Plausible Repairs for Inconsistent Requirements*

Alexander Felfernig[1], Gerhard Friedrich[2], Monika Schubert[1], Monika Mandl[1], Markus Mairitsch[2], and Erich Teppan[2]

[1]Applied Software Engineering, Graz University of Technology
email:{alexander.felfernig, monika.schubert, monika.mandl}@ist.tugraz.at

[2]Intelligent Systems and Business Informatics, University of Klagenfurt
email: {gerhard.friedrich, markus.mairitsch, erich.teppan}@uni-klu.ac.at

Abstract

Knowledge-based recommenders support users in the identification of interesting items from large and potentially complex assortments. In cases where no recommendation could be found for a given set of requirements, such systems propose explanations that indicate minimal sets of faulty requirements. Unfortunately, such explanations are not personalized and do not include repair proposals which triggers a low degree of satisfaction and frequent cancellations of recommendation sessions. In this paper we present a personalized repair approach that integrates the calculation of explanations with collaborative problem solving techniques. In order to demonstrate the applicability of our approach, we present the results of an empirical study that show significant improvements in the accuracy of predictions for interesting repairs.

1 Introduction

Knowledge-based recommenders [Burke, 2000; Felfernig *et al.*, 2007] support the effective identification of interesting items from large and complex assortments. Examples for such items are different types of financial services, computers, or digital cameras. In contrast to collaborative filtering [Konstan *et al.*, 1997] and content-based filtering approaches [Pazzani and Bilsus, 1997], knowledge-based recommenders rely on an explicit representation of item and advisory knowledge. Those systems exploit two different types of knowledge sources: on the one hand explicit knowledge about the given set of customer requirements (in this paper denoted as $R=\{r_1, r_2, \ldots, r_m\}$), on the other hand deep knowledge about the underlying items (denoted as $I=\{i_1, i_2, \ldots, i_n\}$). Recommendation knowledge is represented in the form of explicit constraints that relate requirements to the corresponding item properties. For simplicity, in this paper requirements will be directly tested on the item properties in the form of conjunctive queries. Interacting with a knowledge-based recommender application typically means to *answer a set of questions* (requirements elicitation phase), *repairing inconsistent requirements* (if no recommendation could be found), and *evaluating recommendations*. In this paper we focus on situations where no recommendation could be found. In order to deal with such situations, we propose an algorithm *for the automated and personalized repair of inconsistent requirements* that can improve the overall quality and acceptance of recommender interfaces.

Existing approaches to the handling of inconsistent requirements are focusing on low-cardinality *diagnoses* [Felfernig *et al.*, 2004; Jannach, 2006; Junker, 2004] respectively *explanations* [O'Sullivan *et al.*, 2007] (where diagnoses/explanations are computed in order-increasing cardinality [Reiter 1987]), or high-cardinality sets of *maximally successful sub-queries* [Godfrey, 1997; McSherry, 2004]. Such diagnoses resp. successful sub-queries indicate potential areas for changes but do not propose concrete repair actions. Furthermore, it definitely cannot be guaranteed that low-cardinality diagnoses lead to the most interesting repair actions for a user [O'Sullivan *et al.*, 2007]. In order to tackle this challenge, we propose an approach that integrates k-nearest neighbor algorithms typically applied in case-based [Burke, 2000] and collaborative recommendation [Konstan *et al.*, 1997] with the Hitting Set Directed Acyclic Graph (HSDAG) algorithm used for the calculation of diagnoses [Reiter, 1987]. Thus, our major contribution is to enhance model-based diagnosis with collaborative problem solving and thus to support the calculation of individualized repairs.

The remainder of this paper is organized as follows. In Section 2 we introduce a working example from the domain of financial services. In Section 3 we sketch the basic concepts behind non-personalized repair actions for inconsistent requirements. In Section 4 we introduce our algorithm for calculating personalized repairs. The results of evaluations are presented in Section 5. In Section 6 we provide an overview of related work. With Section 7 we conclude the paper.

