Conflict-directed relaxation of constraints in content-based recommender systems

Dietmar Jannach and Johannes Liegl

Institute for Business Informatics & Application Systems
University Klagenfurt, A-9020 Klagenfurt, Austria
dietmar.jannach@ifit.uni-klu.ac.at, johannes.liegl@edu.uni-klu.ac.at

Abstract. Content-based recommenders are systems that exploit detailed knowledge about the items in the catalog for generating adequate product proposals. In that context, *query relaxation* is one of the basic approaches for dealing with situations, where none of the products in the catalogue *exactly* matches the customer requirements. The major challenges when applying query relaxation are that the relaxation should be minimal (or optimal for the customer), that there exists a potentially vast search space, and that we have to deal with hard time constraints in interactive recommender applications.
In this paper, we show how the task of finding adequate or customer optimal relaxations for a given recommendation problem can be efficiently achieved by applying techniques from the field of model-based diagnosis, i.e., with the help of extended algorithms for computing *conflicts* and *hitting sets*. In addition, we propose a best-effort search algorithm based on branch-and-bound for dealing with hard problems and also describe how an optimal relaxation can be immediately obtained when partial queries can be (pre-)evaluated.
Finally, we discuss the results of an evaluation of the described techniques, which we made by extending an existing knowledge-based recommender system and which we based on different real-world problem settings.

Introduction

Content-based recommender systems are a special class of recommendation systems that operate on the basis of explicit knowledge about customer requirements, product characteristics, and recommendation logic (typically some sort of "filter rules") that determines the set of products to be proposed, when given some specific customer requirements. One of the main problems of such content-based systems, however, is that situations can easily arise, where all of the existing products are filtered out and no adequate proposal can be made [1] as there exists no exact match. "*No products found*" is an undesirable system response in such situations, in which we would appreciate a more intelligent behaviour, i.e., an explanation of the situation or – even better – a list of products that fulfil as many as possible of the originally posted requirements. In that context, approaches based on Case-Based Reasoning (CBR) have the advantage that they are in principle capable of also retrieving products that are *similar* to the user's query. Nonetheless, also these similarity-based

approaches have their limitations, for example, that the most similar product might not be acceptable for the user, or that the means for explaining the proposal are limited [9].

Another approach of dealing with such situations is therefore to "relax" [10] the problem by giving up some of the requirements (i.e., remove parts of the query) and then test whether there exists a product that fulfils at least the remaining requirements. However, finding good relaxations is not a trivial problem, because typically many different alternative relaxations exist, the proposed relaxations should be minimal (or optimal for the user), and finally, the time-frame for finding such relaxations is strictly limited, because recommender systems are interactive applications. In this paper, we show how the task of finding adequate relaxations for a given recommendation problem can be efficiently achieved by applying techniques from the field of model-based diagnosis, i.e., with the help of extended algorithms for computing *conflicts* and *hitting sets*. After giving an introductory example, we develop a general and implementation-independent model of the recommendation problem and show how the relaxation problem can be mapped to a model-based diagnosis problem such that extended algorithms for conflict-identification and hitting set computation can be applied. In addition, we also propose a best-effort search algorithm based on branch-and-bound for dealing with hard problems and finally discuss experimental results which were achieved by extending an existing recommender system with these algorithms and by using different real-world test cases. The paper ends with a discussion of related and future work.

Example

In the following, we will give a simple example from the domain of digital cameras for illustrating the relaxation problem and the importance of conflict-directed search. Let us assume our product database (PDB) of digital cameras contains the following entries.

ID	USB	Firewire	Price	Resolution	Make
p1	true	false	400	5	Canon
p2	false	true	500	5	Canon
p3	true	false	200	4	Fuji
p4	false	true	400	5	HP

Our knowledge base of filter rules (FRS) comprises the following definitions.

f1: IF *the customer requires high-quality printouts of the pictures* THEN
recommend cameras with a resolution of 5 mega-pixels.
f2: IF *the customer wants to have a cheap camera* THEN
recommend cameras with a price smaller than 300.
f3: IF *customer needs a USB port* THEN
recommend cameras with a USB port.
f4: IF *customer wants extended connectivity* THEN
recommend cameras supporting Firewire.
f5: IN ANY CASE
recommend cameras with a resolution higher than 3 mega-pixels.

Now, let us assume that a customer has the following requirements (REQ):

USB-support, extended-connectivity, only cheap cameras, high-quality printouts.

