Knowl Inf Syst (2012) 32:383–414
DOI 10.1007/s10115-011-0432-2

REGULAR PAPER

Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures

Wilhelmiina Hämäläinen

Received: 28 January 2011 / Revised: 3 May 2011 / Accepted: 25 June 2011 /
Published online: 28 July 2011

Abstract Statistical dependency analysis is the basis of all empirical science. A commonly occurring problem is to find the most significant dependency rules, which describe either positive or negative dependencies between categorical attributes. In medical science, for example, one is interested in genetic factors, which can either predispose or prevent diseases. The requirement of statistical significance is essential, because the discoveries should hold also in future data. Typically, the significance is estimated either by Fisher's exact test or the χ^2-measure. The problem is computationally very difficult, because the number of all possible dependency rules increases exponentially with the number of attributes. As a solution, different kinds of restrictions and heuristics have been applied, but a general, scalable search method has been missing. In this paper, we introduce an efficient algorithm, called Kingfisher, for searching for the best non-redundant dependency rules with statistical significance measures. The rules can express either positive or negative dependencies between a set of positive attributes and a single consequent attribute. The algorithm itself is independent from the used goodness measure, but we concentrate on Fisher's exact test and the χ^2-measure. The algorithm is based on an application of the branch-and-bound search strategy, supplemented by several pruning properties. Especially, we prove a new lower bound for Fisher's p and introduce a new effective pruning principle. According to our experiments on classical benchmark data, the algorithm is well scalable and can efficiently handle even dense and high-dimensional data sets. An interesting observation was that Fisher's exact test did not only produce more reliable rules than the χ^2-measure, but it also performed the search much faster.

Keywords Dependency rule · Negative dependence · Statistical significance · Fisher's exact test · χ^2-measure · Kingfisher algorithm · Rule discovery

W. Hämäläinen (✉)
Department of Biosciences, University of Eastern Finland, Joensuu, Finland
e-mail: whamalai@cs.helsinki.fi

1 Introduction

Dependency analysis is a fundamental problem in all empirical science. A common form of the problem is to find dependency rules between categorical attributes. Such rules $X \rightarrow A$ or $X \rightarrow \neg A$ can explain, which combinations of factors can cause or prevent other factors. Statistical dependencies do not necessarily express causal relationships, but still they give an important insight on the regularities in data.

In medical science, for example, an important task is to search for dependencies between gene alleles, environmental factors and diseases. The interesting dependencies do not necessarily have to be strong or frequent, but instead they should be statistically valid, i.e., genuine dependencies which hold also in future data. Therefore, the significance of discoveries is tested with Fisher's exact test or—when it is infeasible—with the χ^2-test.

When the number of attributes is relatively small, it is possible to implement a brute-force search, which tests all possible dependency rules. However, the problem becomes soon intractable when the number of attributes increases. For example, if we have 9 binary attributes, there are about million possible dependencies to check. If the number of attributes is 15, there are already 15×10^9 possible dependencies. This exponential explosion is not surprising, since even a simpler problem—finding the best classification rule containing only positive attributes—is already NP-hard [19].

The traditional rule discovery algorithms, like Apriori [3], do not work either, because they are based on the frequency-based pruning. In the worst case, they find only independencies or spurious rules, while all significant dependencies are missed [23,24].

In addition, statistical dependence is not an anti-monotonic property, which means that an attribute set can produce a strong dependency rule, even if all of its subsets contained only independencies. The same holds for the statistical significance, which means that a totally different kind of a search strategy is needed.

In this paper, we introduce a new algorithm called *Kingfisher*, which is able to search for the best non-redundant dependency rules efficiently. We focus on Fisher's exact test (p-value) and the χ^2-measure as search criteria, but the same algorithm is applicable to other goodness measures of statistical dependence, like mutual information or leverage (which actually measures only the strength of the dependency), as well. The requirement for the non-redundancy means that a more specific rule is pruned out, if a better but more general rule has already been found. This is an essential property, when statistical dependencies are analyzed, because redundant rules can obscure the real dependencies and make the rules less accurate.

Before Kingfisher, there has been no algorithms for searching for even classification rules (with a fixed consequence) using Fisher's exact test or for searching for general (positive and negative) dependency rules with the χ^2-measure. The nearest existing solutions have searched for only classification rules with the χ^2-measure or general (positive) dependency rules with other measures, like leverage. Compared to them, Kingfisher is significantly better scalable, without their restrictions like minimum frequency thresholds, restricted rule lengths or fixed consequent attributes. In addition, the algorithm searches for both positive and negative dependency rules, which is a more demanding task than the search for only positive dependencies.

The algorithm is based on the common branch-and-bound strategy, supplemented by several pruning properties. The most important new inventions are a tight lower bound for Fisher's p and a new effective pruning principle called *Lapis Philosophorum*.

According to our experiments, Kingfisher is extremely well scalable and can handle even dense and high-dimensional data sets efficiently. As expected, Fisher's exact test produces

Table 1 Basic notations

Notation	Meaning
$A, B, C, \ldots$	Binary attributes
$a, b, c, \ldots \in \{0, 1\}$	Attribute values
$R = \{A_1, \ldots, A_k\}$	Set of all attributes
$\|R\| = k$	Number of attributes
$Dom(R) = \{0, 1\}^k$	Attribute space
$X, Y, Z, Q \subseteq R$	Attribute sets
$Dom(X) = \{0, 1\}^l$	Domain of X, $\|X\| = l$
$(X = \overline{x}) \equiv \{(A_1 = a_1), \ldots, (A_l = a_l)\}$	Instance of $X = \{A_1, \ldots, A_l\}$
$(X = 1) \equiv (A_1 = 1, \ldots, A_l = 1)$	Shorthand notations, when
$(X = 0) \equiv (A_1 = 0 \vee \ldots \vee A_l = 0)$	$X = \{A_1, \ldots, A_l\}$
$t \in Dom(R)$	Row (transaction)
$r = \{t_1, \ldots, t_n\}$	Data set
$\|r\| = n$	Number of rows in r
$m(X = \overline{x})$	Absolute frequency
$P(X = \overline{x}) = \frac{m(X=\overline{x})}{n}$	Relative frequency
$P(A = a \| X = \overline{x}) = \frac{P(X=\overline{x}, A=a)}{P(X=\overline{x})}$	Confidence of rule
	$X = \overline{x} \rightarrow A = a$
$\gamma(X = \overline{x}, A = a) = \frac{P(X=\overline{x}, A=a)}{P(X=\overline{x})P(A=a)}$	Lift of rule
$\delta(X = \overline{x}, A = a) = P(X = \overline{x} A = a) - P(X = \overline{x})P(A = a)$	Leverage of rule

more reliable results than the χ^2-measure, in the sense that the discovered dependencies hold well also in future data. However, a surprising result was that searching with Fisher's exact test is also remarkably more efficient than with the χ^2-measure. This is an important result, because scientists prefer to use Fisher's exact test, whenever it is feasible.

The rest of the paper is organized as follows: In Sect. 2, we define all basic concepts and the problem in detail. In Sect. 3, we review the existing solutions and other related research. The new theoretical results are introduced in Sect. 4. Section 5 represent the Kingfisher algorithm along with the complexity analysis. Experimental results are reported in Sect. 6, and the final conclusions are drawn in Sect. 7.

The paper extends the author's ICDM'10 conference paper [11].

2 Problem statement

In this section, we formalize the idea of dependency rules, introduce Fisher's exact test and the χ^2-measure, analyze the problem of redundancy and define the search problem. The basic notations are introduced in Table 1.

2.1 Dependency rules

Dependency rules are rules of the form $X = \overline{x} \rightarrow A = a$, where antecedent $X = \overline{x}$ is a conjunction of binary attributes or their negations, consequence $A = a$ consists of a single binary attribute or its negation, and the antecedent and the consequence are statistically dependent. Formally, dependency rules are defined as follows:

Definition 1 (*Dependency rule*) Let R be a set of binary attributes, $X \subseteq R$ and $A \in R \setminus X$. Rule $X = \overline{x} \rightarrow A = a$, where $|X| = l$, $\overline{x} \in \{0, 1\}^l$ and $a \in \{0, 1\}$, is a dependency rule, if $P(X = \overline{x} A = a) \neq P(X = \overline{x})P(A = a)$.

The dependency is (i) positive, if $P(X = \overline{x} A = a) > P(X = \overline{x})P(A = a)$ and (ii) negative, if $P(X = \overline{x} A = a) < P(X = \overline{x})P(A = a)$. Otherwise, the rule is called an independence rule.

In this paper, we concentrate on rules, where all attributes in X are positive. For values $A = 1$ and $A = 0$, we use notations A and $\neg A$. Now, dependency rules can be expressed simply by $X \rightarrow A$, $X \rightarrow \neg A$, $\neg X \rightarrow A$ and $\neg X \rightarrow \neg A$. We notice that negative dependence between X and A is the same as positive dependence between X and $\neg A$. Therefore, it is enough to consider only rules which express positive dependencies.

The strength of the dependency is usually measured by *leverage*

$$\delta(X, A = a) = P(XA = a) - P(X)P(A = a)$$

or lift

$$\gamma(X, A = a) = \frac{P(X, A = a)}{P(X)P(A = a)}.$$

Leverage is used in the *variable-based semantics* (common in statistics), where one is interested in the dependency between variables X and A, while lift is used in the *value-based semantics* (common in the association rule research), where one is interested in the dependency between certain values $X = 1$ and $A = a$ [6]. In this paper, we concentrate on the variable-based semantics, which typically produces more reliable results, although the same search algorithm can be applied to both semantics, given suitable goodness measures.

Finally, we note that generally dependency rules are not the same as *association rules* [2], because the latter do not necessarily express any statistical dependence. Instead, association rules express relations between frequently occurring attribute sets. In the traditional *frequency-confidence* framework, association rule $X \rightarrow A$ merely expresses that frequency $P(XA)$ and confidence $P(A|X)$ are larger than some user-defined thresholds. Therefore, it is quite possible that association rule $X \rightarrow A$ expresses actually negative dependence or independence between X and A. *Negative association rules* [28] are defined similarly, but now either the antecedent or consequent part is negated, i.e., rules are of the form $X \rightarrow \neg A$, $\neg X \rightarrow A$ or $\neg X \rightarrow \neg A$. We note that by definition, these rules do not necessarily express negative dependencies (or any statistical dependence) between X and A, but only state that $P(X = xA = a)$, $x, a \in \{0, 1\}$, is sufficiently frequent and confidence $P(A = a|X = x)$ is sufficiently large. In fact, $\neg X \rightarrow \neg A$ expresses positive dependency always when $X \rightarrow A$ is a positive dependency rule.

