and their sum of weights is computed. Lines 20-28 sum up for each item the weights of all applicable constraints that are satisfied. Note, that in case the consequent part contains also variables from X_U (e.g. compare to the first and third row in Table 3) one additional query to the database is required where these variable symbols are replaced by their assigned values (see line 25). Every item from the catalog is scored according to premise (7) (see line 31) and finally a ranked list of items is returned.

It remains to be stated that partitioning T into a set of hard constraints T_{hard} that must always be fulfilled and a set of the remaining constraints $T \setminus T_{hard}$ that are soft is straightforward and that the definitions can be easily extended, i.e. $\theta_i \models T_{max} \Leftrightarrow \theta_i \models (T_{hard} \cup T_{max} \setminus T_{hard})$.

Input: Catalog I_T , Theory T , Top k parameter
Output: List of (item, score) tuples

```

1  $List = \text{emptyset}$ ;
2  $max = 0$ ;
3  $count = 0$ ;
4 %precompute matrix, skip in case of multiple invocation
5 foreach  $c \in T$   $\text{vars}(cons(c)) \subseteq X_I$  do
6   | foreach  $i \in \pi_{[id]}(\sigma_{[cons(c)]}I_T)$  do
7   |   |  $matrix(c, i) = \text{true}$ ;
8   | end
9 end
10 %consider only applicable constraints
11 foreach  $c \in T$  do
12   | if  $cond(c) \cup T$  is satisfiable then
13   |   |  $app(c) = \text{true}$ ;
14   |   |  $max = max + w(c)$ ;
15   | else
16   |   |  $app(c) = \text{false}$ ;
17   | end
18 end
19 %compute score for each item
20 foreach  $i \in \pi_{[id]}(I_T)$  do
21   | foreach  $c \in T$   $app(c)$  do
22   |   | if  $\text{vars}(cons(c)) \subseteq X_I \wedge matrix(c, i)$  then
23   |   |   |  $count = count + w(c)$ ;
24   |   | end
25   |   | else if  $\neg(\text{vars}(cons(c)) \subseteq X_I) \wedge i \in \pi_{[id]}(\sigma_{[cons(c)]}I_T)$  then
26   |   |   |  $count = count + w(c)$ ;
27   |   | end
28   | end
29   |  $List \leftarrow List \cup \{(i, \frac{count(c)}{max(c)})\}$ ;
30 end
31  $List \leftarrow \text{sort}(List)$ ;
32 Return top k of List;

```

Algorithm 1: TopkMCRecTask(I_T, T, k):List

The problem of top-k retrieval in databases can be considered related to the approach presented here [41]. In top-k retrieval the most similar tuples according to a relational query need to be retrieved and usually metric distance functions are applied while in constraint-based recommendation the applicability and satisfiability of weighted soft constraints is exploited for computing a ranked list of items. However, the use of histograms for data analysis or multi-attribute indexes as presented in [41] could be applied for algorithmic extension and tuning of the work presented herein.

5 Implementation

The system is part of a generic *ISeller* recommendation framework presented in [42, 43]. The constraint-based recommender resides on top of the core framework and utilizes a *Data Integration Service (DD)* for storing the item catalog as well as the *User Modeling Service (UM)* to evaluate the set of *SRS* deriving from a given user. Variables can be *n-ary* or *unary* depending on whether they can hold multiple values or only a single value. Each variable refers either to a feature of items managed by the *DD* (path expression starts with "DD." - see Figure 2) or references a characteristic of a user model managed by the (path expression starts with "UM."). Thus, when the recommendation component is requested to compute recommendations for an instantiated user model at runtime the recommendation scores for the top-ranked items in the product catalog are provided to client applications through the interfaces supplied by the core *Recommendation Service*.

Similar to other modules of *ISeller*, the constraint-based recommendation module also provides several graphical modeling tools which assist knowledge engineers in modeling the constraint-base and supports them in building valid constraints. Following the problem definition in Section 4 it is assumed that only syntactically valid constraints are defined. They can be marked either as hard or soft, where the latter are assigned a weight signifying the cost associated with not fulfilling them. A simplified EBNF grammar of the constraint language is sketched in Table 4. Note that in contrast to the actual system, the grammar presented in Table 4 lacks parentheses, semantic checks on datatypes and terminal symbols for variables, sets and constants for reasons of simplicity.