Given these requirements, the product database and the filter-rules, no single product will fulfil all the requirements. We can now try to relax (retract) some of the filter rules, in order to find products that fulfil as many of the customer's constraints as possible. A simple algorithm for finding an adequate relaxation is to compute all possible combinations of filters (the powerset of FRS) and check for each of these combinations whether products remain when these filter rules are retracted. Of course, such an algorithm will – in the worst case – require 2^n checks, which is inadequate for realistic settings, where response times below one second are required.
We therefore propose adopting a conflict-directed approach similar to Reiter's [13] Hitting-Set algorithm: If we look closer on the problem situation of our example, we see that there are two "minimal conflicts", c_1 = {f1, f2} and c_2 = {f3, f4}, i.e., in any valid problem relaxation, at least one of the filters of c_1 and one of the filters of c_2 has to be removed. We also see that f5 is not involved in any conflict, which basically means that we do not have to take f5 into account when searching for relaxations. Furthermore, based on Reiter's general theory [13], we can conclude that computing the set of possible, minimal relaxations (i.e., *minimal diagnoses* in the sense of [13]) can be efficiently done by computing the *Hitting Set* of all minimal conflicts. In our example problem the set of minimal relaxations thus is the set:

{{f1, f3}, {f1, f4}, {f2, f3}, {f2, f4}}.

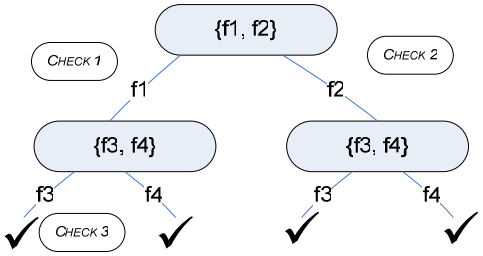

Figure 1 Hitting set search tree for example problem

Given these conflicts, finding the *first* valid relaxation {f1,f3} thus involves only the examination of three paths ({f1},{f2},{f1,f3}), i.e., three queries to the database, when constructing the search tree in breadth-first manner (Figure 1).

Conflict-directed filter relaxation

In the following, we develop a basic formalization of content- or filter-based[1] recommender problems by mapping the problem of finding adequate products in a catalog to a selection-query for a relational database, which has the advantage that we can rely on an existing, well-established formalism. In addition, knowledge-based

[1] We view "filter-based recommenders" [1] as a special class of content-based approaches.

recommenders or simple product finders in fact work by dynamically constructing database queries for filtering the products. However, please note that the chosen formalisation does not necessarily imply that a database system has to be used; it is rather used here to clearly characterize the problem.

Content-based recommender systems rely on the existence of a product catalog that contains detailed descriptions of the items to be recommended. In general, such a product catalog can be described as a relational database table as follows.

Definition (Product catalog): *Items in a product catalog are described by a set of attributes A; each attribute* $a_i \in A$ *is assigned a domain* d_i*. A* ***product catalog*** *P is then a subset of the Cartesian product of the domains* d_i*, i.e.* $P \subseteq d_1 \times \ldots \times d_n$*.*

The set of suitable products for some given customer requirements is determined by a set of *filter rules*, which were informally sketched as "if-then" rules in the example section.

Definition (Filter rule): *Let C be the set of attributes for describing customer requirements. A filter rule f can be described by the two characteristics f.AC, and f.FE, where f.AC is a Boolean formula (activation constraint[2]) over customer requirements C which describes the condition, when the filter rule shall be applied. f.FE represents the actual filter expression on the elements of the catalog (subquery) and is a Boolean formula over constant values, attributes of the product catalog and attributes from C.*

Note that in our definition we allow the usage of variables (from C) in the filter expression, e.g. for modelling dynamic filter rules like "*In any case, propose only products whose price is equal or lower to the price the customer has specified.*"
The recommendation problem consists of finding a set of items from the catalog that fulfils all the filter expressions of the active filter rules.

Definition (Recommendation problem): *A recommendation problem RP can be described by a tuple <P, FRS, C, CRS>: P is a product catalog, FRS is the set of filter rules, C the set of attributes for describing customer requirements, and CRS a function over the elements of C describing the actual customer requirements.*

The compound filter expression CFE for RP is a Boolean formula defined to be the conjunction of the filters whose activation condition is true, given some customer requirements CRS, i.e,

$$CFE = \bigwedge_{f \in FRS,\ f.AC = true} (f.FE).$$

Finding a ***solution to <P, FRS, C, CRS>*** *then corresponds to performing the database selection* σ_{CFE} *on the product catalog P.*

If none of the products in P satisfies all the compound expression CFE, we aim at finding a relaxation of the problem by retracting some of the filter rules, such that the selection results in a non-empty set of items from the catalog.

