Learning Uncertain Rules with CONDORCKD [*]

Jens Fisseler
Department of Computer Science
University of Hagen
58084 Hagen
Germany

Gabriele Kern-Isberner
Department of Computer Science
University of Dortmund
44227 Dortmund
Germany

Christoph Beierle
Department of Computer Science
University of Hagen
58084 Hagen
Germany

Abstract

CONDORCKD is a system implementing a novel approach to discovering knowledge from data. It addresses the issue of relevance of the learned rules by algebraic means and explicitly supports the subsequent processing by probabilistic reasoning. After briefly summarizing the key ideas underlying CONDORCKD, the purpose of this paper is to present a walk-through and system demonstration.

Introduction

Knowledge discovery in databases (KDD) is the process of identifying novel and interesting patterns in data (Fayyad, Piatetsky-Shapiro, & Smyth 1996). The data set typically consists of possibly millions of objects, each described by a set of measurements or attributes, and the pattern types searched for can be roughly categorized as either descriptive or predictive. Assessing the novelty and interestingness of the discovered patterns is a difficult task by itself (Hilderman & Hamilton 1999), and in most cases the subsequent use of the discovered knowledge is ignored during the discovery process. The purpose of this paper is to present a walk-through and system demonstration of CONDORCKD, an implementation of a novel approach to learning (uncertain) rules from data. CONDORCKD addresses the issue of relevance by algebraic means and explicitly supports the subsequent processing of information via the principle of maximum entropy.

We start by summarizing some key ideas underlying CONDORCKD and briefly sketching its implementation.

CONDORCKD – Conditional Knowledge Discovery by Algebraic Means

The focus of the knowledge discovery method that we developed and implemented during the CONDOR project (Beierle & Kern-Isberner 2003b) was on linking the learning process more closely to the subsequent inference process – the learned patterns ought to be useful in the sense that they could serve directly as inputs for knowledge representation and reasoning. We found rules or conditionals, respectively, to be adequate patterns that satisfy the demand for finding generic and understandable knowledge, on the one hand, and that reveal important structures for inductive probabilistic inferencing, on the other.

There are two key ideas underlying the approach we used for our implementation: First, the link between structural and numerical knowledge is established by an algebraic theory of conditionals, which makes it possible to consider complex interactions between rules (Kern-Isberner 2000). By applying this theory, we develop an algorithm that computes sets of probabilistic rules from distributions. Second, knowledge discovery is understood as a process which is inverse to inductive knowledge representation, i.e. a set of rules is searched for that may generate the investigated distribution via induction, hence serving as a basis. In this way, the inductive processing of information is anticipated and used in order to improve the conciseness of the output. In particular, the relevance of discovered information is judged with respect to the chosen induction method, giving rise to a formal quality criterion that is implicitly applied to select "best" rule sets. The inductive representation method used here is based on maximizing entropy, an information theoretical principle (*ME-principle* (Paris 1994)). So, the discovered rules can be considered as being most informative in a strict, formal sense, and can be used directly as input for the SPIRIT system which processes probabilistic information on optimum entropy and may be used e.g. for decision support (Rödder, Reucher, & Kulmann 2006). This approach is described in detail in (Kern-Isberner & Fisseler 2004); we will give a brief overview in this section, also presenting a small running example, cf. (Fisseler *et al.* 2007). A more complex example will be given in the walk-through.

Example 1 *Suppose in our universe there are* animals *(A),* fish *(F),* aquatic beings *(Q),* objects with gills *(G) and* objects with scales *(S). The following table may reflect our observations:*

object	*freq.*	*prob.*	*object*	*freq.*	*prob.*
$afqgs$	59	0.5463	$a\overline{f}qgs$	11	0.1019
$afqg\overline{s}$	21	0.1944	$a\overline{f}qg\overline{s}$	9	0.0833
$afq\overline{g}s$	6	0.0556	$a\overline{f}q\overline{g}\,\overline{s}$	2	0.0185

We are interested in any relationship between these objects, e.g., to what extent can we expect an animal that is an