*The work presented in this paper has been developed within the scope of the research projects WECARE, V-KNOW (both are funded by the Austrian Research Agency, FFG/FWF), and Softnet Austria that is funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

2. Example Domain: Financial Services

The following (simplified) assortment of financial services will serve as working example throughout the paper (see Table 1). The set of financial services $\{i_1, i_2, \ldots, i_8\}$ is stored in the item table I. Let us assume that our example customer has specified the following requirements that cannot be satisfied by the items of I: R = $\{r_1$:*return-rate* >= 5.5, r_2:*runtime* = 3.0, r_3:*accessibility* = yes, r_4:*bluechip* = yes$\}$. The feasibility of those requirements can simply be checked by a relational query $\sigma_{[R]}I$ where $\sigma_{[R]}$ represents the selection criteria of the query. For example, $\sigma_{[\text{return-rate} >= 5.5]}I$ would result in the financial services $\{i_6, i_7, i_8\}$.

Note that $\sigma_{[r1:\text{return-rate} >= 5.5,\ r2:\text{runtime} = 3.0,\ r3:\text{accessibility} = \text{yes},\ r4:\text{bluechip} = \text{yes}]}I = \emptyset$, i.e., no solution could be found for the set of requirements. In such situations customers ask for the recommendation of possible repairs which restore consistency between the requirements in R and the underlying item assortment I. State-of-the-art knowledge-based approaches [Felfernig *et al.*, 2004; Jannach, 2006; Felfernig *et al.*, 2007; O'Sullivan *et al.*, 2007] calculate *minimal sets* of faulty requirements which should be changed in order to find a solution. We show how to extend those approaches by introducing an algorithm for the calculation of *personalized repairs*.

id	return-rate (p.a.)	runtime	risk level	shares percentage	accessibility	plow back earnings	blue chip
i_1	4.2	3.0	A	0	no	yes	yes
i_2	4.7	3.5	B	10	yes	no	yes
i_3	4.8	3.5	A	10	yes	yes	yes
i_4	5.2	4.0	B	20	no	no	no
i_5	4.3	3.5	A	0	yes	yes	yes
i_6	5.6	5.0	C	30	no	yes	no
i_7	6.7	6.0	C	50	yes	no	no
i_8	7.9	7.0	C	50	no	no	no

Table 1: Example financial services I = $\{i_1, i_2, \ldots, i_8\}$

3. Calculating Non-Personalized Repairs

We exploit the concepts of Model-Based Diagnosis (MBD) [Reiter 1987; de Kleer *et al.*, 1992] which allows the automated identification of minimal sets of faulty requirements $r_i \in R$ [Felfernig *et al.*, 2004]. Model-based diagnosis starts with the description of a system which is in our case the predefined item assortment I. If the actual behaviour of the system conflicts with its intended behaviour (recommendation can be found), the diagnosis task is to determine those components (in our case the requirements in R) which, when assumed to be functioning abnormally, explain the discrepancy between the actual and the intended system behaviour. A diagnosis represents a minimal set of faulty components (in our case requirements) whose adaptation (repair) will allow the identification of a recommendation.

On a more technical level, minimal diagnoses for faulty requirements are identified as follows. Given I = $\{i_1, i_2, \ldots, i_n\}$ (item set) and a set R = $\{r_1, r_2, \ldots, r_m\}$ of requirements which is *inconsistent* with I, i.e., $\sigma_{[R]}I = \emptyset$. In such a situation, state-of-the-art recommenders [Felfernig *et al.*, 2004; Felfernig *et al.*, 2007] calculate a set of minimal diagnoses D = $\{d_1, d_2, \ldots, d_k\}$, where $\forall d_i \in D$: $\sigma_{[R - d_i]}I \neq \emptyset$, i.e., each d_i is a minimal set of requirements which have to be changed in order to make recommendations feasible. Minimality means that $\forall d_i \in D$: not $\exists\ d_i' \subset d_i$ s.t. $\sigma_{[R - d_i']}I \neq \emptyset$.

A corresponding *Customer Requirements Diagnosis Problem* (CRQ Diagnosis Problem) can be defined as follows:

Definition 1 (**CRQ Diagnosis Problem**): A CRQ Diagnosis Problem (Customer Requirements Diagnosis Problem) is defined as a tuple (R, I) where R=$\{r_1, r_2, \ldots, r_m\}$ is a set of requirements and I=$\{i_1, i_2, \ldots, i_n\}$ is an item assortment.