2.2 Statistical significance measures

Finding strong dependencies does not yet guarantee that the results would be reliable. It is quite possible that the observed dependency is spurious, i.e., occurred just by chance in the studied data set but does not hold in future data. Therefore, we should estimate the statistical significance of the dependency to get guarantees that the dependency is genuine.

Generally, the statistical significance of a positive dependency between X and $A = a$ is estimated by calculating the probability p that the observed or a stronger dependency would have occurred by chance, if X and A were actually independent. The p-value can be calculated from a cumulative hypergeometric distribution using Fisher's exact test:

$$p_F(X \to A = a) = \sum_{i=0}^{J} \frac{\binom{m(X)}{m(XA = a) + i}\binom{m(\neg X)}{m(\neg XA \neq a) + i}}{\binom{n}{m(A = a)}},$$

where $J = \min\{m(XA \neq a), m(\neg XA = a)\}$.

A commonly used alternative is to estimate the p-value from the χ^2-test or use the χ^2-measure as such to rank the dependency rules. The χ^2-measure is defined as

$$\begin{aligned}\chi^2(X \to A = a) &= \frac{n(P(XA = a) - P(X)P(A = a))^2}{P(X)P(\neg X)P(A = a)P(A \neq a)} \\ &= \frac{n\delta(X, A = a)^2}{P(X)P(\neg X)P(A = a)P(A \neq a)}.\end{aligned}$$

As a classical rule of thumb [9], the χ^2-measure should not be used, if any of the expected frequencies $nP(X = x)P(A = a),\ x \in \{0, 1\},\ a \in \{0, 1\}$ is less than five. However, this condition does not yet guarantee accurate results, if the underlying distribution is skewed [31,4]. The accuracy may be slightly improved by using a *continuity correction*, i.e., subtracting 0.5 from the difference between observed and expected frequencies, $n\delta(X, A = a)$ (or $\frac{0.5}{n}$ from $\delta(X, A = a)$) before applying the test [31].

In this paper, both p_F and χ^2 are used only as goodness measures for ranking the dependency rules. We do not try to solve the multiple testing problem and decide at which level the rule can be called significant in a statistical sense. Solutions to this problem can be found in [24,25].

The following search algorithm is not restricted to these two measures, but any measure M for the statistical significance of dependence could be used, as well. We assume that the significance is measured in the variable-based semantics and therefore $M(X \to A) = M(\neg X \to \neg A)$ and $M(X \to \neg A) = M(\neg X \to A)$. With this assumption, it is enough to search for only rules of the form $X \to A$ or $X \to \neg A$. If this assumption does not hold (typical to value-based measures, like the z-score), then the algorithm can be used to search for only positive rules.

2.3 Redundant rules

An important task in all rule discovery is to identify *redundant* rules, which add no new information to the remaining rules. When the objective is to find statistical dependencies, independent attributes do not add any new information on the dependency. In fact, they can rather blur the interpretation of dependencies. For example, if people suffering for disease A are more likely to have a gene allele B than healthy people, then there is a positive dependency between A and B. In addition, there are a lot of gene alleles which have no effect on disease A or the occurrence of B. For any such independent allele C, we can construct a dependency rule $BC \to A$, which is equally strong to the original rule $B \to A$, but the conclusions could be quite different. Now, one could assume that the alleles B and C together cause the disease and consider only people with both alleles B and C as a potential risk group. Even a more serious error could happen, if C actually prevented the disease (A and C were negatively dependent), but now the dependency would also be weaker than in the original rule.

Generally, adding new attributes to the antecedent of a dependency rule can 1) have no effect, 2) weaken the dependency or 3) strengthen the dependency. When the task is to search for only positive dependencies (concerning either A or $\neg A$), the first two cases can only add redundancy. The third case is more difficult to judge, because now the extra attributes make the dependency stronger, but in the same time, the rule usually becomes less frequent. If the frequency is very small, the observed improvement can well be due to chance. For example, if BC occurs on just one row—where A is also true—then the dependency is the strongest possible, but we have no guarantees that the same dependency would hold in future data. Therefore, we need some test to decide, whether the extra attributes made the rule better or worse.

In this paper, we generalize the classical definition of redundancy (e.g., [1]) to a general goodness measure M:

Definition 2 (*Redundancy*) Let M be a goodness measure. Rule $X \rightarrow A = a$ is redundant, if there exists rule $Y \rightarrow A = a$ such that $Y \subsetneq X$ and $M(Y \rightarrow A = a)$ is equally good to or better than $M(X \rightarrow A = a)$. Otherwise, rule $X \rightarrow A = a$ is non-redundant.

2.4 Search problem

The search problem for the most significant, non-redundant dependency rules can occur in two forms: In the *enumeration problem*, the task is to search for all non-redundant rules whose goodness value M does not exceed some predefined threshold (min_M or max_M). In the *optimization problem*, the task is to search for the K best non-redundant dependency rules, given the desired number of rules K. Typically, the enumeration problem produces a larger number of rules, but with suitable parameter settings the results are identical.

In practice, it is often a good compromise to give both the maximal number of rules to be searched for as well as some predefined threshold. Typically, the user is interested in just 50, 100 or at most 1000 best rules. However, it is good to have also a threshold to guarantee a certain level of significance and improve the search in the beginning, when less than K rules have been found so far. After K sufficiently significant rules have been found, the threshold will be updated automatically, and the initial threshold has no effect on the final results. The initial threshold affects the results only, if it is so demanding that K sufficiently good non-redundant rules could not be found in the data. This situation can also occur with permissive thresholds, if the data does not contain significant dependencies.

Finally, we note that our objective is to find globally optimal results, and therefore, the search algorithm should be efficient without any suboptimal heuristics like minimum frequency thresholds or maximal rule lengths (numbers of attributes in the rule).

3 Related research

The problem of searching for the most significant, non-redundant, positive or negative dependency rules with a statistical goodness measure can be divided into three subproblems. First, each rule $X \rightarrow A = a$ should express a statistical dependency, which means that the lift should deviate from one. Second, the significance of the rule should be measured by a statistical goodness measure, which reflects the probability that such a strong dependency had occurred by chance, if X and A were independent. Third, the rules should be non-redundant, which means that a more specific rule is pruned out, if there already exists a better but more general rule.

Most of the previous research has tackled only some of these subproblems, and only a few of the existing algorithms have searched for both positive and negative dependencies. As far as we know, no scalable solutions are known to the general problem.

Let us first consider the search for positive rules. The simplest way to solve the problem is to search for association rules [2] with a sufficiently low minimum frequency threshold and then select the most significant non-redundant dependency rules in the post-processing phase. In this way, it is possible to generate also certain kinds of negative dependency rules, allowing only frequent sets in the rule antecedent and consequence [32]. The problem of this approach is that association rule mining requires relative large minimum frequency thresholds to be feasible. With large-dimensional data sets, the threshold can be 0.60–0.80, which means that the dependency should occur on at least 60–80% of the rows. Such rules have always low lift values, and usually, the strongest and most significant dependencies are missed. In addition, a lot of futile work is done, because most of the frequent rules do not express any significant dependencies or are redundant specializations of already discovered rules.

A better approach is to search directly for dependency rules, but the problem is still complex. Therefore, a common solution is to consider only some special cases and use some minimum frequency thresholds, restrict the maximal rule length, fix the consequent attribute or use some other restrictions to prune out "uninteresting" rules.

The most common approach is to search for classification rules $X \rightarrow C$ with a fixed consequent attribute C. Morishita and Sese [19] as well as Nijssen and Kok [21] searched for classification rules with the χ^2-measure. Both of these approaches utilized the convexity of the χ^2-function and determined a minimum frequency threshold, which guarantees a certain minimum χ^2-value. It was noted that the approach is quite inefficient, because the resulting minimum frequency thresholds were too low. Nijssen et al. [20] developed a more efficient solution for searching for the best classification rule with a closed set in the rule antecedent. The problem of this approach is that the closed sets can contain redundant attributes (which are conditionally independent from C given the other attributes) and the rule can be suboptimal in future data.

Liu et al. [18] used also the χ^2-measure, in addition to minimum frequency thresholds and some other pruning heuristics, to test the goodness of rule $X \rightarrow C$ and whether the *productivity* (improvement in the confidence) of rule $X \rightarrow C$ with respect to its immediate generalizations ($Y \rightarrow C,\ X = YB$ for some attribute B) was significant. It is customary to test the productivity only against the immediate generalizations, because checking all $2^{|X|}$ subrules of the form $Y \rightarrow A,\ Y \subsetneq X$ is inefficient. Unfortunately, this also means that some rules may appear as productive, even if they are weaker than some more general rules.

Li [17] introduced an algorithm for searching for only non-redundant classification rules using different goodness measures, including leverage and lift. No minimum frequency thresholds were used, but instead it was required that $P(X|A)$ was larger than some predefined threshold.

Searching for general dependency rules with any consequent attribute is a more difficult and less studied problem. Xiong et al. [30] concentrated in positive dependencies between just two attributes using an upper bound for Pearson's correlation coefficient as a measure function. Koh et al. [16] searched for "sporadic rules", i.e., positive dependency rules concerning rare consequent attributes. The minimum frequency thresholds for antecedent sets, given consequent attributes, were determined by Fisher's exact test. However, the algorithm was not complete, because it was based on the (faulty) assumption that statistical significance would be an anti-monotonic property, i.e., that if $Z \rightarrow A$ is significant, then all $X \rightarrow A,\ X \subseteq Z$ are significant, or, equivalently, if $X \rightarrow A$ is insignificant, then all $Z \rightarrow A,\ Z \supseteq X$ are insignificant. Based on this assumption, new antecedent sets Z were generated for a consequence

A only if there were subsets X and Y, $Z = X \cup Y$, $|X| = |Y| = |Z| - 1$, such that $X \rightarrow A$ and $Y \rightarrow A$ were sufficiently significant. Therefore, all statistically significant dependency rules having any insignificant generalizations were missed.

In our previous research [10,13,14], we searched for non-redundant, positive dependency rules with the z-score. No minimum frequency thresholds were required in these algorithms, but the notion of redundancy was more restrictive than the classical definition.