---

[2] The notion of "activation constraints" is inspired by the *Dynamic Constraint Satisfaction* approach, see e.g., [9]. We introduce that concept such that we can apply the approach also for "knowledge-based" recommenders, like, e.g., [7]

Definition (Valid relaxation): *Given a recommendation problem RP <P, FRS, C, CRS>, for which the size of the selection* $|\sigma_{CFE}(P)| = 0$*, a valid relaxation is a set RFRS* $\subseteq$ *FRS such that the solution for the modified recommendation problem RP' <P, FRS – RFRS, C, CRS> contains at least one element.*

Lemma: Given a recommendation problem *RP <P, FRS, C, CRS>* and $P \neq \emptyset$, a valid relaxation will always exist, because if we set *RFRS = FRS*, the selection query will be empty and all $p \in P$ are in the solution for *RP*.

In general, however, we are interested in finding optimal or minimal relaxations, e.g., we should try to find products that fulfil as many of the customer requirements as possible.

Definition (Minimal relaxation): *A relaxation RFRS for RP is said to be minimal if there exists no RFRS'* $\subset$ *RFRS such that RFRS' is a valid relaxation for RP.*

Computing minimal relaxations. One possible algorithm to compute possible relaxations is to compute the powerset of *FRS* and evaluate the value of the individual solutions. Because of the natural inefficiency of such an approach, we propose to apply a *conflict-directed* approach for finding relaxations: Given the definitions above, we can view the relaxation problem as a model-based diagnosis problem in the sense of [13] and correspondingly view the set of filter rules as the set of diagnosable and thus possibly faulty components of the system.

Definition (Conflict): *Given a recommendation problem RP<P, FRS, C, CRS>, a conflict CF is a subset of FRS such that there exists no solution for RP<P, CF, C, CRS>. A conflict CF is said to be minimal if there is no CF'* $\subset$ *CF which is also a conflict for RP<P, FRS, C, CRS>.*

Adapted Hitting-Set algorithm (sketch): Given a recommendation problem *RP<P, FRS, C, CRS>*, construct a HS-DAG[3] for the collection K of conflicts in breadth-first manner. Each node *n* of the HS-DAG is labelled with a conflict CS(n) $\in$ K. Edges leading away from *n* are labelled with a filter rule f $\in$ CS(n), the set of edge labels from the root to *n* is referred to as *H(n).* Every call to the *Theorem Prover* TP [13] at a node *n* returns true when there exists a solution for the adapted recommendation problem RP <P, FRS – H(n), C, CRS> (meaning that *n* can be closed) or a conflict in the other case.

Computing minimal conflicts. If we are given a recommendation problem *RP<P, FRS, C, CRS>* whose result set is empty, the whole set of (active) filter rules of course represents a conflict. However, the size of the conflict sets directly influences the size of the resulting search tree, i.e., we are in general interested in finding small or minimal conflict sets. We therefore propose to use Junker's algorithm for efficiently computing such minimal conflicts [8]: QUICKXPLAIN is a recent, non-intrusive conflict detection algorithm that – based on a divide and conquer strategy – decomposes the overall problem based on the concept of "preferred constraints". The main advantage of the approach lies in its general applicability, i.e., it is not bound to specific inference and dependency-tracking mechanisms in the underlying reasoning

[3] Hitting Set Directed Acyclic Graph; according to [13], finding the set of minimal diagnoses corresponds to computing the *Hitting Set* of all minimal conflicts.

engine. In addition, QUICKXPLAIN also supports search for "preferred" conflicts for cases where not all elements of the conflict have the same priority. When applied to our problem, conflict detection means to identify minimal sets of active filter rules that have no products in common, i.e., which will lead to an empty result set.

Best-effort search for customer-optimal relaxations. When using Reiter's algorithm, we search for diagnoses in breadth-first manner, implicitly assuming that smaller diagnoses (relaxations) are preferable. In recommender applications, however, not all of the filter rules may have equal importance for the customer. On the one hand the domain engineer might annotate the rules in advance with some priority of application based on his/her expert knowledge; on the other hand we could allow the user to express his/her preferences on the importance of rules or derive it from other (external) data sources like from the behaviour of other users.