Based on this definition of a CRQ Diagnosis Problem, a CRQ Diagnosis (Customer Requirements Diagnosis) can be defined as follows:

Definition 2 (**CRQ Diagnosis**): A CRQ Diagnosis (Customer Requirements Diagnosis) for (R, I) is a set d=$\{r_1, r_2, \ldots, r_l\} \subseteq$ R, s.t. $\sigma_{[R - d]}I \neq \emptyset$.

Following the basic principles of Model-Based Diagnosis (MBD), the calculation of diagnoses $d_i \in D$ is based on the determination and resolution of conflict sets. A conflict set can be defined as follows (see, e.g., [Junker, 2004]):

Definition 3 (**Minimal Conflict Set CS**): A Conflict Set CS is defined as a subset $\{r_1, r_2, \ldots, r_q\} \subseteq$ R, s.t. $\sigma_{[CS]}I = \emptyset$. A conflict set CS is minimal iff there does not exist a conflict set CS' with CS' $\subset$ CS.

As already mentioned, the requirements $r_i \in R$ in our working example are inconsistent with the given assortment I of financial services, i.e., there does not exist one financial service in I that completely fulfills the requirements R = $\{r_1, r_2, r_3, r_4\}$. The conflict sets are CS_1:$\{r_1,r_2\}$, CS_2:$\{r_1,r_4\}$, CS_3:$\{r_2,r_3\}$ since $\sigma_{[CS1]}I = \emptyset$, $\sigma_{[CS2]}I = \emptyset$, and $\sigma_{[CS3]}I = \emptyset$. Furthermore, the identified conflict sets are minimal, since there do *not* exist $CS_1' \subset CS_1$, $CS_2' \subset CS_2$, $CS_3' \subset CS_3$ with $\sigma_{[CS1']}I = \emptyset$, $\sigma_{[CS2']}I = \emptyset$, and $\sigma_{[CS3']}I = \emptyset$.

Diagnoses $d_i \in D$ can be calculated by resolving conflicts in requirements. Due to its minimality property, one conflict can be easily resolved by deleting one of the elements from the conflict set. After having deleted at least one element from each of the conflict sets we are able to present a minimal diagnosis. Diagnoses derived from $\{CS_1, CS_2, CS_3\}$ are D = $\{d_1$:$\{r_1,r_2\}$, d_2:$\{r_1,r_3\}$, d_3:$\{r_2,r_4\}\}$. A discussion of the algorithm for calculating diagnoses is given in [Reiter, 1987; Friedrich and Shchekotykhin, 2006]. A personalized version of this algorithm will be presented in Section 4.

After having identified the set of possible minimal diagnoses (D), we have to propose repair actions for each of those diagnoses, i.e., we have to identify possible adaptations to the existing set of requirements such that the user is able to find a solution. The number of repair actions could potentially become very large which makes the identification of

acceptable repair actions a very tedious task for the user (see, e.g., [Felfernig *et al.*, 2007; O'Sullivan *et al.*, 2007]).

Alternative repair actions can be derived by querying the item table I with $\pi_{[attributes(d)]}(\sigma_{[R\text{-}d]}I)$. This query identifies all possible repair alternatives for a single diagnosis $d \in D$ where $\pi_{[attributes(d)]}$ is a projection and $\sigma_{[R\text{-}d]}I$ is a selection of tuples from I which satisfy the criteria in R-d. Executing this query for each of the identified diagnoses produces a complete set of possible repair alternatives. Table 2 depicts the complete set of repair alternatives REP = {rep_1, …, rep_5} for our working example where $\pi_{[attributes(d1)]}(\sigma_{[R\text{-}d1]}I) = \pi_{[return\text{-}rate, runtime]}(\sigma_{[r3:accessibility = yes,\ r4:bluechip = yes]}I)$ = {<return-rate=4.7, runtime=3.5>, <return-rate=4.8, runtime=3.5>, <return-rate=4.3, runtime=3.5>}. Furthermore, $\pi_{[attributes(d2)]}(\sigma_{[R\text{-}d2]}I) = \pi_{[return\text{-}rate, accessibility]}(\sigma_{[r2:runtime = 3.0,\ r4:bluechip = yes]}I)$ = {<return-rate=4.2, accessibility=no>} and $\pi_{[attributes(d3)]}(\sigma_{[R\text{-}d3]}I) = \pi_{[runtime,bluechip]}(\sigma_{[r1:return\text{-}rate>=5.5,\ r3:accessibility=yes]}I)$ = {<runtime=6.0, bluechip=no>}.