Webb's *MagnumOpus* software [26,27] is able to search for general dependency rules $X \rightarrow A$ with different goodness measures, including leverage and lift. In addition, it is possible to test that the rules are significantly productive. In practice, MagnumOpus tests the improvement of the confidence of rule $X \rightarrow A$ with respect to its immediate generalizations ($Y \rightarrow A$, $X = YB$ for some B) with Fisher's exact test. Otherwise, redundant rules are not pruned out and the discovered rules are not necessarily the most significant. The algorithm is quite well scalable, and if only the K best rules are searched for with the leverage, it is possible to perform the search without any minimum frequency requirement. However, we note that the leverage itself favours rules with a frequent consequent, and therefore, some significant and productive rules can be missed.

All of the previously mentioned algorithms search for only positive dependencies, but there are a couple of approaches which have searched for also negative dependencies. The algorithm by Antonie and Zaïane [5] searched for negative dependency rules using a minimum frequency threshold, minimum confidence threshold and Pearson's correlation coefficient as a goodness measure. Since Pearson's correlation coefficient for binary attributes is the same as $\sqrt{\chi^2/n}$, the algorithm should be able to search for dependencies with the χ^2-measure, as well.

Thiruvady and Webb [22] introduced an algorithm for both positive and negative dependency rules of the form $(X = x) \rightarrow (A = a)$, $x, a \in \{0, 1\}$, using leverage as a goodness measure. No minimum frequency thresholds were needed, but the rule length was restricted.

In addition, there are a couple of algorithms based on the assumption that statistical dependence or significance would be an anti-monotonic property. In these algorithms, the basic idea is to generate new attribute sets Z by combining two simpler sets, X and Y, $Z = X \cup Y$, satisfying certain extra requirements. For example, Wu et al. [29] required that sets X and Y were sufficiently frequent and both of them contained at least one sufficiently strong dependency, evaluated by a heuristic criterion. Negative dependency rules were generated from infrequent sets Z and further pruned by their frequency, leverage and certainty factor. Koh and Pears [15] did not require any minimum frequency, but instead, both X and Y (where $|X| = |Y| = |Z| - 1$) should contain a sufficiently significant positive dependency, evaluated by Fisher's exact test. Either positive or negative dependency rules were generated from sets Z based on Fisher's exact test and minimum confidence requirement. In both algorithms, all positive dependency rules satisfying the extra requirements could be produced correctly, if statistical dependence would be an anti-monotonic property. However, only certain special cases of negative dependency rules could be found, even if the anti-monotonicity assumption were true. Since statistical dependence and significance are not anti-monotonic properties, these kind of algorithms are incomplete (even if no minimum frequency thresholds were used) and there are no guarantees that the strongest or most significant dependency rules would be discovered.

4 Pruning the search space

In this section, we introduce the theoretical basis for the search algorithm. We describe the basic branch-and-bound search, give required bounds for the goodness measures p_F and χ^2 and introduce additional pruning properties.

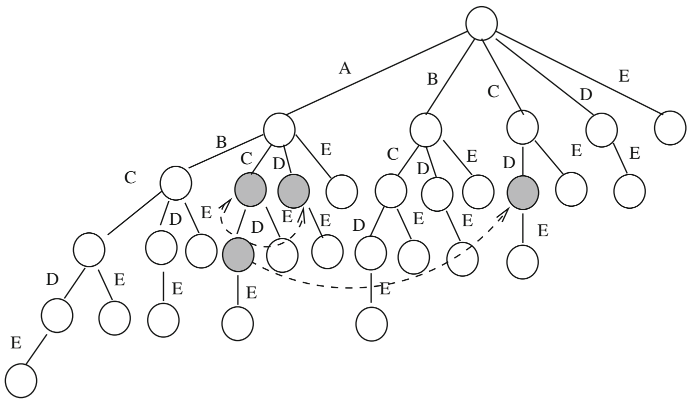

Fig. 1 A complete enumeration tree on attributes $A, \ldots, E$. A node and its parent nodes are emphasized

4.1 Branch-and-bound search

The whole search space can be represented by an enumeration tree (Fig. 1). Given a set of attributes R, the enumeration tree lists all possible attribute sets $Z \in \mathcal{P}(R)$. In each set Z, rules $X_i \rightarrow A_i = a_i,\ Z = X_i A_i,\ a_i \in \{0, 1\}$ can be generated. We recall that this form of rules covers dependency rules $X_i \rightarrow A_i,\ X_i \rightarrow \neg A_i,\ \neg X_i \rightarrow A_i$ and $\neg X_i \rightarrow \neg A_i$. If any of these rules $X_i \rightarrow A_i = a_i$ has a sufficiently good M-value and there are no better more general rules $Y \rightarrow A_i = a_i,\ Y \subsetneq X_i$, the rule is non-redundant and significant. If the task is to search for only the best K rules, the threshold value ($\max_M$ or $\min_M$) is updated always, when a new Kth best rule with a better M-value is found. On the other hand, if the task is to search for all non-redundant dependency rules, the user-defined threshold $\max_M$ or $\min_M$ remains constant during the whole search.

The basic idea of the branch-and-bound search is to traverse the enumeration tree and in each node, estimate the goodness of rules, which can be constructed in the underlying subtree or whose antecedent occurs in the subtree. Each node, representing set Z, has a list of possible consequences $A_i = a_i$, inherited from its parent nodes. For all possible consequences $A_i = a_i$, where $A_i \in Z$, we should estimate the maximal goodness of rules $X_i Q \rightarrow A_i = a_i$, where $Z = X_i A_i$ and $Q \subseteq R \setminus Z$ (possibly $Q = \emptyset$). If the value is too poor, then $A_i = a_i$ becomes an impossible consequence (can be removed from the list of possible consequences). On the other hand, if the bound is sufficiently good, we check also the goodness of rule $X_i \rightarrow A_i = a_i$. If this rule is sufficiently good (significant and non-redundant), the rule is stored into the collection of the best rules. On the other hand, if $A_i \notin Z$, then Z or any of its supersets $ZQ,\ A_i \notin Q$, can be an antecedent for the consequence $A_i = a_i$. Now, we should estimate the maximal goodness of rules $ZQ \rightarrow A_i = a_i$ for any $Q \subseteq R \setminus (Z \cup \{A_i\})$. If the value is too poor, then $A_i = a_i$ becomes an impossible consequence. If all consequences in the node become impossible, the whole node can be removed.

In the following subsection, we will introduce the required lower and upper bounds, when the goodness measure M is either Fisher's p_F or the χ^2-measure. However, the basic branch-and-bound search is quite inefficient as such. Therefore, we will introduce additional pruning properties, which can prune the search space remarkably.

The search order has also an important impact on the efficiency. In principle, the attributes can be in any order in the enumeration tree and the enumeration tree can be traversed either in a breadth-first or depth-first manner, from right to left or left to right. In practice, the best approach is to order the attributes in an ascending order by their frequency, i.e., in Fig. 1, $P(A) \leq \ldots \leq P(E)$. Now, the largest subtree (under A) is likely to contain the least frequent sets and many of the sets can be totally missing. The breadth-first search is also more efficient than the depth-first search, because we search for only non-redundant dependency rules. In the breadth-first search, the more general rules are checked before their specializations, and it is possible to prune out the specializations without checking, if we know that they would be redundant with respect to already discovered rules. Finally, we prefer to perform the search from left to right, because it enables an efficient extra pruning property called *Lapis Philosophorum*, which is described in a subsequent subsection.

4.2 Required upper and lower bounds

To perform the search, we need a bound (best possible value) for rules of the form $M(Z \setminus \{A_i\}Q \rightarrow A_i = a_i)$ (when $A_i \in Z$) and $M(ZQ \rightarrow A_i = a_i)$ (when $A_i \notin Z$) for any extension set $Q \subseteq R \setminus (Z \cup \{A_i\})$, given the information available in the node corresponding to set Z. If the goodness measure M is increasing (like χ^2), an upper bound is needed, while for decreasing measures (like p_F), lower bounds are used.

In practice, we need bounds for three different cases, given set Z: First, if $A_i \notin Z$ and $m(Z) \geq m(A_i = a_i)$, it is possible to find a rule $ZQ \rightarrow A_i = a_i$ expressing maximal dependence between ZQ and $A_i = a_i$. Since the statistical dependence is maximal, when ZQ and $A_i = a_i$ are identical, the bound is achieved when $m(ZQA_i = a_i) = m(ZQ) = m(A_i = a_i)$ and the consequence frequency $m(A_i = a_i)$ alone determines the bound. This case occurs also in the beginning, when sets of single attributes are checked and $Z = \emptyset$. In this phase, we may already find out that some consequence $A_i = a_i$ is impossible in all rules and attribute A_i can occur only in the rule antecedent.

Second, if $A_i \notin Z$ but $m(Z) < m(A_i = a_i)$, then the maximal dependence is no more possible to achieve in any rules $ZQ \rightarrow A_i = a_i$. Since frequency $m(ZA_i = a_i)$ is not yet known, the bound is determined based on frequencies $m(Z)$ and $m(A_i = a_i)$. This case occurs also in the beginning, when rules of the form $A_j \rightarrow A_i = a_i$ are checked and $m(A_j) < m(A_i = a_i)$. The resulting bound is tighter than in the first case, and it can be possible to eliminate consequence $A_i = a_i$ from the node representing set A_j, even if $A_i = a_i$ is possible in other nodes. Later, we show that with measures p_F and χ^2 (as well as with all common statistical significance measures), this bound is achieved, when $m(ZQA_i = a_i) = m(ZQ)$ and thus $P(A_i = a_i|ZQ) = 1.0$.

Third, if $A_i \in Z$, then we already know frequencies $m(Z) = m(X_iA_i)$, $m(Z \setminus \{A_i\}) = m(X_i)$ and $m(A_i = a_i)$, where $Z = X_iA_i$. From these, we also get $m(X_iA_i = a_i)$ (i.e., $m(X_i\neg A_i)$, if $a_i = 0$). Therefore, we can define a tighter bound than in the second case. It turns out that with p_F and χ^2, this bound is achieved, when $m(X_iQA_i = a_i) = m(X_iQ)$ and thus $P(A_i = a_i|X_iQ) = 1.0$.

In the following, we will notate the lower bounds for these three different cases by *lb1*, *lb2* and *lb3* and corresponding upper bounds by *ub1*, *ub2* and *ub3*. Table 2 gives *lb1*, *lb2* and *lb3* for p_F and *ub1*, *ub2* and *ub3* for the χ^2-measure. The upper bounds for the χ^2-measure are well-known (e.g., [19]), but the lower bounds for p_F are new results, introduced in the conference version of this paper [11]. The proofs for the lower bounds are given in A.