[*]The research reported here was partly supported by the DFG – Deutsche Forschungsgemeinschaft (grant BE 1700/5-3).

```
                    Algorithm CKD
          (Conditional Knowledge Discovery)

Input   A frequency/probability distribution $P$
Output  A set of probabilistic conditionals

Begin
    % Initialization
    Compute the list NC of null-conjunctions
    Compute the set $\mathcal{S}_0$ of basic rules
    Compute ker $P$
    Compute ker $g$
    Set $\mathcal{K} :=$ ker $g$
    Set $\mathcal{S} := \mathcal{S}_0$

    % Main loop
    While equations are in $\mathcal{K}$ Do
        Choose $gp \in \mathcal{K}$
        Modify (and aggregate) $\mathcal{S}$
        Modify (and reduce) $\mathcal{K}$
    Calculate the probabilities of the conditionals in $\mathcal{S}$
    Return $\mathcal{S}$ and appertaining probabilities
End.
```

Figure 1: High-level description of our CKD algorithm (Kern-Isberner & Fisseler 2004).

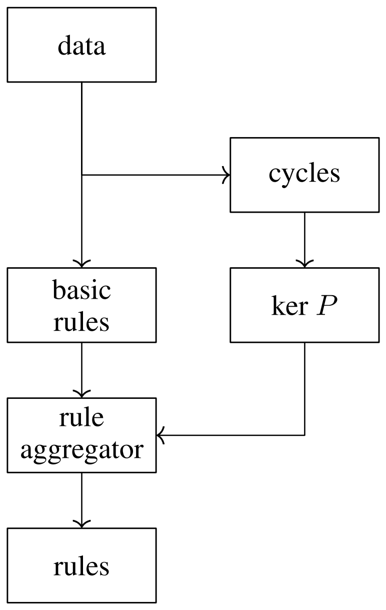


Figure 2: Dataflow of the CONDORCKD algorithm.

aquatic being with gills to be a fish? This relationship is expressed by the conditional $(f|aqg)$, *which is read as "f, under the condition a and q and g". If P is the probability distribution given by the table above and* $x \in [0,1]$ *is a probability value, P satisfies the* probabilistic conditional $(f|aqg)[x]$, *written as* $P \models (f|aqg)[x]$ *iff the conditional probability* $P(f|aqg) = x$. *In our example, it is easily calculated that* $P \models (f|aqg)[0.8]$.

From the given data, a frequency distribution is computed which is mined for structural information to be processed in the algebraic machinery. Here, conditionals are represented by group generators, and group homomorphisms are used as a formal means to elaborate and process structural information. As is well-known from algebraic theories, the kernels of such homomorphisms reflect structural invariants. In our case, so-called *conditional indifferences* are discovered in probability distributions. Conditional indifference is a notion that may help organizing probabilistic networks, in a similar way as conditional independencies do so for Markov and Bayesian networks. Indeed, conditional indifference turned out to be more finely grained than conditional independence (for further details, cf. (Kern-Isberner 2000; 2004; Beierle & Kern-Isberner 2003a)). An overview of the algorithm in pseudocode is given in Figure 1 and its data flow is illustrated in Figure 2; both will be explained in a bit more detail in the following. The algorithm has been implemented as a component of the CONDOR system (for an overview, cf. (Beierle & Kern-Isberner 2003b))

The frequency distributions calculated from data are mostly sparse, full of zeros, with only scattered clusters of non-zero probabilities. In our approach, these zero values are treated as non-knowledge without structure. They play a prominent role in setting up a set $\mathcal{S}_0$ of *basic rules* of manageable size in the beginning. The set *NC* of *null-conjunctions NC* (i.e. conjunctions of attribute-value-literals, with frequency 0) represents missing information in a most concise way. Next, the numerical relationships in P have to be explored to set up the structural kernel of P, *ker* P. Any such relationship is found to correspond to a simple cycle of even length (i.e. involving an even number of vertices) in an associated graph. Therefore, the search for numerical relationships holding in P amounts to searching for such cycles in a graph. Finally, as the last step of the initialization, the kernel of a structure homomorphism, *ker* g, has to be computed from *ker* P with respect to the set $\mathcal{S}_0$ of conditionals. In this way, algebraic representations of numerical probabilistic information are obtained, which are encoded as equations holding in groups associated with the respective set of conditionals. Solving these equations successively yields modifications both on the groups and on the appertaining conditionals.