repair	return-rate	runtime	accessibility	bluechip
rep_1	4.7	3.5	√	√
rep_2	4.8	3.5	√	√
rep_3	4.3	3.5	√	√
rep_4	4.2	√	no	√
rep_5	√	6.0	√	no

Table 2: Repair alternatives for customer requirements

Note that in real-world scenarios (see, e.g., [Felfernig *et al.*, 2007]), the number of potential repairs could become very large and it is crucial to propose representative ones in order to avoid drawing false conclusions [O'Sullivan *et al.*, 2007].

4. Calculating Personalized Repairs

The above repair alternatives would be presented to the customer without taking into account the initially defined set of requirements. Important to be mentioned in this context is the fact that the repair alternatives in our example are at least partially ignoring the originally defined requirements and it is unclear which of those alternatives would be the most interesting for the customer.

In order to systematically reduce the number of repair candidates we need to integrate personalization concepts. Our approach is to identify those repair actions which resemble the original requirements of the customer. In order to derive such repair actions, we exploit the existing item definitions (see Table 1) for identifying alternatives which are near the originally defined requirements in R.[1]

From the item data in Table 1 we calculate the n-nearest neighbors (in our case, n = 5), i.e., those entries of the item table which are (most) similar to the given set of requirements in R. In our case, the determination of nearest neighbors is based on a simple similarity metric that calculates the individual similarities between the *m* given customer requirements in R (in our case m = 4) and the attribute settings in the item table. For the purpose of our example we use the following similarity function *sim*(R, i_j) where R represents a set of requirements and i_j is the j^{th} item in I.[2] Furthermore, *max*(k) denotes the maximum value in the domain of attribute *k*, and *w*(k) denotes the importance (weight) of attribute *k* for the customer – see Formula (1).

$$sim(R, i_j) = \sum_{k=1}^{m} \left(1 - \frac{|r_k - i_j[k]|}{\max(k) - \min(k)}\right) * w(k) \qquad (1)$$

The similarity between the customer requirements of our working example and item i_1 in the item table is calculated as follows: *sim*(R, i_1) = (1-|5.5-4.2|/3.7)*1/4 + (1-|3.0-3.0|/4)*1/4 + (1-|1-0|/1)*1/4 + (1-|1-1|/1)*1/4 = 0.66. Note that we assume w(1) = 1/4, w(2) = 1/4, w(3) = 1/4, and w(4) = 1/4 ($\Sigma w(j) = 1$).[3] Using Formula (1) we can determine the n-nearest neighbors which are those n items with the highest similarity to $r_i \in R$. The 5-nearest neighbors used in our example are shown in Table 3, the corresponding similarity values between R and $i_j \in I$ are shown in Table 4.

id	return-rate (p.a.)	runtime	risk level	shares percentage	accessibility	plow back earnings	blue chip
i_1	4.2	3.0	A	0	no	yes	yes
i_2	4.7	3.5	B	10	yes	no	yes
i_3	4.8	3.5	A	10	yes	yes	yes
i_5	4.3	3.5	A	0	yes	yes	yes
i_7	6.7	6.0	C	50	yes	no	no

Table 3: Calculated nearest neighbors $\{i_1,i_2,i_3,i_5,i_7\}$

id	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
sim(R,p_i)	**.66**	**.91**	**.92**	.41	**.88**	.36	**.48**	.08

Table 4: Calculated *sim*(*R*,i_j) for $\{i_1, …, i_8\}$

On the basis of the entries in Table 4, we now show step-by-step how our algorithm (Algorithm 1) calculates a personalized repair. The requirements in R are inconsistent with the nearest neighbors NE = $\{i_1,i_2,i_3,i_5,i_7\}$ (i.e., $\sigma_{[R]}NE = \emptyset$). Consequently, our algorithm activates a conflict detection component which returns one conflict set per activation. We determine minimal conflicts using a QuickXPlain [Junker, 2004] type algorithm. For the requirements in R and the items in NE the following conflict sets are derived: CS_1:{r_1,r_2}, CS_2:{r_1,r_4}, CS_3:{r_2,r_3}. Note that the reduction of the considered items from I to NE could change the set of calculated conflicts – in our case the set remains the same.