We note that in the case of the χ^2-measure, the first upper bound *ub1* is uninformative, because any consequence $A_i = a_i$ can gain the maximal possible χ^2-value. However, the first lower bound *lb1* for p_F contains already useful information, which can be used to eliminate some consequences $A_i = a_i$ before any other frequencies except $m(A_i = a_i)$ are checked.

Table 2 Lower bounds *lb1*, *lb2* and *lb3* for p_F and upper bounds *ub1*, *ub2* and *ub3* for χ^2

Lower bounds for p_F	Upper bounds for χ^2
$lb1 = \frac{m(A)!m(\neg A)!}{n!}$	$ub1 = n$
$lb2 = \frac{m(\neg X)!m(A=a)!}{n!(m(A=a)-m(X))!}$	$ub2 = \frac{nm(X)m(A\neq a)}{m(\neg X)m(A=a)}$
$lb3 = \frac{m(A)!m(\neg A)!(n-m(XA=a))!}{n!m(A\neq a)!m(\neg XA=a)!}$	$ub3 = \frac{nm(XA=a)m(A\neq a)}{(n-m(XA=a))m(A=a)}$

We also observe that *lb1* obtains its best (minimal) value, when $m(A_i) = \frac{n}{2}$, and the further $m(A_i)$ diverges from $\frac{n}{2}$, the more *lb1* deteriorates. In practice, this means that if $m(A_i)$ or $m(\neg A_i)$ is very low, *lb1* can be so large that no rule with consequence A or $\neg A$ is significant and both of them can be marked as impossible consequences in the whole enumeration tree. However, attribute A_i may still occur in the antecedent part of significant rules. The following result shows that with the extra condition $m(A_i) \leq \frac{n}{2}$, A_i cannot occur even in the antecedent and it can be totally pruned out.

Observation 1 If $m(A) \leq \frac{n}{2}$, then for all $Q \subseteq R \setminus \{A\}$, $B \in R \setminus (Q \cup \{A\})$ holds $p_F(QA \rightarrow B=b) = p_F(B=b \rightarrow QA) \geq \frac{m(QA)!m(\neg(QA))!}{n!} \geq \frac{m(A)!m(\neg A)!}{n!}$.

Corollary 1 *If* $m(A) \leq \frac{n}{2}$ *and* $lb1 > max_M$, *then* A *cannot occur in any significant rule.*

This result can be used to determine a minimum frequency threshold $min_{fr} \leq 0.5$ such that no rule containing A, $P(A) < min_{fr}$, can be significant. We note that p_F is not the only measure, which allows this extra pruning, but corresponding results can be shown for some other measures, like mutual information (when $ub1 < min_M$), as well.

4.3 Pruning by minimality

Rule $X \rightarrow A=a$ with $P(A=a|X)=1$ is called *minimal*, because none of its specializations $XQ \rightarrow A=a$ can achieve a better M-value. However, this kind of minimal rules can be used to prune out other rules as insignificant or redundant, too. The following observation extends a well-known result (e.g., [17]) for consequence $A=a$, $a \in \{0,1\}$, with a general statistical goodness measure M. We note that this result extends also our previous work [11], where the result was shown only for p_F.

Observation 2 Let M be a statistical goodness measure, whose value $M(X \rightarrow A=a)$ for any rule $X \rightarrow A=a$ depends on only $m(X)$, $m(XA=a)$, $m(A=a)$ and n.

If $P(A=a|X)=1$, then all rules of the form $XQA \rightarrow B=b$, where $B \in R\setminus(X\cup\{A\})$, $Q \subseteq R \setminus (X \cup \{A,B\})$ and $b \in \{0,1\}$ are either insignificant or redundant.

Proof First, we notice that M can be expressed by function $f : \mathbb{N}^4 \rightarrow \mathbb{R}$ such that $M(X \rightarrow A=a) = f(m(X), m(XA=a), m(A=a), n)$.

(1) If $P(A|X)=1$, $m(XQA)=m(XQ)$ and $m(XQAB=b)=m(XQB=b)$ for any $B \in R\setminus(X\cup\{A\})$, $Q \subseteq R\setminus(X\cup\{A,B\})$, and $b \in \{0,1\}$. Therefore, $M(XQA \rightarrow B=b) = f(m(XQA), m(XQAB=b), m(B=b), n) = f(m(XQ), m(XQB=b), m(B=b), n) = M(XQ \rightarrow B=b)$ and rule $XQA \rightarrow B=b$ is redundant.

(2) If $P(\neg A|X)=1$, $m(XA)=0$ and also $m(XQA)=0$ and $m(XQAB=b)=0$. Therefore, rule $XQA \rightarrow B=b$ does not express any dependence ($P(XQAB=b) = 0 = P(XQA)P(B=b)$). □

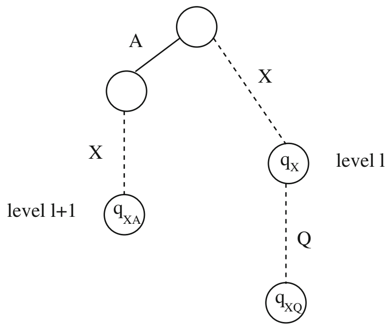

Fig. 2 Lapis Philosophorum principle. If consequence $A = a$ is impossible in node q_{XA}, then it is impossible in nodes q_X and q_{XQ}, $|X| = l$. Dash lines represent paths

4.4 The Lapis Philosophorum principle

The basic branch-and-bound search prunes possible consequences only in the subtrees of a given node (like in [27]). However, it is also possible to prune consequences in the parent nodes and propagate the results to other subtrees. This requires that the node is processed before any children are generated for its parents, except the immediate parent. Figure 1 shows an example. When we proceed level by level, from left to right, set ACD is processed before any children are created for its parent sets AD and CD. If set ACD is now removed (e.g., has a zero frequency, which means that rule $AD \rightarrow \neg C$ was minimal), then C and $\neg C$ become impossible consequences in the node for AD and its subtrees. Similarly, if we find in the node for ACD that no rule $QCD \rightarrow A$ (for any $Q \subseteq R \setminus \{A, C, D\}$) could be significant and non-redundant, then A can be marked as an impossible consequence in the node for CD and its subtrees. This simple principle performs so effective pruning that it is called *Lapis Philosophorum*, the legendary Philosopher's stone.

Principle 1 (*Lapis Philosophorum*) Let M be a decreasing measure, where the goodness of rules increases when the corresponding M-value decreases.

Let q_{XA} be a node corresponding to set XA, and q_X a node corresponding to set X as shown in Fig. 2. If any of the following conditions holds in node q_{XA}, consequence $A = a$ can be marked as impossible in node q_X and all nodes q_{XQ} in its subtrees:

(i) Node q_{XA} does not exist (i.e., no consequence was possible in set XA or its supersets).
(ii) Rule $X \rightarrow A = a$ is minimal.
(iii) Rule $XQ \rightarrow A = a$ would be insignificant or redundant, i.e., for a decreasing goodness measure M lower bound $lb = LB(M(XQ \rightarrow A = a)) > max_M$ or $lb \geq \min\{M(Y \rightarrow A = a) | Y \subsetneq X\}$.

The principle is based on the fact that all supersets XQA lie in the subtree under q_{XA}, and q_{XQ} is needed only as a parent node for rules $XQ \rightarrow A = a$. The same principle can be applied to any goodness measure M, but for increasing goodness measures (where high values are better), the lower bound should be replaced by an upper bound, threshold max_M by min_M, and the inequality signs should be reversed.

5 Algorithm

In this section, we represent the Kingfisher algorithm, which implements an efficient search for the best non-redundant dependency rules. The algorithm is represented by a detailed pseudocode and illustrated by an example. In addition, we discuss central implementation issues and analyze the worst-case time and space complexity.

5.1 Pseudocode for the Kingfisher algorithm

The Kingfisher algorithm is given in Algorithms 5.1, 5.2, 5.3, 5.4 and 5.5. For simplicity, we represent the algorithm for a decreasing goodness measure M (like p_F), but increasing goodness measures can be used as well, with the previously mentioned adjustments. For p_F, the required lower bounds are given in Table 2.

The node corresponding to attribute set X is denoted by $Node(X)$. In each node $v = Node(X)$, we use the following fields:

- $v.set$ set X; in practice, it is enough to store just the last attribute in the path from the root to node v.
- $v.children$ table of pointers to v's child nodes. The size of the table is denoted by $|v.children|$.
- $v.ppossible$ and $v.npossible$ bit-vectors, whose jth bits indicate whether consequence A_j or $\neg A_j$ is a possible consequence in node v or its descendants. For simplicity, we assume that both vectors have $|R|$ bits.
- $v.pbest$ and $v.nbest$ tables for the best M-values of rules $Y \rightarrow A_j$ and $Y \rightarrow \neg A_j$, $Y \subseteq X$, $A_j \in X$. For simplicity, we assume that both tables have $|R|$ elements and index j corresponds to consequence A_j. In practice, the tables can be implemented more compactly by tables of $|X|$ elements.