For knowledge acquisition by domain experts the system offers a context-sensitive editing support and inline syntax checking to prevent the creation of invalid constraints. In addition, the environment enables knowledge engineers to simulate recommendation results for historical user sessions in order to verify and debug constraints in the knowledge base. For instance, the left side of Figure 3 depicts a portion of an historical user session that contains

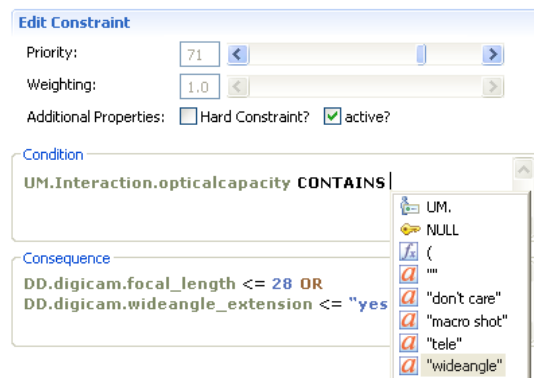
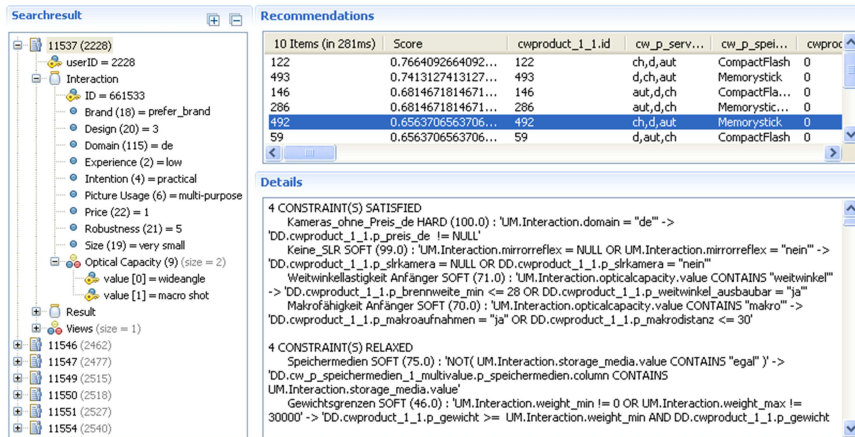


Fig. 2 Constraint Editor

Table 4 EBNF of constraint language.

NR	Projection
1	Constraint := LogicalExpression \mapsto LogicalExpression
2	LogicalExpression := [NegationOperator] Expression {LogicalOperator LogicalExpression}*
3	Expression := UnaryAtomicExpression UnaryOperator UnaryAtomicExpression N-aryAtomicExpression N-aryOperator N-aryAtomicExpression
4	UnaryAtomicExpression := UnaryVariable Constant
5	N-aryAtomicExpression := N-aryVariable Constant
6	NegationOperator := \neg
7	LogicalOperator := \vee \wedge
8	UnaryOperator := = \neq <i>LIKE</i> < > <= >= + - \times \div
9	N-aryOperator := <i>CONTAINS</i>
10	UnaryVariable := NumberVar BooleanVar StringVar
11	N-aryVariable := NumberSet BooleanSet StringSet
12	Constant := NumberConstant BooleanConstant StringConstant

**Fig. 3** User interface components for knowledge base validation

the explicit requirements observed for a user with *userID* = 2228. A list of recommendations (see upper right corner of Figure 3) is produced for the selected user model and a detailed summary on satisfaction or relaxation of constraints is provided for each item (see lower right corner of Figure 3).

6 Evaluation

The goal of the empirical work was to compare different variants of constraint-based recommendation algorithms in terms of standard accuracy and coverage metrics. The evaluation is performed on historical user sessions like the one sketched in Table 1, where a set of specific requirements *SRS* and the codes/ids of actually purchased items are known. Furthermore, domain knowledge *KB* and a table of fully specified item *I_T* is available.