Let us assume that CS_1:{r_1,r_2} is returned as the first conflict set. A Hitting Set Directed Acyclic Graph (HSDAG) [Reiter

[1] Alternatively, customer interaction logs (or cases) can be exploited for the personalized recommendation of repair actions.

[2] Note that different metrics (e.g., similarity, diversity, selection probability) could be applied in this context, for simplicity we only use the *nearer-is-better* (NIB) similarity. An overview of related metrics can be found, for example, in [Wilson and Martinez, 1997; McSherry 2003].

[3] Different approaches are possible for the determination of importance weights, for example, direct specification by the customer or preference learning [Biso et al., 2000].

1987] is now instantiated (see Figure 1) with two outgoing paths from the root: $[r_1]$ and $[r_2]$. These paths indicate which elements of the identified conflict set have been used to resolve the conflict. Since it is our goal to identify repairs similar to the original requirements, we have to analyze which repairs are possible after eliminating one element from a conflict set (requirement $r_i \in R$). For example, eliminating r_1 would allow repairs supported by $\{i_1,i_2,i_3,i_5\}$ since $(\sigma_{[\neg r1]}NE) = \{i_1,i_2,i_3,i_5\}$, furthermore, $(\sigma_{[\neg r2]}NE) = \{i_2,i_3,i_5,i_7\}$. We compute the average similarity value for the first k items (in our case k=3). The first three items in $\{i_2,i_3,i_5,i_7\}$ have a higher average similarity than those in $\{i_1,i_2,i_3,i_5\}$, therefore we decide to extend the path $[r_2]$ to $[r_2,r_1]$ and $[r_2,r_4]$.

The most promising paths (candidates) are leading to diagnoses accepting those items in NE that are most similar to the original set of requirements in R. Again, for each of the resulting paths ($[r_2,r_1]$ and $[r_2,r_4]$) we have to analyze which are the *supporting items*: $(\sigma_{[\neg r2,\neg r1]}NE) = \{i_2,i_3,i_5\}$ and $(\sigma_{[\neg r2,\neg r4]}NE) = \{i_7\}$. The idea is to follow a best-first regime which expands the most promising candidate paths. Following our best-first strategy, we decide to extend the path $[r_2,r_1]$. However, since $\sigma_{[R-\{r2,r1\}]}NE$, i.e., $\sigma_{[r3:accessibility=yes,\ r4:bluechip=yes]}NE$, results in a non-empty set, $\{r_1,r_2\}$ is already a diagnosis (d). This diagnosis directly leads to repairs that are most similar to the original set of customer requirements (the fringe of the HSDAG does not contain any candidate paths with better combinations of repair alternatives). The set of possible repair actions for d can be simply determined by executing the query $\pi_{[attributes(d)]}(\sigma_{[R-d]}NE) = \pi_{[return\text{-}rate,\ runtime]}(\sigma_{[accessibility=yes,\ bluechip=yes]}NE)$. Executing this query results in the set of repairs $\{$<return-rate=4.7, runtime=3.5>, <return-rate=4.8, runtime=3.5>, <return-rate=4.3, runtime=3.5>$\}$.

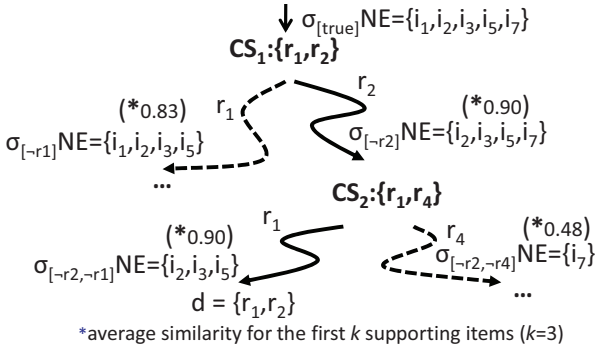

Figure 1: Personalized diagnosis d ∈ D calculated using the n-nearest neighbors. d = {r1,r2} is selected as basis for the derivation of personalized repairs