So, in the main loop of the algorithm *CKD*, the sets $\mathcal{K}$ of group elements and $\mathcal{S}$ of conditionals are subject to change. In the beginning, $\mathcal{K} =$ *ker* g and $\mathcal{S} = \mathcal{S}_0$; in the end, $\mathcal{S}$ will contain the discovered conditional relationships. Note that no probabilities are used in this main loop – only structural information (derived from numerical information) is processed. It is only afterwards, that the probabilities of the conditionals in the final set $\mathcal{S}$ are computed from P, and the probabilistic conditionals are returned.

Example 2 *We continue Example 1. First, the set* NC *of* null-conjunctions *has to be calculated from the data; here, we find* $NC = \{\overline{a}, \overline{q}, \overline{f}\,\overline{g}\}$ *– no object matching any one of these partial descriptions occurs in the data base. These null-conjunctions are crucial to set up a starting set* $\mathcal{S}_0 = \mathcal{B}$

of basic rules of feasible size:

$$\begin{array}{lrcllcl}
\mathcal{B} = \{ & \phi_{f,1} & = & (f|aqgs) & \phi_{g,1} & = & (g|afqs) \\
 & \phi_{f,2} & = & (f|aqg\overline{s}) & \phi_{g,2} & = & (g|afq\overline{s}) \\
 & \phi_{f,3} & = & (f|\overline{g}) & \phi_{g,3} & = & (g|\overline{f}) \\
 & \phi_{s,1} & = & (s|afqg) & \phi_{a,1} & = & (a|\top) \\
 & \phi_{s,2} & = & (s|afq\overline{g}) & & & \\
 & \phi_{f,3} & = & (f|a\overline{f}qg) & \phi_{q,1} & = & (q|\top)\ \}
\end{array}$$

The next step is to analyze numerical relationships in P. In this example, we find two numerical relationships that hold with near equality:

$$P(a\overline{f}qgs) \approx P(a\overline{f}qg\overline{s}) \quad \textit{and} \quad \frac{P(afqgs)}{P(afqg\overline{s})} \approx \frac{P(afq\overline{g}s)}{P(afq\overline{gs})}$$

These relationships are translated into algebraic group equations and help modifying the set of rules. For instance, $\phi_{f,1}$ and $\phi_{f,2}$ are joined to yield $(f|aqg)$, and $\phi_{s,3}$ is eliminated. As a final output, the CKD algorithm returns the following set of conditionals:

Conclusion	*Premise*	*Prob.*
A=YES		1.0
F=YES	*G={NO}*	1.0
F=YES	*A={YES}, Q={YES}, G={YES}*	0.8
Q=YES		1.0
G=YES	*F={NO}*	1.0
G=YES	*A={YES}, F={YES}, Q={YES}*	0.91
S=YES	*A={YES}, F={YES}, Q={YES}*	0.74

This result can be interpreted as follows: All objects in our universe are aquatic animals which are fish or have gills (corresponding to the four rules with probability 1.0*). Aquatic animals with gills are mostly fish (with a probability of* 0.8*), aquatic fish usually have gills (with a probability of* 0.91*) and scales (with a probability of* 0.74*).*

Note that our system already generated the LaTeX-code for the table given above as output.

Implementing CondorCKD

We implemented CondorCKD using Haskell (Hudak *et al.* 1992), a non-strict, strongly typed functional programming language. Using a functional language allowed us to quickly implement a working prototype, which could subsequently be refined and improved. Other advantages of Haskell are its clean syntax, and the conciseness of functional languages in general. The whole *documented* code-base involves little more than 9000 lines of code, whereas an implementation written in C++ or Java can be expected to be several times larger. More details are given in (Fisseler *et al.* 2007).

In the following two subsection we want to give a short overview of two crucial, language-independent parts of CondorCKD, before illustrating its application in the next section.