We compared different non-interactive constraint-based recommendation mechanisms. They take for each historical user session the same *SRS*, *KB*

and I_T as input and their output is compared against the codes of actually bought items in the respective historical user session.

- *No relax* reasoning considers all constraints to be hard constraints and is thus the baseline strategy.
- *MAX CSPRecTask* determines maximally succeeding subqueries (XSS) like implemented in [3].
- *TopkMCRecTask* ranks items according to their satisfiability with (subsets of) the constraint base and returns the top k of them.

The goal of the evaluation is to research the following hypothesis:

The presented TopkMCRecTask algorithm shows improved accuracy results measured in terms of Precision and Recall when compared with a recommendation strategy that identifies a maximal set of succeeding subqueries and does not exploit preference weights for ranking the returned items.

This section is structured into three parts: the first one describes the datasets and how they have been extracted, Subsection 6.2 describes the research methodology and finally in Subsection 6.3 results are presented and discussed.

6.1 Datasets

The evaluation is based on two different datasets of extracted log data from two conversational RS applications and the weblogs of the corresponding e-commerce sites. The two original RS were implemented using the *CWAdvisor Suite* framework [2,3] in the domains of *digital cameras* and *premium cigar blends*. Table 5 summarizes their characteristics¹.

	Digital Cameras (<i>DC</i>)	Cigars (<i>PC</i>)
Product items	>400	143
Users sessions	750	535
Positive unary ratings	819	1,422
Knowledge base		
Variables	88	42
Hard constraints	3	2
Soft constraints	46	44

Table 5 Characteristics of dataset and knowledge base

The digital camera dataset (*DC*) was recorded over a period of 3 months (Dec/03 - Feb/04) and includes 750 different user sessions possessing explicit user requirements and purchase information for items. The knowledge base *KB* consists of 49 constraints out of which only three are hard. On average 2.8 distinct variables and 3.65 *Logical Expressions* that are combined by logical

¹ Note, that the datasets are available for download from http://isl.ifit.uni-klu.ac.at/datasets_constraints.zip.

operators (see Table 4) appear in each constraint. The constraint model maps the users' requirements with respect to optical characteristics, resolution or other features such as memory cards to the properties encoded in the catalog of items. The user model of a session consists of 31 different variables (X_U) and items are characterized by 57 features (X_I). 750 conversational interactions that showed interest (i.e. positive implicit rating) on a share of 122 items from the overall product catalog were used for experimentation.

The second dataset stems from the domain of premium cigars (PC) and was logged between Oct/05 to May/09. We identified 535 distinct sessions in which users disclosed their preferences in an online dialogue and implicitly rated at least one item by an add to basket action or by accessing the item's details page at least twice. The knowledge base KB consists of 46 implication constraints that are formulated on 3 distinct variables and consist of 5.8 *Logical Expressions* on average. User requirements and items are described by $|X_U| = 19$ and $|X_I| = 23$ variables respectively. The constraints ensure for instance that inexperienced users do not receive recommendations for cigars with straining effects or long smoking duration and that pricing restrictions of the user are observed. A more detailed description of the knowledge base has been presented in [39]. Both knowledge bases have been acquired with the help of domain experts and are in productive use since several years.

6.2 Methodology

An experimental research design, where the research subjects are the historical user sessions in log data, is widespread for the evaluation of recommendation systems [44]. The basic idea is to have a collection of user profiles u containing preference information like unary constraints on variables in X_U representing the specific requirements SRS (see variables in Table 1) in this setup and the codes/ids of actually purchased items that are withheld for the $testset_u$ (see codes of rated items in Table 1). Algorithms then exploit the specific requirements for the given user profile SRS_u , the knowledge base KB and the item catalog I_T in order to make predictions that can be compared against the hidden $testset_u$. The obvious advantage of this approach is that it allows the performance of different algorithms to be compared against each other on real-world data that was collected on commercial platforms.