The algorithm for calculating personalized repair alternatives is the following (Algorithm 1 - CRQ Repairs). We keep the description of the algorithm on a level of detail which has been used in the description of the original HSDAG algorithm [Reiter, 1987]. In Algorithm 1, the different paths of the HSDAG are represented as separate elements in the bag structure *H* which is initiated with ∅. H stores all paths of the search tree in a best-first fashion, where the currently best path (*h*) is the one with the most promising potential repair alternatives (those repair alternatives most similar to the original requirements in R). If the theorem prover (TP) call (*TP*(*R-h*,*NE*)) does not detect any further conflicts for the elements in h (*isEmpty*(*CS*)), a diagnosis is returned and the corresponding repair alternatives can be calculated with $\pi_{[attributes(d)]}(\sigma_{[R-d]}NE)$. The major role of the theorem prover is to check whether there exists a recommendation for R minus the already resolved conflict set elements in h. If the TP call *TP*(*R-h*,*NE*) returns a non-empty conflict set *CS*, h is expanded to paths each containing exactly one element of CS (in this case no recommendation could be found). In the case that h is expanded, the original h must be deleted from H (*delete*(*h*,*H*)). Finally, if new elements have been inserted to H, it has to be sorted in order to determine the h with the most nearest repair candidates (*SimilaritySort*(*H,k*)).[4] This function calculates a set of *supporting items* for each $h_i \in H$ ($\sigma_{[\wedge(\neg rj)]}NE$, $r_j \in h_i$) and ranks each $h_i \in H$ conform to the highest item similarity (see Table 4) in its set of supporting items.[5]

Note that the function CRQ-Diagnosis in Algorithm 1 returns exactly one diagnosis d at a time (for which in the following potentially more than one repair is returned by CRQ-Repairs). Since NE ⊆ I, the detected conflict sets could differ for TP(R-h,NE) and TP(R-h,I) and some minimal diagnoses could be detected by CRQ-Diagnosis(R,NE,∅,k) which are not contained in CRQ-Diagnosis(R,I,∅,k)[6].

Algorithm 1 - CRQ-Repairs

```
/* R: set of customer requirements
   I: set of items
   n: number of nearest neighbours
   k: k most similar items to be used by SimilaritySort(H,k) */

CRQ-Repairs (R↓,I↓,n↓,k↓):Repair Set REP↑
(1)  { NE ← GetNearestNeighbors(R,I,n)
(2)    d ← CRQ-Diagnosis(R,NE, ∅,k)
(3)    return π[attributes(d)](σ[R-d]NE)  }

CRQ-Diagnosis(R↓,NE↓,H↓,k↓): Diagnosis h↑
(1)  {  h ← first(H)
(2)     CS ← TP(R-h,NE)
(3)     if (isEmpty(CS))
(4)        {return h}
(5)     else
(6)        {foreach X in CS:H ← H ∪ {h ∪ {X}}
(7)         H ← delete(h,H)
(8)         H ← SimilaritySort(H,k)
(9)         CRQ-Diagnosis(R,NE,H,k)}  }
```

[4] Note that the necessary HSDAG pruning is implemented by the functionalities of *SimilaritySort*(*H,k*).

[5] Many different quality measures are possible in this context. A simple one is the *highest average* of the *k* most similar items i_j (in our working example $k = 3$).

[6] In this case, Algorithm 1 takes into account all items in I.

5. Evaluation

Although our proposed repair functionalities do not significantly change the design of a recommender user interface, they clearly contribute to a more intelligent behavior of recommender applications and have the potential to trigger increased trust and satisfaction of users. The personalization approach as it is presented in this paper is definitely not restricted to conjunctive query based recommender applications but as well applicable with *constraint* [Junker, 2004] and *configuration technologies* [Felfernig *et al.*, 2004].