A data structure for probability distributions

Like most data mining algorithms, CondorCKD processes tabular data, treating this data as a discrete multivariate probability distribution with a set $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ of random variables, each with a finite domain. Every record of the data set represents a *complete conjunction* of the variables $\mathcal{V}$, i.e. a conjunction of attribute-value-literals, one for each $V \in \mathcal{V}$. Whereas the given tabular data representation can be used for computing the probability of each (complete) conjunctions (although this would be highly inefficient), the requirement to be able to compute which conjunctions do *not* occur in the given data (the so-called *null-conjunctions*), makes this simple representation inappropriate for CondorCKD. Thus, to support frequency computations and the computation of null-conjunctions, CondorCKD utilizes a tree-like data structure for representing the probability distribution in memory. Each internal node of this *distribution tree* (*DTree*) is marked with one of the variables $V \in \mathcal{V}$, and every edge leaving an internal node labeled with variable V_i is assigned one of the values v_i of V_i. As no two nodes on any path from the root to another node are marked with the same variable, each of these paths depicts a conjunction, whose frequency is stored at the node that path ends in. Paths depicting complete conjunctions end at a leaf node.

A DTree can easily be built by starting with an empty tree and processing each record of the data set in turn, thereby building only those parts of the DTree containing non-null conjunctions. The empty DTree has implicitly assigned the same variable to all internals nodes of the same depth, i.e. the root node would be assigned variable V_1, all of its children would be assigned variable V_2, and so on. As each data record represents a complete conjunction, it depicts a path from the root node of the DTree to a leaf. If all nodes of such a path are already part of the DTree, the frequencies stored at all path nodes only need to be incremented by one in order to account for the current record. Otherwise the path can be divided into a prefix and suffix. All prefix nodes are already part of the DTree, and thus their frequencies need to be incremented by one. The suffix nodes are not yet part of the tree and are inserted, each with a frequency of 1. Building a DTree this way ensures that only those parts of the DTree containing non-zero conjunctions are built, leading to memory savings in comparison to a complete tree, because those parts of the tree containing only null-conjunctions will never be created.

Once the DTree representing a data set has been built, the null-conjunctions can be computed by a simple tree traversal. For example, suppose we have built a DTree for a data set with four binary variables A, B, C and D, whose only null-conjunctions are $abc\overline{d}$, $\overline{a}bc\overline{d}$, $ab\overline{c}\overline{d}$, $\overline{a}b\overline{c}\overline{d}$. Traversing the tree will yield these four null-conjunctions, but note that by rearranging the variables (and thereby the DTree) to B, D, A and C, a traversal would give the shorter null-conjunction $b\overline{d}$. Thus, in order to obtain more concise null-conjunctions, several reordering heuristics can be applied.

These heuristics change the structure of the DTree by reordering the variables in the subtrees, trying to find an order-ing that will possibly yield shorter null-conjunctions. The DTree is processed in-order, starting at its root. At every inner node, one of the remaining variables (i.e. one of the variables not marking any other node on the path from the current node to the root) is chosen to mark the current inner node. One way to choose a variable is to favor variables with many possible values, resulting in a DTree with large branching factors near the root, which will possibly result in shorter null-conjunctions, as some of combinations of literals near the root might not occur in the data set.

Computing *ker* P

Another important part of the implementation of CONDORCKD is the computation of *ker* P, the kernel of the multivariate probability distribution P. As stated in the previous section, the computation of *ker* P amounts to searching for simple, even-length cycles in an undirected graph. There are several approaches to compute the simple cycles of an undirected graph (Mateti & Deo 1976), of which the *vector space* approach and *search-based algorithms* are the most important ones. We will introduce both approaches briefly, before sketching our new combined approach.