Definition (Optimal relaxation): *Given a recommendation problem RP<P, FRS, C, CRS>, let RC be a function over the elements of FRS describing the costs of retracting a single filter $f \in FRS$. Let COSTS be a function describing the costs of a relaxation as an integer number, which takes the set of retracted filters, the individual costs RC, and eventually also customer characteristics[4] into account. A relaxation RFRS for RP is said to be optimal if there exists no other set $RFRS' \subseteq FRS$ such that*

$$COSTS(RFRS', CRS, RC) < COSTS(RFRS, CRS, RC).$$

For ensuring monotonicity of the COSTS function, COSTS(RFRS', CRS, RC) < COSTS(RFRS, CRS, RC), has to hold in cases where $RFRS' \subset RFRS$

We introduce this general cost function, as minimal relaxations may not always be optimal, i.e., it might be better to relax two rules with lower importance than to relax one rule which is highly important for the customer. Applying breadth-first search is thus not reasonable in that context because a potentially many different conflicts have to be computed for finding the first solution; exhaustive search is only possible for very small problems. We therefore propose to explore the search space in depth-first manner, such that a first (non-optimal) relaxation can be found in very short time. After having found this initial solution, we proceed by applying a branch-and-bound search in which we prune those search paths that will definitely not lead to a better solution. We also propose to implement a time-constrained best-effort variant, where we define a maximum time (e.g., one second) for optimization. Of course, optimality of the result cannot be guaranteed then in general, but our experiments suggest that even rather complex problems can be solved in very short time or *good* relaxations can be found, e.g., when using a domain-independent, priority-based search heuristic.

Depth-first algorithm (sketch): Given a recommendation problem *RP<P, FRS, C, CRS>*, construct the HS-DAG for the collection K of *minimal*[5] conflicts in depth-first manner. Label edges and nodes similar to the algorithm described previously. As a general search heuristic, sort the elements of each conflict according to their weight in ascending order. Remember the value of the optimization function for the first valid relaxation as the currently best solution. When further exploring the search tree,

[4] The cost function could, for instance, take personal preferences of the user into account.

[5] Note that we rely on *minimal* conflicts that are computed with QuickXPlain, which also means that tree pruning [13] is not required.

always remember the value of the currently best solutions and prune those paths that will definitely result in solutions which are worse than the optimum found so far. In addition – since the relaxations that are found in the depth-first approach may be non-minimal – we need to minimize each relaxation that is found before proceeding with the search process. This minimization operation, however, requires at most *n-1* additional consistency checks for a relaxation of size *n*, since we can check each of the elements of the actual relaxation individually.

Implementation and discussion

In order to evaluate our approach, we have implemented the proposed algorithms in the knowledge-based Advisor Suite [7] system. Our particular aim was to demonstrate that the diagnosis approach is suitable given the hard real-time requirements of interactive recommender applications: in general, given a set of n diagnosable components, i.e., filter rules, the computational complexity of finding diagnoses of cardinality k is O(nk). From various real-world instantiations of the Advisor Suite system we know that a typical knowledge base contains about 40-70 filter rules; we also claim that the size of allowed relaxations should be limited to a maximum of five to seven filter rules for pragmatic reasons. From practical settings we know that users are not satisfied with proposals where for instance half of the rules (corresponding to the original wishes) could not be applied. The most costly and frequent operation during the construction of the HS-DAG and the computation of conflicts is to determine for a given set of filter rules whether these rules, if applied together, lead to an empty result set. For increased performance, we have chosen to perform a pre-processing step in which we evaluate the active filter rules individually, i.e., for each of the filter rules, we set up the list of products that fulfil the filter expression. For all of the filter rules that contain no variables in the filter expression, we can pre-compute the set of products that fulfil the expression. For all other rules, we have to set up the product list at run-time for each customer session. The results of these filters can be represented as bit-sets in compact way, i.e., for each product in the catalog we set a flag, if it is in the result or not (Figure 2). The computation of the conjunction of the filter rules can then be efficiently done by applying a "bitwise and" on these bit-sets. If the resulting bit-set contains no entries, we know that the result set is empty and we have to relax some of the filters. Note that this sort of representation is not mandatory for applying our approach but was actually implemented in the Java-based Advisor Suite system. Our measurements of the running times (see below) also include numbers for an implementation variant that does not rely on these pre-processing step but rather on individual database queries (HS/DBQ).