Performance. Algorithm 1 has been implemented on the basis of the standard hitting set algorithm proposed by [Reiter, 1987]. The algorithm is NP-hard in the general case [Friedrich *et al.*, 1991] but is applicable in interactive recommendation settings (see below). In our implementation, the calculation of minimal conflict sets is based on a QuickXPlain type algorithm [Junker, 2004]. QuickXPlain needs O(2k*log(n/k)+2k) consistency checks (worst case) to compute a minimal conflict set of size *k* out of *n* constraints in R. Consistency checks in our implementation are represented as conjunctive queries on I (Hypersonic SQL database). A performance evaluation (time effort depending on the number of calculated diagnoses) clearly shows the applicability of Algorithm 1 for interactive settings. On the basis of a test set of 5000 items with 10 associated properties we can observe the following calculation times for |D| diagnoses (see Table 5). These performance results confirm the results of our previous evaluations of financial service recommender applications.

|D|	time in secs (avg.)	std.dev.
1	0.22	0.08
2	0.35	0.14
3	0.36	0.10
4	0.54	0.10
5	0.59	0.11
6	0.67	0.13
7	0.79	0.18
8	0.92	0.25
9	1.00	0.20
10	1.70	0.31

Table 5: Performance of Algorithm 1

Empirical evaluation. In order to demonstrate the improvements achieved by our approach we conducted an empirical study. In this study we compared two different algorithms for the calculation of repairs. The first (Type 1) supported repairs based on the algorithm proposed in [Reiter, 1987], where diagnoses and corresponding repair alternatives are ranked according to their cardinality (standard breadth-first search). The second approach (Type 2) supported the calculation of personalized repair actions on the basis of Algorithm 1 (best-first search).

The used item assortment (90 items) has been selected from a dataset of an e-Commerce platform (digital cameras). In the scenario, the study subjects interacted with the recommender in order to identify a pocket-camera that best suits their needs. In the case of inconsistent requirements, subjects were confronted with a list of repair alternatives from which they had to select the most interesting one. N=493 subjects participated in the study. In order to systematically induce conflicts (no recommendation could be found), we filtered out items from the assortment that fulfilled requirements posed by the subject. On an average, 2.63 items (std.dev. 1.68) were filtered out per recommendation session. Subjects were informed about the fact that the item which had been specified was not available and they had to select a repair action from a proposed list of alternatives. With the remaining set a corresponding repair process was triggered. Thus each subject was confronted with a conflict situation where a repair alternative had to be selected. In order to keep the cognitive efforts realistic and acceptable, we set the upper limit of the number of repair actions to 5 in both recommender versions (Type 1 and Type 2).

Our hypothesis was that subjects will select higher ranked repair actions significantly more often if the Type 2 repair algorithm was used. For both recommender versions (Type 1 and Type 2) we measured in each session the normalized distance between the position of the selected repair action and position 1 (*position of selected repair* / *#repair alternatives*). On the basis of a two-sample t-test, a significant difference between the two repair approaches in terms of prediction quality can be observed. We detected significant higher deviation values (t-score=4.859, $p < 0.001$) for Type 1 recommenders. The average deviation for Type 1 recommenders was 0.599 (std.dev. 0.279), the value for Type 2 recommenders was significantly lower: 0.474 (std.dev. 0.274). Consequently, the above hypothesis can be confirmed. Furthermore, repair alternatives with the highest ranking (position) have been selected significantly more often in Type 2 recommenders ($\chi 2=17.746$, $p<0.001$). The precision (*#correctly predicted repair actions* / *#predicted repair actions*) of Type 2 recommenders was 0.551 whereas for Type 1 recommenders the precision was lower: 0.307. Correct prediction is interpreted as ranking a repair on position 1 that has been selected then.

	Type 1	Type 2	significance level
deviation	.599 (.279)	.474(.274)	p<0.001
precision	.307	.551	p<0.001

Table 6: Summarization of study results

6. Related Work

[Felfernig *et al.*, 2004] have developed concepts for the diagnosis of inconsistent customer requirements in the context of configuration problems. The idea was to apply the concepts of Model-based Diagnosis [Reiter, 1987] (MBD) in order to be able to determine minimal cardinality sets of