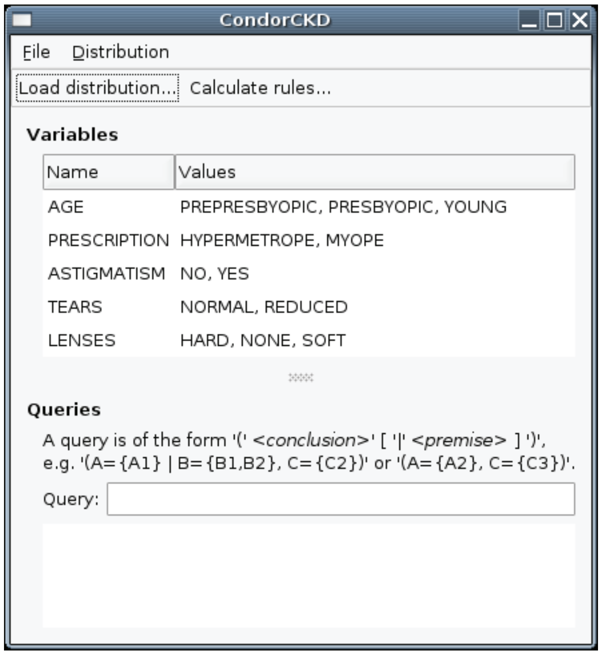
Algorithms utilizing the vector space approach for computing the cycles of an undirected graph first compute a set of so-called *fundamental cycles*, from which all cycles can be generated by combining the fundamental cycles. As there are exponentially many combinations of fundamental cycles, vector space algorithms must try to compute as few of these combinations as possible, yet very little has been done regarding pruning these unnecessary computations, let alone incorporating cycle length restrictions.

Search-based algorithms use a modified depth-first search with backtracking, during which edges are appended to a path until a cycle is found. Although search-based algorithms are the fastest known algorithms for enumerating all cycles of an undirected graph, even their running times on several of the graphs induced by our data sets were prohibitively large. Therefore, we utilize certain characteristics of the vector space approach in order to further reduce the search space for the search-based algorithm. Basically, we associated a subgraph to each fundamental cycle, and conducted a search-based cycle enumeration in this subgraph only. Details can be found in (Fisseler *et al.* 2007).

We have compared the running time[1] of our novel algorithm (column "FCs + DFS") and that of a standard search-based algorithm (DFS) on three different graphs, and the preliminary results are encouraging (see Figure 3).

Graph	*Max. cycle length*	*#Cycles*	*FCs + DFS*	*DFS*
A	10	2827	0:00:01	0:00:02
	12	16699	0:00:05	0:00:13
	14	119734	0:00:45	0:08:13
	16	890204	0:05:48	7:23:54
B	10	2929	0:00:04	0:00:06
	12	23021	0:00:42	0:01:28
	14	222459	0:09:11	0:42:09
C	6	2927	0:00:10	0:01:26
	8	18695	0:00:36	0:02:13
	10	268097	0:13:51	0:30:27

Figure 3: Runtime comparison.



Name	Values
AGE	PREPRESBYOPIC, PRESBYOPIC, YOUNG
PRESCRIPTION	HYPERMETROPE, MYOPE
ASTIGMATISM	NO, YES
TEARS	NORMAL, REDUCED
LENSES	HARD, NONE, SOFT

A query is of the form '(' *<conclusion>* ' ['|' *<premise>*] ')', e.g. '(A={A1} | B={B1,B2}, C={C2})' or '(A={A2}, C={C3})'.

Figure 4: Main window of CONDORCKD after loading the "Lenses" database.

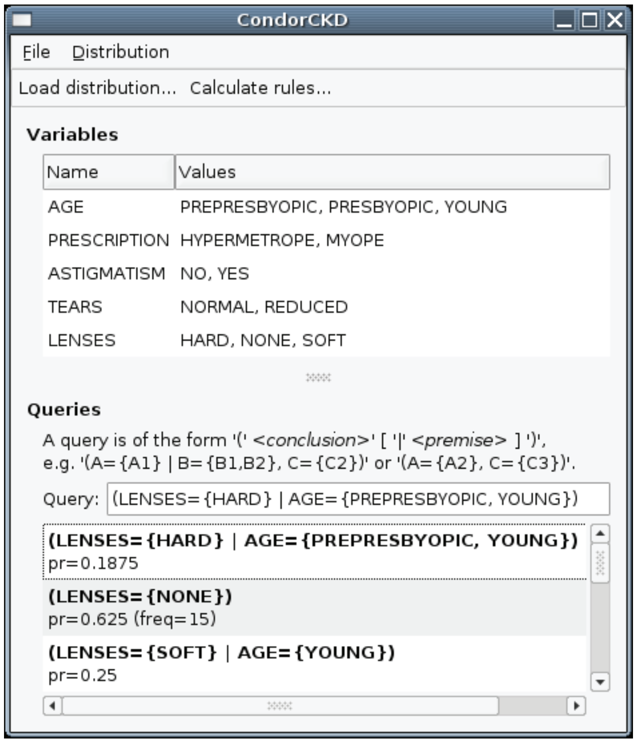
CONDORCKD – Walk-through and System Demonstration

Having introduced the theory of CONDORCKD and two important parts of its implementation, we now want to exemplify its application on the "Lenses" database from the UCI machine learning repository (Newman *et al.* 1998). The "Lenses" database contains information about the type of contact lenses suitable for what kind of patients. This decision is based on certain medical conditions of the patient, e.g. whether she is long-sighted, suffers from reduced tear production or astigmatism.