A particular side-effect when using the pre-processing technique is that we can directly determine the optimal or minimal relaxation without even constructing the HS-DAG: If we remember for each product the set of filter rules that had a "0" for the product and the sum of the weights of these rules (see Figure 2) we only have to determine that product in the (empty) result set which minimizes the given optimization function. As such, the search for the best valid relaxation can be done by only evaluating each filter rule once, computing the conjunction of the bit-sets, and

finding the minimum by scanning the result set (compare also the results of [6] for finding the first *maximal succeeding subquery*). We could therefore compute the optimal relaxation for all test cases in a few milliseconds (see column BSA in the table below). Still, this is only possible due to a very specific implementation in our system and is not as general as the described conflict-directed approach.

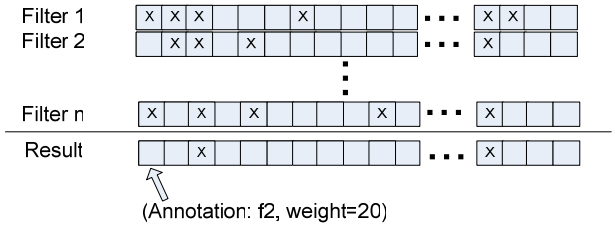

Figure 2 Bit-set representation of filter rules

In the different test examples we varied the relaxation goal (cardinality only/optimization), the number of (active) filter rules, the number of products and conflicts, the average size of the conflicts, and the cardinality of the minimal diagnoses[6]. Our test databases contained up to 2.000 different products – the ramp up time for constructing the needed data structures is about 2 seconds on a Pentium IV desktop PC, which is not problematic since this has to be done only once during server start-up. The additional memory requirement for the data structures alone is *nbProducts* * *nbFilterRules* bits and thus strictly limited. The following tables show the average running times for a rather hard test case for one of our real-world knowledge bases that contains 70 filter rules. In the test cases, 25 of them were active for some given customer requirements. The number of and size of the conflicts was adequately increased with the size of the minimal relaxation that was to be found. The numbers in the table correspond to average running times in milliseconds for finding the first/best relaxation.

C	HS/BS	HS/DBQ	BSA	#N	#TP
1	12	177.1	10	1	15
3	14	345.6	10	4	22
5	14	416.6	10	23	75
7	18	472.9	10	95	99
10	67	861.2	10	1346	228

Table 1: Breadth-first search for first diagnosis

C	HS/BS	HS/DBQ	#N	#TP
1	12	164.5	1	15
3	15	338.5	6	22
5	16	414.6	19	75
7	16	465.8	62	99
10	35	697.1	471	187

Table2: Depth-first / Branch & bound for optimal diagnosis

C: Cardinality of smallest relaxation

HS/BS: HS-DAG construction, usage of bit-set representation

HS/DBQ: HS-DAG construction, usage of dynamically constructed queries

BSA: Analysis of bit-sets only (no HS-DAG)

#N: Number of search tree nodes

#TP: "Theorem prover" calls: DB-queries / Bit-set operations

[6] Note that a relaxation of size 10 is already rather unrealistic and of little help for the user.

In that test case, all of the problems could be solved within the targeted time-frame of one second, even when using database queries instead of the bit-set representation. If no pre-computation of filter results was done, e.g., because all filter rules contain variables, the search process needed about 40 additional milliseconds in our test cases. The performance of the depth-first search in that test case is comparable (or slightly better) than the breadth-first approach, i.e., a good solution was found early and a lot of pruning could be done, i.e., fewer nodes had to be expanded. In general this will vary, depending on the number of conflicts and diagnoses, the search heuristic for the depth-first search and the particular characteristics of the importance factors and so forth. An implementation variant based on A* produced results comparable to those of the depth-first search.

Related work and Conclusions

Related work. Query-relaxation for content-based recommenders was recently addressed in [9] and [10]. In this work, McSherry proposed an approach in which the user is incrementally presented individual explanations for the retrieval failure and the problem is iteratively solved. Such system behaviour can in principle also be implemented based on our approach if we compute a minimal conflict in each step and let the user decide how to proceed. Nonetheless, there might be many interactions required (e.g., if we need to relax many filters), the system will not be able to predict in general if the relaxation of the next sub-query will succeed, and no overall optimum can be guaranteed. In addition, the number of queries required for computing all conflicts before entering the recovery process may be too large for interactive applications in their approach. An comparable iterative repair approach is also implemented in our surrounding framework [7], where the user can interactively change the priorities of the rules according to his/her preferences.