requirements which have to be changed in order to be able to find a solution – repairs were not supported in this context. In [O'Sullivan *et al.*, 2007] such minimal cardinality sets are called minimal exclusion sets, in [Godfrey, 1997; McSherry, 2004] the complement of such a set is denoted as maximally successful sub-query. The calculation of a diagnosis for inconsistent requirements relies on the existence of minimal conflict sets. Such conflict sets can be determined, for example, on the basis of QuickXPlain [Junker, 2004], a divide-and-conquer algorithm. The approach presented in [Felfernig *et al.*, 2004] follows the standard breadth-first search regime for the calculation of diagnoses [Reiter, 1987]. The major contribution of our paper in this context is the extension of this algorithm with collaborative problem solving concepts. This approach improves the prediction quality for repair alternatives and thus contributes to higher-quality recommender user interfaces. [O'Sullivan *et al.*, 2007] introduce the concept of representative explanations. Representative explanations follow the idea of generating diversity in alternative diagnoses – informally, constraints which occur in conflicts should as well somehow be included in diagnoses presented to the user. [Jannach, 2006] introduces the concept of preferred relaxations for conjunctive queries that help to select diagnoses on the basis of predefined utility functions. Compared to those previous approaches, the work presented in this paper first introduces an approach that exploits nearest neighbors concepts for the determination of personalized repair actions. Finally, case-based recommender systems [Burke, 2000] profit from the work presented in this paper since in addition to intelligent case retrieval, personalized and (if needed) minimal diagnoses and repairs can be calculated systematically.

7. Conclusions

In this paper we have introduced an algorithm that calculates personalized (plausible) repairs for inconsistent requirements. The algorithm integrates the concepts of model-based diagnosis (MBD) with the ideas of collaborative problem solving and thus significantly improves the quality of repairs in terms of prediction accuracy. We have evaluated our approach with a recommender application based on real-world items. The results of the study clearly demonstrate the improvements induced by personalized repair actions. Future work will include the evaluation of different alternative metrics (e.g., similarity, diversity, and selection probability) w.r.t. their impact on diagnosis/repair prediction accuracy.

References

[Biso *et al.*, 2000] A. Biso, F. Rossi, A. Sperduti. Experimental Results on Learning Soft Constraints, 7th Intl. Conf. on Knowledge Representation and Reasoning (KR 2000), Breckenridge, CO, USA, pages 435-444, 2000.

[Burke, 2000] R. Burke. Knowledge-based Recommender Systems. *Lib. & Inform. Systems*, 69(32):180-200, 2000.

[de Kleer *et al.*, 1992] J. de Kleer , A. Mackworth and R. Reiter. Characterizing diagnoses and systems, *AI Journal* 56(2-3):197-222, 1992.

[Felfernig *et al.*, 2004] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based Diagnosis of configuration knowledge bases, *AI Journal*, 152(2):213–234, 2004.

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

[Friedrich and Shchekotykhin, 2005] G. Friedrich and K. Shchekotykhin. A General Diagnosis Method for Ontologies. International Semantic Web Conference, LNCS 3729, pages 232-246, Galway, Ireland, 2005.

[Friedrich *et al.*, 1990] G. Friedrich, G. Gottlob, and W. Nejdl. Physical Impossibility Instead of Fault Models. AAAI/IAAI 1990, pages 331-336, Boston, Massachusetts, 1990.

[Godfrey, 1997] P. Godfrey. Minimization in cooperative response to failing database queries. *Intl. Journal of Cooperative Information Systems* 6(2):95–149, 1997.

[Konstan *et al.*, 1997] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news Full text. *Communications of the ACM*, 40(3):77-87, 1997.

[Jannach, 2006] D. Jannach. Finding Preferred Query Relaxations in Content-based Recommenders, IEEE Intelligent Systems Conf. (IS'2006), pages 355-360, 2006.

[Junker, 2004] U. Junker. QuickXPlain: Preferred Explanations and Relaxations for Over-Constrained Problems. AAAI'04, San Jose: AAAI Press, pages 167–172, 2004.

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

[McSherry, 2003] Similarity and Compromise. Intl. Conference on Case-based Reasoning (ICCBR'03), pages 291-305, Trondheim, Norway, 2003.

[O'Sullivan *et al.*, 2007] B. O'Sullivan, A. Papadopoulos, B. Faltings, P. Pu. Representative Explanations for Over-Constrained Problems. AAAI'07, pages 323-328, 2007.

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

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

[Wilson and Martinez, 1997] D. Wilson and T. Martinez. Improved Heterogeneous Distance Functions, *Journal of Artificial Intelligence Research*, 6:1-34, 1997.