The first step when working with CONDORCKD is, of course, loading the data file into memory. Currently, CSV (comma-separated values) and ARFF (Witten & Frank 2005) files are supported, as these are the two most common file formats supported by data mining systems. After loading the data, CONDORCKD displays the list of variables and their corresponding ranges, cf. Figure 4, which shows the main window of CONDORCKD after loading the "Lenses" database.

After loading a database, the user could immediately initiate the calculation of the conditionals. But especially with new and unknown data sets, acquiring a better understand-

[1]Implementation in Haskell, compiled with GHC 6.4.1, optimizations turned on and code generation via C. The executables were running on an AMD Athlon64-3200+ in 32bit-mode with 1GB RAM, using Linux.



Variables

Name	Values
AGE	PREPRESBYOPIC, PRESBYOPIC, YOUNG
PRESCRIPTION	HYPERMETROPE, MYOPE
ASTIGMATISM	NO, YES
TEARS	NORMAL, REDUCED
LENSES	HARD, NONE, SOFT

Queries

A query is of the form '(' <*conclusion*>' ['|' <*premise*>] ')',
e.g. '(A={A1} | B={B1,B2}, C={C2})' or '(A={A2}, C={C3})'.

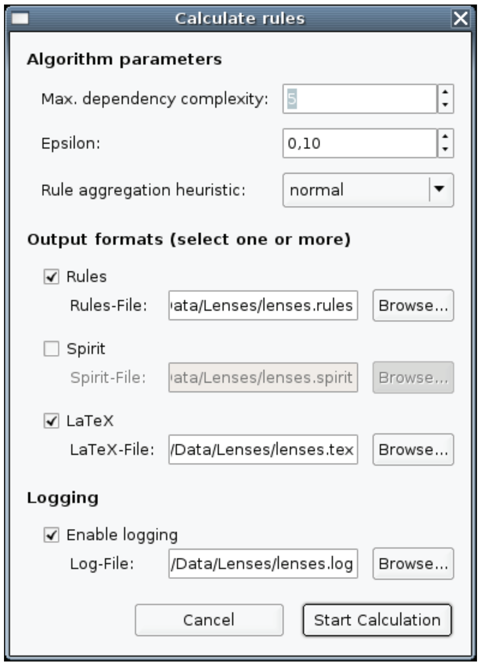
Query: (LENSES={HARD} | AGE={PREPRESBYOPIC, YOUNG})

(LENSES={HARD} | AGE={PREPRESBYOPIC, YOUNG})
pr=0.1875

(LENSES={NONE})
pr=0.625 (freq=15)

(LENSES={SOFT} | AGE={YOUNG})
pr=0.25

Figure 5: Main window of CONDORCKD after posing several queries.



Algorithm parameters

Max. dependency complexity: 5

Epsilon: 0,10

Rule aggregation heuristic: normal

Output formats (select one or more)

[✓] Rules
Rules-File: ›ata/Lenses/lenses.rules

[ ] Spirit
Spirit-File: ›ata/Lenses/lenses.spirit

[✓] LaTeX
LaTeX-File: /Data/Lenses/lenses.tex

Logging

[✓] Enable logging
Log-File: /Data/Lenses/lenses.log

Figure 6: "Calculate rules" dialog window.