The identification of the reasons for a failure of a query is also an issue in the area of Cooperative Query Answering. In [6], Godfrey shows complexity results for finding *Minimal Failing Sub-queries* MSF (correspond to *conflicts* in our approach) and *maximal succeeding queries* XSS (corresponding to our relaxed query) and proposes algorithms for finding one and enumerating all MSF/XSS. Our algorithms can be seen as alternatives to these basic mechanisms which are in addition capable of exploiting priorities for steering the (best-effort) search process and also incorporate recently developed algorithms for fast detection of minimal conflicts [8].

Incremental relaxation of overconstrained problems is also a well-known technique in the Constraint Satisfaction world, see e.g., [12] or [14], where *soft constraints* and *Partial Constraint Satisfaction* were introduced. These approaches, however, were designed to work in the context of some specific reasoning technique and cannot be easily applied to other problem solving or inference algorithms.

From the technological perspective, model-based diagnosis techniques for detecting problems in Software systems have been introduced e.g., in [2] or [4]. We see our approach in the general tradition of that work. Our future work includes the extension of the approach towards the computation of "repair alternatives": Instead of computing only possible relaxations, the goal will be to find a minimal set of changes in the user requirements such that a solution can be found. Finally, we also aim at

evaluating approaches that exploit abstraction hierarchies within the knowledge base (see e.g., [3]), an approach that was also suggested as a tool for tackling the relaxation problem for recommenders in [11].

Conclusions. In this paper, we have shown how conflict-directed search can be used to significantly enhance the search for minimal relaxations of unsuccessful queries in filter-based recommender systems. The evaluation of the approach in a knowledge-based advisory framework has shown that even hard problem instances can be solved within the short time frames that are required in interactive recommender applications. Furthermore it was demonstrated how the usage of specialized data structures and pre-processing techniques can further reduce the number of required database queries and thus shorten response times. In our current work, we aim at generalizing this pre-processing approach and also extending it with a mechanism with which we can also compute relaxations (based on the bit-set matrix) that lead at least to *n* different products, which shall give the end user more choices when searching and comparing different items in a proposal.

References

[1] D. Bridge. Product recommendation systems: A new direction. In R. Weber and C. Wangenheim, eds., Proceedings of the Workshop Programme at the Fourth International Conference on Case-Based Reasoning, 2001, p. 79-86.
[2] L. Console, G. Friedrich, D. T. Dupré: Model-Based Diagnosis Meets Error Diagnosis in Logic Programs. IJCAI'1993, Chambéry, France, 1993, p. 1494-1501.
[3] A. Felfernig A., G. Friedrich, D. Jannach, M. Stumptner: Hierarchical Diagnosis of large configurator knowledge bases. In: S. McIlraith, D. T. Dupré (Eds.): 12th Intl. Workshop on Principles of Diagnosis, 2001, p. 55-62.
[4] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner. Consistency-based diagnosis of configuration knowledge bases, Artificial Intelligence, 152(2), 2004, p. 213-234.
[5] E. Freuder, R. J. Wallace: Partial Constraint Satisfaction, Artificial Intelligence (58), 1992.
[6] P. Godfrey. Minimization in Cooperative Response to Failing Database Queries, International Journal of Cooperative Information Systems Vol. 6(2), 1997, p. 95-149.
[7] D. Jannach. ADVISOR SUITE - A knowledge-based sales advisory system. In: Proceedings of ECAI/PAIS 2004, Valencia, Spain, IOS Press, 2004, pp. 720-724.
[8] U. Junker: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. Proceedings AAAI'2004, San Jose, 2004, p. 167-172.
[9] D. McSherry, Explanation of Retrieval Mismatches in Recommender System Dialogues, ICCBR'03 Mixed-Initiative Case-Based Reasoning Workshop, Trondheim, Norway, 2003.
[10] D. McSherry, Incremental Relaxation of Unsuccessful Queries, Lecture Notes in Computer Science, Volume 3155, 2004, p. 331–345.
[11] N. Mirzadeh, F. Ricci and M. Bansal, Supporting User Query Relaxation in a Recommender System, Proceedings of 5th Intl. Conference on Electronic Commerce and Web Technologies, EC-Web '04, Zaragoza, Spain, 2004, p. 31-40.
[12] S. Mittal, B. Falkenhainer. Dynamic constraint satisfaction problems. In: Proceedings of the 8th National Conference on Artificial Intelligence, AAAI'89, 1989, p. 25-32.
[13] R. Reiter. A theory of diagnosis from first principles. Artificial Intelligence, 32(1), Elsevier, 1987, p. 57-95.
[14] T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: Hard and easy problems. In International Joint Conference on Artificial Intelligence IJCAI'95, Montreal, Canada, 1995, p. 631-639.