Predictions of the algorithm are denoted by $recset_u$, where recommendation size k is varied between 3, 5 and 10 for all algorithms and all user sessions. Successful predictions are termed *hits* and are defined as follows:

$$hits_u = recset_u \cap testset_u$$

Precision and *Recall* are the two best-known metrics for measuring classification tasks like recommendation, but they are also used for measuring the quality of information retrieval tasks in general [45,44]. Both are computed as fractions of $hits_u$, the number of correctly recommended relevant items for

user profile u . The Precision metric (P) relates the number of hits to the total number of recommended items ($|recset_u|$).

$$P_u = \frac{|hits_u|}{|recset_u|}$$

In contrast the Recall (R) computes the ratio of hits to the theoretical maximum number of hits due to the testing set size ($|testset_u|$).

$$R_u = \frac{|hits_u|}{|testset_u|}$$

By increasing the size of a recommendation set the tradeoff between Precision and Recall metric can be observed. Recall will typically improve as the chance of hitting more elements from the testset increases with recommendation set size at the expense of lower Precision.

Consequently, the F1 metric is used in order to produce evaluation results that are more universally comparable:

$$F1 = \frac{2 \cdot P_u \cdot R_u}{P_u + R_u}$$

The F1 metric effectively averages Precision and Recall with bias towards the weaker value. In addition, we also report the proportion of users for which at least one item from user's testset is recommended. I.e. hit rate is defined as:

$$hitrate_u = \begin{cases} 1 & : \text{if } hits_u > 0 \\ 0 & : \text{else} \end{cases}$$

Rank scores extend the results of classification metrics with a finer level of granularity. They differentiate between successful hits by also taking their relative position in recommendation lists into account. Breese et al. [46] propose a metric that assumes decreasing utilities based on items' rank. The parameter α sets the half-life of utilities, which means that a successful hit at the first position of the recommendation list has twice as much utility to the user than at the $\alpha + 1$ rank. The rationale behind this weighting is that later positions have a higher chance of being overlooked by the user although they might be useful recommendations.

$$RS_u = \sum_{i \in hits_u} \frac{1}{2^{\frac{rank(i)-1}{\alpha}}}$$

$$RS_u^{max} = \sum_{i \in testset_u} \frac{1}{2^{\frac{id_x(i)-1}{\alpha}}}$$

$$RS'_u = \frac{RS_u}{RS_u^{max}}$$

The function $rank(i)$ returns the position of item i in the user's recommendation list. RS_u^{max} is required for normalization and returns the maximum

achievable score if all the items in the user’s $testset_u$ were assigned to the lowest possible ranks, i.e. ranked according to a bijective index function $idx()$ assigning values $1, \dots, |testset_u|$ to the testset items.

One metric that allows evaluators to compare different techniques based on their capability to compute recommendations for a large share of the population is user coverage ($Ucov$) [44].

$$Ucov = \frac{\sum_{u \in U} \rho_u}{|U|}$$

$$\rho_u = \begin{cases} 1 & : \text{if } |recset_u| > 0 \\ 0 & : \text{else} \end{cases}$$

It measures the share of users from U to whom non-empty recommendation lists can be provided. Obviously, it is only sensible to measure user coverage in conjunction with an accuracy metric as otherwise recommending arbitrary items to all users would be considered as an acceptable strategy.

6.3 Results and discussion

Experiments were repeated with three different sizes of the recommendation set (k) for each dataset and preference reasoning strategy. Table 6 and Table 7 list the results of the experiments in digital cameras and cigars domain respectively. Probably the most important result is that the *No relax* strat-

Strategy	k	Avg. k	Hitrate	RS	R	P	F1
No relax	3	2.91	3%	22.99%	23.80%	9.12%	13.09%
No relax	5	4.65	4%	29.14%	31.43%	8.44%	13.08%
No relax	10	8.46	6%	37.57%	44.76%	8.39%	13.44%
MAX CSPRecTask	3	2.99	5%	4.02%	4.16%	1.60%	2.29%
MAX CSPRecTask	5	4.95	6%	4.91%	5.26%	1.42%	2.18%
MAX CSPRecTask	10	9.78	8%	6.13%	7.33%	1.35%	2.15%
Top3MCRRecTask	3	3	20%	18.04%	18.70%	6.80%	9.99%
Top5MCRRecTask	5	5	31%	26.03%	28.61%	6.24%	10.24%
Top10MCRRecTask	10	10	49%	36.36%	44.50%	4.85%	8.74%