Algorithm 5.1 Kingfisher(R, r, max_M, K)

```
Input: set of attributes $R$, data set $r$, initial threshold $max_M$, maximal number of best rules $K$
Output: the best $K$ non-redundant dependency rules for which $M \leq max_M$
Method:
1   determine $min_{fr}$ using Corollary 1
2   $t \leftarrow$check1sets($R$,$r$,$max_M$,$min_{fr}$)
3   $l \leftarrow 2$
4   while (number of ($l-1$)-sets $\geq l$)
    // check l-sets
5       for $i = 1$ to $|R|$
6           bfs($t.children[i]$, $l$, 0)
7       $l \leftarrow l + 1$
8   output the $K$ best rules
```

The main idea of the algorithm is the following:

1. Use Corollary 1 to determine the maximal absolute frequency value min_{fr} such that even the best possible rule $X \rightarrow A = a$ with $m(A = a) = m(X) = m(XA = a) < min_{fr}$ cannot be significant. Prune out all attributes which cannot occur in significant rules. (Algorithm 5.1, line 1 and Algorithm 5.2, lines 3–4.) This step is possible only with some goodness measures like p_F and mutual information, satisfying the conditions of Corollary 1.
2. Order the remaining attributes $A_i \in R$ into an ascending order by frequency and add them to the enumeration tree. For all A_i, use lower bounds *lb1* and *lb2* to determine possible consequences $A_j = a_j$ such that $XA_i \rightarrow A_j = a_j$ can be significant. The possible consequences are marked into tables $ppossible$ and $npossible$ in node

Algorithm 5.2 Check1sets(R, r, max_M, min_{fr})

Input: set of attributes R, data set r, thresholds max_M and min_{fr}
Output: root of an enumeration tree t containing the first level of nodes
Method:

```
1   for $\forall A_i \in R$
2       calculate absolute frequency $m(A_i)$
3       if $(m(A_i) < min_{fr})$
4           $R \leftarrow R \setminus \{A_i\}$
5   order $R$ into an ascending order by frequency
6   create root node $t$
7   for $\forall A_i \in R$
8       create node $v = Node(A_i)$
9       add $v$ to $t.children$
10      for $\forall A_j \in R$
    // initialize best-tables for all consequences $A_j$ and $\neg A_j$
11          $v.pbest[j] \leftarrow \max\{M(\cdot)\}$          // maximal possible $M$-value
12          $v.nbest[j] \leftarrow \max\{M(\cdot)\}$
    // is $A_j$ or $\neg A_j$ a possible consequence for set $XA_i$?
13          $v.ppossible[j] \leftarrow$possible($A_j$, 1, $s$, $A_i$)
14          $v.npossible[j] \leftarrow$possible($A_j$, 0, $s$, $A_i$)
15  return $t$
```

Algorithm 5.3 bfs(st, l, len)

Input: root of a subtree st, goal level l, path length len
Output: discovered new best rules at level l in subtree st are stored into collection $brules$
Method:

```
1   if $(len = l - 2)$
    // combine $(l-1)$-sets to create new $l$-sets
2       for $i = 1$ to $|st.children| - 1$
3           $Y \leftarrow st.children[i].set$
4           for $j = i + 1$ to $|st.children|$
5               $Z \leftarrow st.children[j].set$
6               $X \leftarrow Y \cup Z$
7               create node $child = Node(X)$
8               add $child$ to $st.children[i].children$
9               initialize $possible$- and $best$-tables
            // all possible-values are set to 1 and best-values to max$\{M(\cdot)\}$
10              if (checknode($child$)=0)
11                  delete $child$
                // use Lapis Philosophorum
12                  for $\forall$ parent nodes $v = Node(Y_m)$ where $(X = Y_m A_m)$
13                      $v.ppossible[m] \leftarrow 0$
14                      $v.npossible[m] \leftarrow 0$
15              if $(Node(Y).children = \emptyset)$
16                  delete node $Node(Y)$
        // $st$'s last child has never child nodes
17      delete $st.children[|st.children|]$
18  else for $i = 1$ to $|st.children|$
19      bfs($st.children[i]$, $l$, $len + 1$)
20  if $(|st.children| = 0)$          // if all $st$'s children were deleted
21      delete node $st$
```

$Node(A_i)$. If A_j is possible, then $Node(A_i).ppossible[j] = 1$, and if $\neg A_j$ is possible, then $Node(A_i).npossible[j] = 1$. (Algorithm 5.2, lines 5–14.)

3. Expand attribute sets as long as new non-redundant, significant rules can be found. (Algorithm 5.1, lines 4–7.)

Algorithm 5.4 Checknode(v_X)

Input: node $v_X = Node(X)$
Output: return 0, if node v_X can be removed, and 1 otherwise; discovered new best rules in v_X are stored into collection $brules$
Method:

```
1   $ismin \leftarrow 0$                                                   // no minimal rules, yet
2   for $\forall Y \subsetneq X$, where $|Y| = |X| - 1$                    // all parent sets
3       $par_Y \leftarrow$searchset($Y$)
4       if ($par_Y$ not found) return 0
5       updatetables($v_X$, $par_Y$)                                   // update possible- and best-tables
6       if (no possible consequences left) return 0
7   calculate $m(X)$
8   if (($m(X) = 0$) or ($\exists par_Y$ such that $m(Y) = m(X)$))
9       $ismin \leftarrow 1$                                               // minimal rule found
10  for $\forall A_i \in R$                                                // update possible consequences
11      $v_X.ppossible[i] \leftarrow (v_X.ppossible[i]$ & possible$(A_i, 1, v_X, X))$
12      $v_X.npossible[i] \leftarrow (v_X.npossible[i]$ & possible$(A_i, 0, v_X, X))$
13      if (($A_i \in X$) and ($v_X.ppossible[i]$))                        // check pos. rule
14          $val \leftarrow M(X \setminus \{A_i\} \rightarrow A_i)$
15          if (($val \le max_M$) and ($val < v_X.pbest[i]$))
16              add rule $X \rightarrow A_i$ to $brules$; $v_X.pbest[i] \leftarrow val$
17              update $max_M$                                         // nothing to do, until K rules found
18      if (($A_i \in X$) and ($v_X.npossible[i]$))                        // check neg. rule
19          $val \leftarrow M(X \setminus \{A_i\} \rightarrow \neg A_i)$
20          if (($val \le min_M$) and ($val < v_X.nbest[i]$))
21              add rule $X \rightarrow \neg A_i$ to $brules$; $v_X.nbest[i] \leftarrow val$
22              update $max_M$                                         // nothing to do, until K rules found
23  if ($ismin$)                                                           // pruning by minimality
24      for $\forall A_i \in R \setminus X$
25          $v_X.ppossible[i] \leftarrow 0$; $v_X.npossible[i] \leftarrow 0$
26  for $\forall par_m = Node(Y_m)$ where ($Y_m A_m = X$)
27      if (($P(A_m|Y_m) = 1$) or ($P(\neg A_m|Y_m) = 1$))
28          $v_X.ppossible[m] \leftarrow 0$; $v_X.npossible[m] \leftarrow 0$           // by minimality
29          $par_m.ppossible[m] \leftarrow 0$; $par_m.npossible[m] \leftarrow 0$       // by Lapis P.
30  if (no possible consequences left) return 0
31  return 1
```

– Create l-sets from $(l-1)$-sets. (Algorithm 5.3, lines 2–9.)
– For each l-set X, initialize possible consequences in $Node(X)$, given possible consequences in its parent nodes $Node(Y_m)$, where $X = Y_m A_m$ for some $A_m \in X$. Consequence $A_j = a_j$, $A_j \in R$, is possible in $Node(X)$ only if it is possible in all parent nodes $Node(Y_m)$. Initialize the best M-values for all consequences $A_j = a_j$, where $A_j \in X$, using the *best*-values in the parent nodes. (Algorithm 5.4, lines 2–5 and Algorithm 5.5, function **updatetables**.)
– Calculate frequency $m(X)$ and check if $P(A_m = a_m|Y_m) = 1$ for any parent set Y_m. Use lower bounds *lb1*, *lb2* and *lb3* to decide whether any rule $(XQ)\setminus\{A_j\} \rightarrow A_j = a_j$ can be a non-redundant significant rule. (Lower bounds *lb1* and *lb2* are used, when $A_j \notin X$, and *lb3*, when $A_j \in X$.) (Algorithm 5.4, lines 7–12 and Algorithm 5.5, function **possible**.) We note that in function **possible** (Algorithm 5.5), the frequency comparison does not prune out anything else than the lower bound comparison, if the threshold min_{fr} was determined using Corollary 1. However, this point allows the use of other minimum frequency thresholds, if desired. For example, a common requirement is that all statistically valid rules should occur on at least five rows of data.

Algorithm 5.5 Auxiliary functions

```
**updatetables**($s$, $v$)
// update possible- and best-tables in node $s$ given parent node $v$
    for $i = 1$ to $|R|$
        $s.ppossible[i] \leftarrow (v.ppossible[i]$ & $s.ppossible[i])$
        $s.npossible[i] \leftarrow (v.npossible[i]$ & $s.npossible[i])$
        $s.pbest[i] \leftarrow \min\{v.pbest[i], s.pbest[i]\}$
        $s.nbest[i] \leftarrow \min\{n.pbest[i], s.nbest[i]\}$

**possible**($A_j$, $a_j$, $v$, $X$)
// can rule $(XQ) \setminus \{A_j\} \rightarrow A_j = a_j$ be significant and non-redundant?
// lb1, lb2 and lb3 are implemented by LB1, LB2 and LB3
    if ((($A_j \notin X$) and ($m(X) < min_{fr}$)) or
        (($A_j \in X$) and ($m(X \setminus \{A_j\}A_j = a_j) < min_{fr}$)))
            return 0                                    // rule would be too infrequent
    if ($A_j \notin X$)
        if ($m(X) \leq m(A_j = a_j)$)
            $lb \leftarrow$ LB2($m(X), m(A_j = a_j)$)
        else $lb \leftarrow$ LB1($m(A_j = a_j)$)
    else $lb \leftarrow$ LB3($m(X), m(A_j = a_j), m(XA_j = a_j)$)
    if (($lb > max_M$) or (($a_j = 1$) and ($lb \geq v.pbest[j]$))
        or (($a_j = 0$) and ($lb \geq v.nbest[j]$)))
        return 0
    return 1

**searchset**($Y$)
    return $Node(Y)$
```

- If $A_j \in X$ and $A_j = a_j$ was possible, calculate $M(X \rightarrow A_j = a_j)$. If it is sufficiently good (among the best K rules and better than more general rules with consequence $A_j = a_j$), add it to the rule collection and update $Node(X).pbest[j]$ (if $a_j = 1$) or $Node(X).nbest[j]$ (if $a_j = 0$). Update also max_M, if K rules have been found. (Algorithm 5.4, lines 13–22.)
- If minimal rules were found, mark all redundant consequences as impossible as explained in Sect. 4.3 (Algorithm 5.4, lines 23–25 and 28).
- Use the Lapis Philosophorum principle to propagate information on possible consequences to parents. (Algorithm 5.4, line 29 and Algorithm 5.3, lines 12–14.)

We note that when the χ^2-measure is used as a goodness measure, then the upper bounds in Table 2 are used instead of the lower bounds. Because χ^2 is an increasing measure, all comparisons concerning the M-value are reversed and max_M is replaced by min_M.

Another note concerns the interchangeability of the goodness measure M. If M is interchangeable, then $M(A \rightarrow B) = M(B \rightarrow A)$ and it is sufficient to report the dependency just once. In the implementation, we report only the rule with a larger confidence. In addition, with some measures like the χ^2-measure, mutual information and Fisher's p_F also holds $M(A \rightarrow \neg B) = M(B \rightarrow \neg A)$. In this case, we also report the rule, which has a larger confidence. In both cases, the *best*-values are updated for both consequences of equivalent rules.