ing of the data is an essential part of the data mining process (Fayyad, Piatetsky-Shapiro, & Smyth 1996), as it allows for a better comprehension of the data mining results, or even checking some hypotheses about the data before mining. To support interactive exploration of a data set, CONDORCKD offers a facility to query the probability distribution given by the data, i.e. to compute the (conditional) probability of a given conjunction. E.g., given the "Lenses" database, one might be interested in the probability of people wearing soft lenses, given they are young. This question is an example of a conditional query, which can be posed to CONDORCKD by entering $(\mathit{lenses} = \{\mathit{soft}\} \,|\, \mathit{age} = \{\mathit{young}\})$ The probability of this query is 0.25, because a fraction of 0.25 of the records with the *age*-variable having value *young* also have the *lenses*-variable set to *soft*. This and several other queries together with their results can be seen in Figure 5. In general, queries are of the form $(\langle \mathit{conclusion} \rangle)$ for unconditional, and $(\langle \mathit{conclusion} \rangle \,|\, \langle \mathit{premise} \rangle)$ for conditional queries. *Premise* and *conclusion* are conjunctions of attribute-valueset-literals, hence the curly braces that can be seen in Figure 5. One example of a query using a set of values for a query-variable is the question of the probability of people wearing hard lenses, given they are young or not yet presbyopic, i.e. pre-presbyopic. This query can be posed as $(\mathit{lenses} = \{\mathit{hard}\} \,|\, \mathit{age} = \{\mathit{young, prepresbyopic}\})$, with a resulting probability of 0.1875.

Before initiating the actual data mining, the user can adjust several parameters of the CONDORCKD-algorithm. Figure 6 shows the "Calculate rules"-dialog, which is used for this purpose. The "maximum dependency complexity"-parameter restricts the complexity of the numerical relationships used for rule aggregation. In addition to being represented by simple cycles of even length, these numerical relationships must be close to 1, where the degree of closeness is specified by ε. The value of the parameter ε can be varied within $[0, 0.5]$. The third parameter of CONDORCKD is the so-called "rule aggregation heuristic". Though CONDORCKD is based on a non-heuristic theory, complex real-world data necessitates the relaxation of certain aspects of the algorithm. With the help of the rule aggregation heuristic, the user can control the extent of conjoining and shortening of rules. The default heuristic should suffice for most data sets, but certain databases may require more "cautious" aggregation, or, on the contrary, "bold"er aggregation.

Having specified the parameters of CONDORCKD, the user can select several output formats. Currently, CONDORCKD can output the discovered conditionals as plain text, in a special format used by the ME-based expert system shell SPIRIT (Rödder & Kern-Isberner 1997; Rödder, Reucher, & Kulmann 2006), or as LaTeX-code. Additionally, the user can enable the output of logging information, though this is only of interest to the developers.

After choosing appropriate parameters, the user can finally initiate the rule computation. CONDORCKD will then calculate the ME-optimal rules for the given database and the specified parameters. Due to the theory of CONDORCKD, the rule computation can be done independently for each attribute-value-literal of the rule conclusion, which reduces the memory consumption and would allow for a possible parallelization. Nonetheless, a lot of rules have to be processed even for moderately complex data sets. Using the parameter settings depicted in Figure 6, we want to demonstrate some steps of CONDORCKD, especially those leading to the rule $(lenses = \{soft\} \mid astigmatism = \{no\})$.

There are, in principle, 12 basic rules having $lenses = \{soft\}$ as their conclusion and the literal $astigmatism = \{no\}$ in their premise; this follows from the $2 \cdot 2 \cdot 3 = 12$ possible combinations of the literals of the remaining variables *age*, *prescription* and *tears*. By interpreting zero probabilities as missing information on structures, these 12 rules can be reduced to 8 rules, of which the following five can be conjoined:

$$\begin{array}{l}
(lenses = soft \mid age = prepresbyopic, prescription = myope, \\
\qquad astigmatism = no) \\
(lenses = soft \mid age = young, prescription = myope, \\
\qquad astigmatism = no) \\
(lenses = soft \mid age = prepresbyopic, prescription = hypermetrope, \\
\qquad astigmatism = no, tears = normal) \\
(lenses = soft \mid age = presbyopic, prescription = hypermetrope, \\
\qquad astigmatism = no, tears = normal) \\
(lenses = soft \mid age = young, prescription = hypermetrope, \\
\qquad astigmatism = no, tears = normal).
\end{array}$$