Table 6 Experiment Results *DC* domain

egy that interpreted all constraints to be hard reached a User Coverage of $Ucov = 12.67\%$ for *DC* and $Ucov = 72.52\%$ for *PC*, which means that non-empty result sets could only be produced for 95 out of 750 and 388 out of 535 historic user sessions, while the two other strategies that relaxed not satisfiable constraints maintained 100% User Coverage for both datasets. Thus the reported metrics are always relative to the set of users that received recommendations, meaning for instance that the reported Recall value 4.16% of

Strategy	k	Avg. k	Hitrate	RS	R	P	F1
No relax	3	2.58	14%	10.24%	7.74%	8.38%	7.97%
No relax	5	3.90	19%	13.39%	10.37%	8.22%	8.63%
No relax	10	6.45	28%	17.38%	15.05%	8.00%	9.06%
MAX CSPRecTask	3	2.58	19%	9.73%	7.10%	8.04%	7.58%
MAX CSPRecTask	5	3.86	27%	13.18%	10.13%	8.02%	8.50%
MAX CSPRecTask	10	6.22	38%	17.16%	14.42%	7.86%	8.85%
Top3MCRRecTask	3	3	24%	15.52%	9.00%	7.97%	8.34%
Top5MCRRecTask	5	5	36%	19.61%	13.50%	7.18%	9.11%
Top10MCRRecTask	10	10	57%	27.63%	21.52%	5.72%	8.59%

Table 7 Experiment Results *PC* domain

MAXCSPRecTask ($k = 3$) in Table 6 corresponds to 34 correctly predicted items while $R = 23.8\%$ of *no relax* ($k = 3$) is only equivalent to 25 hits in total.

In case of the *No relax* strategy the precision values, i.e. the relative share of relevant items, of computed recommendation lists are highest for both datasets. Moreover, precision consistently deteriorates as the size of the recommendation set n increases for all methods. However, when comparing the hitrate that gives the share of users from the total population that were recommended at least one successful hit the accuracy improvement of the *TopkMCRRecTask* approach in comparison to a *MAXCSP* strategy is clearly shown. Although a hitrate between 20% and over 50% for 10 recommended items is far away from 100% one has to see the real-world context. Only the initial specific requirements of users are known, so producing hits after the first round of interaction is already a success. The improvement of *TopkMCRRecTask* on the *DC* dataset is particularly large, because there the cardinality of the maximal succeeding subsets was very low leading to large candidate sets for recommendation.

In summary, the *TopkMCRRecTask* and its ranking scheme that exploited the constraints' weights clearly outperforms an approach that computes a *MAXCSP* [3] and ignores the preference weights for ranking. As the presented constraint-based recommendation approach is capable of computing a score for a given user and an item instead of returning undifferentiated sets of items it can be more easily integrated and hybridized with other recommendation paradigms like collaborative filtering (CF) [18]. Research into that direction will also be of particular interest to preference reasoning and modeling in general as statistical learning methods like CF can estimate predictions on fine granular preferences and weights that cannot always be explicitly modeled by constraints due to existing bounds on size and user effort.

7 Conclusions

This article discussed the constraint-based recommendation problem in the context of different usage scenarios. It formalized the constraint-based recommendation task both as a MAX CSP and as a Model Checking Problem and showed that the latter conforms more naturally to the task. It presented an algorithm that ranks the recommended items according to their degree of constraint fulfillment and compared it to one computing a maximal succeeding subquery or a *MAXCSP* [3] to avoid an empty result set. The evaluation on two different datasets of historical user profiles provided clear empirical evidence that the proposed strategy with a ranking based on constraint weights is clearly superior to a *MAXCSP* strategy without ranking for constraint-based recommendation tasks in terms of standard classification metrics.