5.2 Simulation

The following simulation demonstrates the Kingfisher algorithm with the example data given in Table 3. In this example, we search for all non-redundant rules, whose p_F-value is at most 1.2×10^{-8}. The discovered rules are represented in Table 4.

Table 3 Example data

Set	Freq.
$ABC\neg D$	10
$A\neg B\neg CD$	85
$\neg AB\neg CD$	5

Table 4 The rules discovered from the example data

Rule	p_F
$AD \rightarrow \neg B$	3.9×10^{-18}
$D \rightarrow \neg C$	5.8×10^{-14}
$AB \rightarrow C$	5.8×10^{-14}
$AB \rightarrow \neg D$	5.8×10^{-14}
$C \rightarrow B$	1.7×10^{-10}
$D \rightarrow \neg B$	1.7×10^{-10}

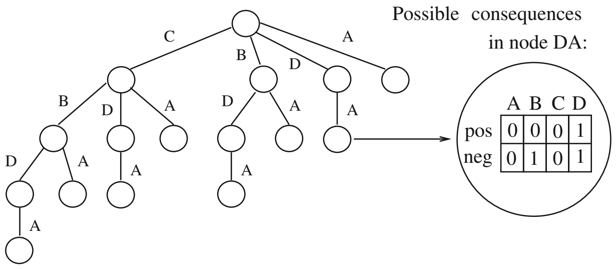

Fig. 3 Nodes of the enumeration tree are represented by bit-vectors corresponding to possible positive and negative consequences. For example, in the node for DA possible consequences are D, $\neg B$ and $\neg D$

The simulation is shown in Fig. 4. Each node of the enumeration tree shows bit-vectors for possible consequences as explained in Fig. 3. Abbreviation "LP" refers to the Lapis Philosophorum principle.

In the beginning, the algorithm calculates frequencies of all single attributes and checks with *lb1*, whether the attribute can occur in any consequence at all. In this example, A and $\neg A$ are impossible consequences, but attribute A can occur in the antecedent of significant rules.

On the first level, all single attributes are added to the enumeration tree and their consequence vectors are initialized using lower bounds (step 1 in Fig. 4). We note that the attributes are added in an ascending order by frequency, because then the potentially largest subtree, under C, tends to have the smallest frequencies and is pruned effectively.

On the second level, the algorithm first creates node CB (step 2). Its consequence vectors are initialized by combining the parents' consequence vectors by logical bit-and operations. Because B, C and $\neg D$ are possible consequences, their lower bounds and rules are checked.

The algorithm finds a significant rule, $C \rightarrow B$. The rule has confidence 1.0, and therefore, B and $\neg D$ are marked as impossible consequences by the minimality condition (step 3). The Lapis Philosophorum principle is used to set B as an impossible consequence also in the parent node C. (In the other parent, B, consequence $\neg C$ was already impossible and LP has no effect.)

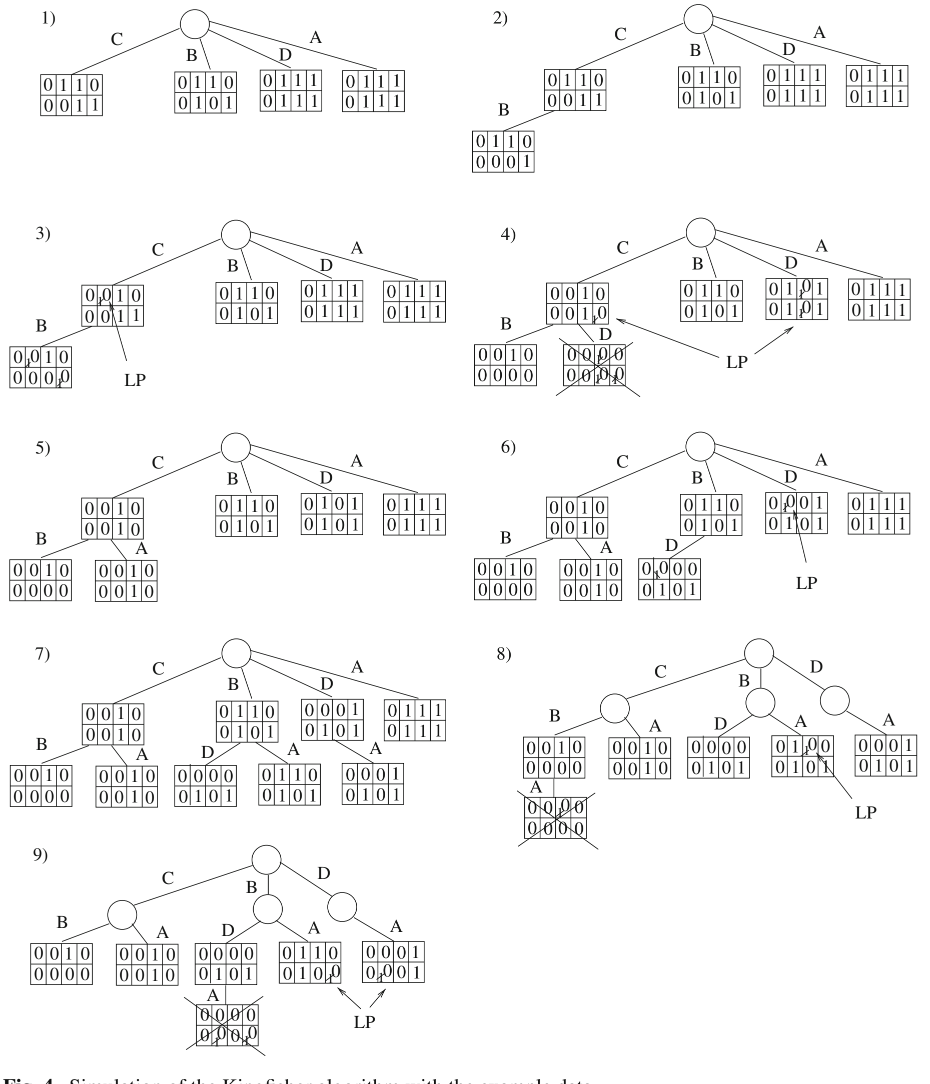

Fig. 4 Simulation of the Kingfisher algorithm with the example data

In step 4, the algorithm creates node CD and initializes and updates its consequence vectors like before. Because all consequences become impossible, the node is removed. Once again, the Lapis Philosophorum principle is applied in the parent nodes C and D.

In step 5, node CA is added, but no rules are found.

In step 6, node BD is created. The algorithm finds a significant rule, $D \rightarrow \neg B$, but it is not minimal. B is set as an impossible consequence based on lower bound *lb3*. The Lapis Philosophorum principle is used to propagate the information to parent node D.

In step 7, BA and DA are created, but no rules are found.

On the third level, the algorithm first creates node CBA (step 8). It finds a significant rule, $AB \rightarrow C$. Since the rule has confidence 1.0, C becomes an impossible consequence. Because all consequences are impossible, the node is removed. LP is used to propagate the information to BA.

In step 9, the algorithm creates node BDA. It finds two minimal rules (with confidence 1.0), $AD \rightarrow \neg B$ and $AB \rightarrow \neg D$. Once again, all consequences become impossible and the node is removed. After that the algorithm finishes, because no more nodes can be created.

5.3 Implementation issues

In the Kingfisher algorithm, the main objective has been to optimize the execution time. This has been achieved by two subobjectives: by minimizing the size of the enumeration tree, especially the number of leaf nodes, and the number of frequency calculations, which are costly operations. In practice, the space requirement is also quite feasible, because all data contents (possible- and best-tables) are stored into the leaf nodes, whose number is minimized. The nodes themselves could be implemented more compactly, but as a trade-off, it would require more costly search operations.

Implementation of the enumeration tree is the most crucial factor for the efficiency, because it represents the exponential search space. In Kingfisher, we have adopted a strategy, where each set X corresponds to rules $X \setminus \{A_i\} \rightarrow A_i = a_i$, for all $A_i \in X,\ a_i \in \{0, 1\}$. Since the enumeration tree is traversed in a breath-first manner, all frequencies of parent sets $Y_i \subsetneq X,\ X = Y_i A_i$, are already stored into the tree. Therefore, it is enough to evaluate just the frequency of set X to get all required frequencies $m(Y_i)$ and $m(Y_i A_i = a_i)$. By contrast, MagnumOpus [26,27]—whose search strategy is otherwise the nearest previous equivalent to Kingfisher—organizes the search space by an enumeration tree of rule antecedents. Each node of the tree, representing set X, corresponds to rules $X \rightarrow A_i = 1,\ A_i \in R \setminus \{A_i\}$. We note that MagnumOpus does not implement negative rules, but the same search strategy could be applied to negative rules as well [22]. The problem of this approach is that the same frequencies have to be evaluated multiple times.

Another important difference to MagnumOpus search strategy is that Kingfisher combines the possible consequences from all $|X|$ parent nodes when evaluating the possible consequences for set X. In MagnumOpus, the node inherits only its immediate parent's possible consequences. In practice, this can cause much unnecessary work, because a consequence may well be possible in the immediate parent, even if it is impossible in another parent. It can also happen that all consequences become impossible, when the parents' possible-tables are combined, and the node can be deleted immediately without checking its frequency or evaluating any rules. Deleting the node can further impact other parent nodes, which may become deleted before any children are created for them. In this way, the information on possible consequences is propagated to the whole enumeration tree, which can lead to dramatic pruning. The only drawback of this strategy is that the possible-tables have to be stored into nodes, which is space consuming. Fortunately, it is enough to keep the possible-tables only in the leaf nodes, which can be parents to next level nodes. Similarly, the best-tables—which enable efficient redundancy checking—are kept only in the leaf nodes. When the data contents (possible- and best-tables) become useless, they can be destroyed and the node becomes an inner node, whose only purpose is to represent the search space (currently examined sets). The node can be even deleted, if it cannot be a parent node to any current level nodes. This saves both space and traversing time significantly.

In practice, the enumeration tree can be implemented as a trie (a prefix tree), whose inner nodes contain just the node labels and pointers to the child nodes, while the leaf nodes contain also the best- and possible-tables and frequencies. If the child nodes are kept in an order by their labels, a child with a given label can be searched with a binary search in $\mathcal{O}(\log(d))$ time, where d is the number of children. This means that a parent set $Y,\ |Y| = l$, can be searched from the tree in $=(\log(d_0) + \ldots + \log(d_{l-1}))$ time, where d_is are numbers of children in the nodes from the root to the node corresponding to set Y. The search could be implemented more efficiently, in $\mathcal{O}(l)$ time, if each node contained pointers to all possible children, whether they existed or not. However, the resulting data structure would be too space consuming.