E.g., because of the numerical relationship

$$P\begin{pmatrix} lenses = soft, \\ age = prepresbyopic, \\ prescription = hypermetrope, \\ astigmatism = no, \\ tears = normal) \end{pmatrix} \approx P\begin{pmatrix} lenses = soft, \\ age = young, \\ prescription = myope, \\ astigmatism = no \end{pmatrix}$$

the second and third rule can be conjoined, yielding the more concise rule

$$(lenses = soft \mid age = \{prepresbyopic, young\}, astigmatism = no).$$

Utilizing the remaining numerical relationships, all five rules can be conjoined, resulting in $(lenses = soft \mid astigmatism = \{no\})$. This rule represents all information (with respect to the ME-principle) contained in the original five rules.

Conclusions and further work

CONDORCKD is a system implementing a novel approach to knowledge discovery that is based on algebraic means and closely links the learning process to a subsequent inference process. We are currently working on several extensions of this approach, including the partitioning of larger sets of attribute sets into manageable parts and further improvement of the probability distribution representation (cf. (Moore & Lee 1998)).

Acknowledgements We'd like to thank the anonymous reviewers for their helpful comments.

References

Beierle, C., and Kern-Isberner, G. 2003a. An alternative view of knowledge discovery. In *Proceedingsf the 36th Annual Hawaii International Conference on System Sciences, HICSS-36*, 68.1. IEEE Computer Society.

Beierle, C., and Kern-Isberner, G. 2003b. Modelling conditional knowledge discovery and belief revision by abstract state machines. In Boerger, E.; Gargantini, A.; and Riccobene, E., eds., *Abstract State Machines 2003 – Advances in Theory and Applications, Proceedings 10th International Workshop, ASM2003*, 186–203. Springer, LNCS 2589.

Fayyad, U. M.; Piatetsky-Shapiro, G.; and Smyth, P. 1996. *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press. Chapter From Data Mining to Knowledge Discovery: An Overview, 1–34.

Fisseler, J.; Kern-Isberner, G.; Beierle, C.; Koch, A.; and Müller, C. 2007. Algebraic knowledge discovery using Haskell. In *Practical Aspects of Declarative Languages, 9th International Symposium, PADL 2007*, Lecture Notes in Computer Science. Berlin, Heidelberg, New York: Springer. (to appear).

Hilderman, R. J., and Hamilton, H. J. 1999. Knowledge discovery and interestingness measures: A survey. Technical Report CS-99-04, Department of Computer Science, University of Regina.

Hudak, P.; Jones, S. L. P.; Wadler, P.; Boutel, B.; Fairbairn, J.; Fasel, J. H.; Guzmán, M. M.; Hammond, K.; Hughes, J.; Johnsson, T.; Kieburtz, R. B.; Nikhil, R. S.; Partain, W.; and Peterson, J. 1992. Report on the programming language Haskell, a non-strict, purely functional language. *SIGPLAN Notices* 27(5):R1–R164.

Kern-Isberner, G., and Fisseler, J. 2004. Knowledge discovery by reversing inductive knowledge representation. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR-2004*, 34–44. AAAI Press.

Kern-Isberner, G. 2000. Solving the inverse representation problem. In *Proceedings 14th European Conference on Artificial Intelligence, ECAI'2000*, 581–585. Berlin: IOS Press.

Kern-Isberner, G. 2004. A thorough axiomatization of a principle of conditional preservation in belief revision. *Annals of Mathematics and Artificial Intelligence* 40(1-2):127–164.

Mateti, P., and Deo, N. 1976. On algorithms for enumerating all circuits of a graph. *SIAM Journal on Computing* 5(1):90–99.

Moore, A., and Lee, M. S. 1998. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research* 8:67–91.

Newman, D. J.; Hettich, S.; Blake, C. L.; and Merz, C. J. 1998. UCI repository of machine learning databases.

Paris, J. 1994. *The uncertain reasoner's companion – A mathematical perspective*. Cambridge University Press.

Rödder, W., and Kern-Isberner, G. 1997. Representation and extraction of information by probabilistic logic. *Information Systems* 21(8):637–652.

Rödder, W.; Reucher, E.; and Kulmann, F. 2006. Features of the expert-system-shell SPIRIT. *Logic Journal of IGPL* 14(3):483–500.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.