References

1. Burke, R.: Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction* **12**(4) (2002) 331–370
2. Jannach, D., Kreutler, G.: A Knowledge-Based Framework for the Rapid Development of Conversational Recommenders. In: *5th International Conference on Web Information Systems Engineering (WISE)*, Brisbane, Australia, Springer (2004) 390–402
3. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: An Integrated Environment for the Development of Knowledge-Based Recommender Applications. *International Journal of Electronic Commerce* **11**(2) (2006) 11–34
4. Felfernig, A., Burke, R.: Constraint-based Recommender Systems: Technologies and Research Issues. In: *10th International Conference on Electronic commerce (ICEC)*, New York, NY, USA, ACM (2008) 1–10
5. Felix, D., Niederberger, C., Steiger, P., Stolze, M.: Feature-oriented vs. needs-oriented product access for non-expert-online shoppers. In: *The First IFIP Conference on E-Commerce, E-Business, E-Government (I3E)*. (2001) 399–406
6. Spiekermann, S., Paraschiv, C.: Motivating human-agent interaction: Transferring insights from behavioral marketing to interface design. *Electronic Commerce Research* **2**(3) (2002) 255–285
7. Linden, G., Hanks, S., Lesh, N.: Interactive Assessment of User Preference Models: The Automated Travel Assistant. In: *6th International Conference on User Modeling (UM)*, Lyon, France (1997) 67–78
8. Burke, R.: Interactive Critiquing for Catalog Navigation in E-Commerce. *Artificial Intelligence Review* **18**(3-4) (2002) 245–267
9. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Dynamic critiquing. In: *7th European Conference on Advances in Case-Based Reasoning (ECCBR)*, Madrid, Spain (2004) 763–777
10. Pu, P., Faltings, B.: Decision tradeoff using example-critiquing and constraint programming. *Constraints* **9**(4) (2004) 289–310
11. Shimazu, H.: Expertclerk: A conversational case-based reasoning tool for developing salesclerk agents in e-commerce webshops. *Artificial Intelligence Review* **18**(3-4) (2002) 223–244
12. Freuder, E.C., Wallace, R.J.: Partial constraint satisfaction. *Artificial Intelligence* **58**(1-3) (1992) 21–70
13. McSherry, D.: Retrieval Failure and Recovery in Recommender Systems. *Artificial Intelligence Review* **24**(3-4) (2005) 319–338
14. Jannach, D.: Techniques for Fast Query Relaxation in Content-Based Recommender Systems. In: *3rd IEEE Conference On Intelligent Systems (IS)*, IEEE Press (2006) 355–360

15. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: Developing Constraint-based Recommenders. In: *Recommender Systems Handbook*. Springer (2010)
16. Mirzadeh, N., Ricci, F., Bansal, M.: Supporting User Query Relaxation in a Recommender System. In: *5th International Conference on E-Commerce and Web Technologies (EC-Web)*, Zaragoza, Spain, Springer (2004) 31–40
17. Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., Teppan, E.: Plausible repairs for inconsistent requirements. In: *21st International Joint Conference on Artificial Intelligence*, Pasadena, CA, USA (2009) 791–796
18. Adomavicius, G., Tuzhilin, A.: 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) (2005) 734–749
19. Burke, R.: The Wasabi Personal Shopper: A Case-Based Recommender System. In: *11th Conference on Innovative Applications of Artificial Intelligence (IAAI)*, Trento, IT, AAAI (2000) 844–849
20. Torrens, M., Faltings, B., Pu, P.: Smartclients: Constraint satisfaction as a paradigm for scalable intelligent information systems. *Constraints* **7**(1) (2002) 49–69
21. Viappiani, P., Faltings, B., Pu, P.: Evaluating preference-based search tools: A tale of two approaches. In: *21st National Conference on Artificial Intelligence (AAAI)*, Boston, Massachusetts, USA (2006) 205–210
22. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Incremental critiquing. *Knowledge-Based Systems* **18** (2005) 143–151
23. Reilly, J., Zhang, J., McGinty, L., Pu, P., Smyth, B.: Evaluating compound critiquing recommenders: a real-user study. In: *Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*, New York, NY, USA, ACM (2007) 114–123
24. Viappiani, P., Faltings, B., Pu, P.: Preference-based search using example-critiquing with suggestions. *Artificial Intelligence Research* **27** (2006) 465–503
25. Teppan, E., Felfernig, A.: Asymmetric dominance- and compromise effects in the financial services domain. In: *IEEE Conference on Commerce and Enterprise Computing (CEC)*, Vienna, Austria (2009) 57–64
26. Rossi, F., Sperduti, A.: Acquiring both constraint and solution preferences in interactive constraint systems. *Constraints* **9**(4) (2004) 311–332
27. Jannach, D., Kreutler, G.: Rapid Development of Knowledge-Based Conversational Recommender Applications with Advisor Suite. *Journal of Web Engineering* **2** (2007) 165–192
28. Jannach, D.: Knowledge-based System Development with Scripting Technology: A Recommender System Example. In: *20th International Conference on Software Engineering & Knowledge Engineering (SEKE)*, San Francisco, CA, USA (2008) 405–416
29. Zanker, M., Bricman, M., Jessenitschnig, M.: Cost-Efficient Development of Virtual Sales Assistants. In: *2nd International Symposium on Intelligent Interactive Multimedia Systems and Services (KES IIMSS)*, Springer (2009) 1–11
30. Tsang, E.: *Foundations of Constraint Satisfaction*. Academic Press Limited, UK (1993)
31. Jannach, D., Kreutler, G.: Personalized User Preference Elicitation for e-Services. In: *5th IEEE International Conference on e-Technology, e-Commerce and e-Services (EEE)*, Los Alamitos, CA, USA, IEEE Computer Society (2005) 604–611
32. Jannach, D., Zanker, M., Fuchs, M.: Constraint-based recommendation in tourism: A multi-perspective case study. *Information Technology & Tourism* **11**(2) (2009) 139–156
33. Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence* **152**(2) (2004) 213–234
34. Reiter, R.: A theory of diagnosis from first principles. *Artificial Intelligence* **32**(1) (1987) 57–95
35. Junker, U.: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In: *19th National Conference on Artificial Intelligence (AAAI)*, San Jose, CA, USA (2004) 167–172
36. Halpern, J.Y., Vardi, M.Y.: Model checking vs. theorem proving: A manifesto. In: *2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*. (1991) 325–334
37. Zanker, M., Jessenitschnig, M., Jannach, D., Gordea, S.: Comparing Recommendation Strategies in a Commercial Context. *IEEE Intelligent Systems* **22**(3) (2007) 69–73

-
38. Winterfeldt, D., Edwards, W.: *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge, England (1986)
 39. Zanker, M., Jessenitschnig, M.: Case-studies on exploiting explicit customer requirements in recommender systems. *User Modeling and User-Adapted Interaction* **19**(1-2) (2009) 133–166
 40. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: *14th Annual ACM Symposium on Theory of Computing (STOC)*, San Francisco, CA, USA, ACM (1982) 137–146
 41. Bruno, N., Chaudhuri, S., Gravano, L.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Database Syst.* **27**(2) (2002) 153–187
 42. Jessenitschnig, M., Zanker, M.: ISeller: A Flexible Personalization Infrastructure for e-Commerce Applications. In: *10th International Conference on Electronic Commerce and Web Technologies (EC-Web)*, Linz, Austria, Springer (2009) 336–347
 43. Jessenitschnig, M., Zanker, M.: A generic user modeling component for hybrid recommendation strategies. In: *11th IEEE Conference on Commerce and Enterprise Computing (CEC)*, Vienna, Austria, IEEE Press (2009) 337–344
 44. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems* **22**(1) (2004) 5–53
 45. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based Collaborative Filtering Recommendation Algorithms. In: *10th International World Wide Web Conference*, New York, NY, USA, ACM (2001) 285–295
 46. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: *14th Conference on Uncertainty in Artificial Intelligence (UAI)*, Madison, WI, USA, Morgan Kaufmann (1998) 43–52