For efficient frequency counting, the data set is stored in the vertical layout, by k bit-vectors, each of them n bits long. In this representation, the frequency of set $X \subseteq R$ can be evaluated efficiently by bitwise logical operations. Compared to horizontal layout (n bit-vectors, each of them k bits long), the efficiency of vertical layout is superior, in spite of the same asymptotic time complexity [13].

The final note concerns an efficient evaluation of Fisher's exact test. Asymptotic measures, like the χ^2-measure, can be evaluated in constant time, but calculating Fisher's p_F can be a time consuming operation, because the value is a sum of several terms, each containing binomial coefficients. In addition, the evaluation can easily cause an overflow or underflow. For the latter problem, a common solution is to use logarithms in the middle steps, when each term is evaluated. For the efficiency problem, we use two strategies. First, the logarithms of all factorials, $\ln(m!) = \sum_{i=1}^{m} \ln(i),\ m = 1, \ldots, n$, are calculated just once and stored into a table. Thus, each term in p_F can be evaluated in constant time. Evaluating $p_F(Y \rightarrow A = a)$ takes $J = \min\{m(YA \neq a), m(\neg YA = a)\} \leq \frac{n}{4}$ steps. Second, we avoid evaluating the exact p_F value as often as possible and instead check only its first term, which gives a lower bound. If the lower bound is too large for rule $Y \rightarrow A = a$ to be sufficiently significant and non-redundant, there is no reason to evaluate the exact value.

5.4 Time and space complexity

Next, we analyze the worst-case time and space complexity of the Kingfisher algorithm. Both time and space complexity are exponential in the number of attributes. This is not surprising, because already a simpler problem—searching for the best classification rules of the form $X \rightarrow C = c,\ c \in \{0, 1\}$, with common goodness measures like the χ^2-measure—is NP-hard [19]. One problem in the complexity analysis is that the complexity depends on the data distribution, and the effect of new pruning strategies is impossible to analyze. In Sect. 6, we will see that in practice the algorithm is quite feasible with the classical benchmark data sets.

The following theorem gives the worst-case time complexity of the Kingfisher algorithm. We assume that the maximal transaction length (maximal number of 1s on any row), L, is given. Parameter L defines the maximal level (depth) and, thus, the maximal size of the generated enumeration tree. Typically $L << k$, and the resulting complexity is more realistic than the assumption that a complete enumeration tree with 2^k nodes is generated.

Theorem 1 *Let n be the number of rows, k the number of attributes, K the number of best rules to be searched for and L the maximal transaction length.*

Then the worst-case time complexity of searching for the K best dependency rules with the Kingfisher algorithm is

(i) $\mathcal{O}\left((k + n + \log(K))L \min\left\{2^{k-1}, \frac{L+2}{k-2(L+2)} \binom{k}{L+2}\right\}\right)$, *if* $L + 2 < \frac{k}{2}$, *and*
(ii) $\mathcal{O}((k + n + \log(K))k2^{k-1})$, *otherwise.*

Proof Processing the first level (single attribute sets) takes $\mathcal{O}(\log(n) + k\log(k) + kn + k^2) = \mathcal{O}(k(n + k))$. This is composed as follows: Determining min_{fr} (Algorithm 5.1 line 1) can be done in $\mathcal{O}(\log(n))$ using a binary search (iterating interval $[1, n/2]$). Algorithm 5.2 takes in the worst case $\mathcal{O}(kn + k\log(k) + k^2)$. Frequency counting for each of the k attributes takes at most n steps, which together make kn steps. The attributes can be ordered in $\mathcal{O}(k\log(k))$ time. Possible consequences are determined for at most k nodes, and for each node, at most $2k$ consequences (each positive and negative consequence) are checked. These make together $2k^2$ steps. With some measures, like Fisher's p_F, this phase can be implemented more efficiently, but it does not make any difference to the overall asymptotic complexity.

Let us then analyze the complexity of processing levels $l \geq 2$. The last level is always $\leq L+1$, because after that level all sets $X \subseteq R$ have frequency 0 (on level $L+1$, $m(XA)=0$ for all $A \notin X$, $|X| = L$, but $m(X\neg A) = m(X)$). However, $l \leq k$ and, therefore, the last level is $\leq \min\{L+1, k\}$. On each level $l = 2, \ldots, \min\{L+1, k\}$, the time complexity is $\mathcal{O}\left(\binom{k}{l} l(k+n+\log(K))\right)$.

Traversing all nodes on the lth level of the tree (Algorithm 5.3) takes at most $\mathcal{O}(lN_l)$, where N_l is the number of nodes on level l. The reason is that the tree is always pruned such that it contains only paths leading to level $l-1$. Reaching each of the new nodes on level l takes at most l steps. If the leaf nodes were linked to their successors, the traversing could be done in N_l steps. The value of N_l is difficult to evaluate, because it depends on the data distribution. However, it has always an upper bound $N_l \leq \binom{k}{l}$, which is the number of all possible l-sets.

Processing each l-set (Algorithm 5.3, lines 9–14 and Algorithm 5.4) takes $\mathcal{O}(l(k+n+\log(K)))$. This is composed as follows:

The first two parents are always known, but the rest $l-2$ parents have to be searched for from the tree (Algorithm 5.4, line 3). Searching an $(l-1)$-set from the enumeration tree can be done in time $\mathcal{O}(l\log(k))$. Then, in each node, the correct child can be found in $\log_2(k)$ time using a binary search. (In practice, the number of possible children is always $< k$, except in the root.) Since the path length is $l-1$, the overall complexity is the given.

All *possible*- and *best*-values are first initialized (Algorithm 5.3 line 9) and then updated using the parents (Algorithm 5.4, line 5). This takes at most $(l+1)(2k+2l)$ steps, because there are at most k consequences in both *possible*-tables and l possible consequences in both *best*-tables, and all tables are processed at most $l+1$ times (initialization + l times updating). Since $l \leq k$, the overall complexity is $\mathcal{O}(kl)$. (We note that in practice, we also have to reserve space, when new child nodes are added, and free space, when they are deleted, but all additions and deletions can be done in $\mathcal{O}(k)$ time per node.)

Frequency counting (Algorithm 5.4, line 7) takes at most ln steps. The *possible*-values are updated once again (Algorithm 5.4, lines 11–12), which takes at most $2k$ steps. Together, these take $\mathcal{O}(ln+k)$.

Assuming that measure M can be calculated in constant time, the rule checking (Algorithm 5.4, lines 13–22) takes at most $\mathcal{O}(l\log(K))$. Each of the at most $2l$ possible rules can be checked in constant time. Adding a rule to collection *brules* can be done in $\mathcal{O}(\log(K))$ time, if the collection is implemented as a binary heap with the worst rule on the top.

If minimal rules are found, the *possible*-tables are updated once again (lines 23–28), taking at most $2k$ steps. Lapis Philosophorum (Algorithm 5.4, line 29 and Algorithm 5.3, lines 12–14) can be implemented in $2l$ steps. Since $l \leq k$, these take together $\mathcal{O}(k)$ time.

Thus, processing each l-set takes $\mathcal{O}(l\log(k)) + \mathcal{O}(kl) + \mathcal{O}(ln+k) + \mathcal{O}(l\log(K)) + \mathcal{O}(k) = \mathcal{O}(l(k+n+\log(K)))$. The lth level takes $\mathcal{O}(N_l l(k+n+\log(K))) = \mathcal{O}\left(\binom{k}{l} l(k+n+\log(K))\right)$. Because the first level takes $\mathcal{O}\left(\binom{k}{1} 1(k+n)\right)$, the time complexity of processing levels $1, \ldots, L+1$ has an upper bound

$$\mathcal{O}\left((k+n+\log(K))\sum_{l=1}^{L+1}\binom{k}{l}l\right).$$

Because $L+1 \leq k$, we can always use an upper bound $\mathcal{O}((k+n+\log(K))k2^{k-1})$, but it is often unnecessarily large. Typically $L+2 < \frac{k}{2}$, and we can use the following upper bound [8, 122–123]:

$$\sum_{l=1}^{L+1}\binom{k}{l}l < (L+1)\sum_{l=1}^{L+1}\binom{k}{l} < \frac{(L+1)(L+2)}{k-2(L+2)}\binom{k}{L+2}.$$

Since $\sum_{l=0}^{\frac{k}{2}}\binom{k}{l} \leq 2^{k-1}$, the upper bound is

$$\sum_{l=1}^{L+1}\binom{k}{l}l < (L+1)\min\left\{2^{k-1}, \frac{L+2}{k-2(L+2)}\binom{k}{L+2}\right\}.$$

Therefore, the total complexity is

$$\mathcal{O}((k+n+\log(K))k2^{k-1}),$$

when $L+2 \geq \frac{k}{2}$, and

$$\mathcal{O}\left((k+n+\log(K))L\min\left\{2^{k-1}, \frac{L+2}{k-2(L+2)}\binom{k}{L+2}\right\}\right),$$

when $L+2 < \frac{k}{2}$. □

Typically $L << k$, and the complexity can be bounded by the binomial coefficient. For example, if $k \geq 3(L+2)$, the complexity reduces to $\mathcal{O}\left((k+n+\log(K))L\binom{k}{L+2}\right)$.

If the maximal transaction length L is not given, we can use an upper bound $L \leq k$. Then, the time complexity becomes $\mathcal{O}((k+n+\log(K))k2^{k-1})$. In a typical case, where $k \leq n$ and $K \leq n$, the expression can be simplified to $\mathcal{O}(nk2^k)$. This is a loose upper bound, because it corresponds to checking all possible sets in $\mathcal{P}(R)$.

In the previous analysis, we have assumed that the measure M can be calculated in constant time. This is true for asymptotic measures, but some measures, like Fisher's p_F, require summing over several terms and each term can contain binomial coefficients. Tabulating all factorials (as explained in Sect. 5.3) enables us to evaluate each term in constant time, but the asymptotic time complexity of calculating p_F is still quite large, $\mathcal{O}(n)$.

Let us then analyze the worst-case space complexity of the Kingfisher algorithm.

Theorem 2 *Let k, K and L be like before. Then the space complexity of searching for the K best dependency rules with the Kingfisher algorithm is*

(i) $\mathcal{O}\left(k\min\left\{2^{k-1}, \frac{L+2}{k-2(L+2)}\binom{k}{L+2}\right\}\right)$, *if $L+2 < \frac{k}{2}$, and*
(ii) $\mathcal{O}(k2^k)$, *otherwise.*

Proof First, we note that the enumeration tree contains two kinds of nodes, which we call *structure nodes* and *data nodes*. When level l is finished, nodes on levels $1, \ldots, l-1$ only

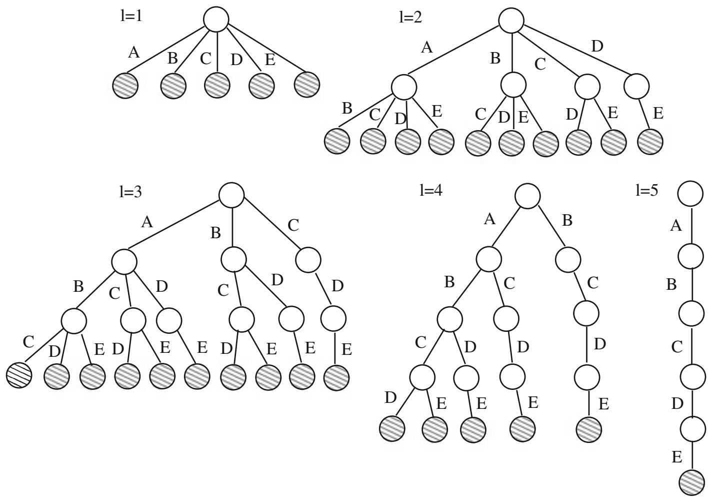

Fig. 5 An example of the worst-case tree development. Data nodes are shaded

code the tree structure. All data nodes (containing *best*- and *possible*-tables) are stored onto level l. On level $l+1$, new data nodes are created onto level $l+1$ and the previous level data nodes are either removed or changed to structure nodes. All nodes which do not lead to the last level are removed.

In the worst case, we have to construct all l-sets, $l = 1, \ldots, \min\{L+1, k\}$. After each level l, the enumeration tree contains all l-sets and all inner nodes (structure nodes), which lead to leaf nodes. Figure 5 shows an example of the worst-case tree development (the tree after each level l), when all sets are generated.

After each level l, the number of data nodes is $N_d = \binom{k}{l}$. For the structure nodes, we can give an upper bound

$$N_{str} \leq \sum_{j=1}^{l-1} \binom{k-1}{j}.$$

The derivation is the following: On each level, we know that at least all nodes with label A_k are leaf nodes and removed. (In addition, new nodes become leaf nodes later, when their children are removed.) If all possible sets are generated, the number of non-leaf nodes on level j is at most the number of all j-sets, which do not contain attribute A_k, i.e., $\binom{k-1}{j}$. Summing over all levels $j = 1, \ldots, l-1$ gives N_{str}.

Because the structure node contains only the label and child pointers, all structure nodes up to level $l-1$ take together space $\mathcal{O}(N_{str} + N_d)$ (each node is pointed by one pointer). This can be bounded by $\mathcal{O}(kN_{str})$, because each structure node contains at most k child pointers.

Each data node contains two *possible*-tables, whose maximum size is k bits and two *best*-tables, whose maximum size is l. Since $l \leq k$, the space requirement for each data node is at most $\mathcal{O}(k)$, and together, all data nodes on level l take space $\mathcal{O}(kN_d)$.

Therefore, the worst-case space requirement for the whole tree after level l is

$$\mathcal{O}\left(k\sum_{j=1}^{l-1}\binom{k-1}{j}+k\binom{k}{l}\right)\leq\mathcal{O}\left(k\sum_{j=1}^{l}\binom{k}{j}\right).$$

This is largest on the last level, which is $l \leq \min\{L+1, k\}$. The sum can always be bounded by $\mathcal{O}(k2^k)$. However, if $L+2 < \frac{k}{2}$, we can give a tighter upper bound (like in the proof for Theorem 1)

$$\mathcal{O}\left(k\min\left\{2^{k-1},\frac{L+2}{k-2(L+2)}\binom{k}{L+2}\right\}\right).$$

In addition to the enumeration tree, we have to store the data and the best rules. Because the data can be represented by a $n \times k$ bit matrix, it takes at most $\mathcal{O}(nk)$ space. (More compact representations are possible, but this supports efficient frequency counting.) For the K best rules, we have to store the attributes of the rule and at least the goodness measure value and information, whether the consequence is negated ("sign"). The measure value, sign and possible other parameters (like frequencies $m(XA=a)$, $m(X)$ and $m(A=a)$) take constant space. Because the maximum rule length is $L+1$, the best rules take at most space $\mathcal{O}(KL)$. □

In a typical case, where $n \leq 2^{k-1}$ and $K \leq n$, and L is not given, the complexity simplifies to $\mathcal{O}(k2^k)$. However, if the maximal transaction length L is known, it gives often a more realistic upper bound for the space complexity.

6 Experiments

The goal of the experiments was twofold. First, we were interested in to compare the quality of discovered rules between p_F and χ^2. This is an interesting issue, since Kingfisher is the first algorithm, which enables the search with p_F, and it has often been assumed that the χ^2-measure produces fairly comparable results. Second, we wanted to assess the efficiency of the search with both measures as well as the effect of the new pruning principle (Lapis Philosophorum). The underlying question was what is the quality-efficiency trade-off between p_F and χ^2. To our surprise, p_F turned out to be clearly superior in both aspects.

6.1 Test setting

For testing, we used classical benchmark data sets, which are available in the FIMI repository (http://fimi.cs.helsinki.fi/data/). The data sets and their dimensions are given in Table 5.

In all data sets, we searched for the 100 best non-redundant rules with p_F and the continuity corrected χ^2-measure. When the χ^2-measure was used, we needed low minimum frequency thresholds with data sets Accidents, Pumsb and Retail. The thresholds are reported in Table 5. With Pumsb, we also restricted the search to level 7, because no new rules were found after that. In other data sets, we used only the general requirement that all rules should occur on at least five rows of data. With p_F, no thresholds were needed, because minimum frequency thresholds are derived implicitly from the maximal p_F-value requirement.

The implementation of Kingfisher algorithm is available on http://www.cs.helsinki.fi/u/whamalai/sourcecode.html. All experiments were run on 2.5 GHz AMD Opteron 8360 SE having 256 GB central memory and Linux operating system. We note that the exceptionally

Table 5 Data sets

	n	k	$tlen$	min_{fr}
Mushroom	8,124	119	23.0	–
Chess	3,196	75	37.0	–
T10I4D100K	100,000	870	10.1	–
T40I10D100K	100,000	942	39.6	–
Accidents	340,183	468	33.8	0.0001
Pumsb	49,046	2,113	74.0	0.0005
Retail	88,162	16,470	10.3	0.0005

n number of rows, k number of attributes, $tlen$ average transaction length, min_{fr} minimum frequency threshold used with the χ^2-measure

large memory size was beneficial for the χ^2-measure, but with p_F, all experiments could be run with just 1 GB main memory.

Finally, we recall that there are no previous algorithms for searching for both positive and negative dependency rules with the χ^2-measure or p_F, and therefore, no comparisons to other algorithms could be made. If we were interested in only positive dependency rules with the χ^2-measure, we could first search for association rules with a minimal possible frequency threshold and then filter the best non-redundant rules afterwards, using the χ^2-measure. However, according to our previous experiments [12, Ch. 5], this approach is enormously inefficient, unless large minimum frequency thresholds are used. For example, if we search for the 100 best rules (including redundant rules) by Borgelt's Apriori [7] in the same test environment, set Pumsb requires $min_{fr} = 0.45$ and the maximal rule length (number of attributes, including the consequence attribute) of five. These are so strict restrictions that 73% of the actually best non-redundant rules could not be found even in principle.

6.2 Quality evaluation

In the quality evaluation, the objective is that the discovered dependency rules should be reliable, i.e., hold also in future data. To estimate the behaviour in future data, we used the following cross-validation scheme: Each data set was divided 10 times randomly to a training set and a test set. Because each test set should also be sufficiently large, we used 2/3 of the data for training and 1/3 for testing. For each training set, we searched for the 100 best rules and evaluated them in the test set. For assessing how well the dependency held in the test data, we calculated the mean squared errors of lift (MSE_γ) and leverage (MSE_δ). If the discovered dependency rules were equally strong in the test set, then both MSE_γ and MSE_δ would be zero. On the other hand, large MSE values indicate that the dependencies were significantly weaker or stronger in the test sets. It is not necessarily harmful, if the dependency is stronger in the test set (future data), but in some applications, one may want to ascertain that the data does not contain any strong dependencies. Because the MSE values are difficult to interpret, we report the root mean squared errors, which are in the same scale as the lift and leverage.

The results of the quality evaluation are represented in Tables 6 and 7. In addition to $\sqrt{MSE_\gamma}$ and $\sqrt{MSE_\delta}$, we report the average frequency, confidence, lift and leverage in the training sets.

In all data sets, p_F produced more frequent rules, with a significantly smaller lift but larger leverage than the χ^2-measure. The χ^2-measure suffered for exaggerated values (due to extremely low or high frequency rules) in all sets except Chess and Retail. This was

Table 6 Statistics characterizing the quality of the 100 best non-redundant rules discovered with measure p_F

Set	fr	cf	γ	δ	$\sqrt{MSE_\gamma}$	$\sqrt{MSE_\delta}$
Mushroom	0.2364	0.97	4.3	0.1713	0.2	0.00479
Chess	0.4534	0.84	2.1	0.1672	0.1	0.00639
T10I4D100K	0.0073	0.89	68.1	0.0071	4.7	0.00053
T40I10D100K	0.0150	0.96	36.0	0.0145	2.2	0.00072
Accidents	0.3495	0.94	3.4	0.1254	0.0	0.00093
Pumsb	0.4876	1.00	2.1	0.2462	0.0	0.00038
Retail	0.0096	0.61	140.1	0.0042	37.1	0.00046

Table 7 Statistics characterizing the quality of the 100 best non-redundant rules discovered with measure χ^2

Set	fr	cf	γ	δ	$\sqrt{MSE_\gamma}$	$\sqrt{MSE_\delta}$
Mushroom	0.0550	1.00	339.0	0.0232	448.0	0.00227
Chess	0.2988	0.93	123.9	0.0585	75.8	0.00456
T10I4D100K	0.0007	1.00	4221.9	0.0007	8250.0	0.00015
T40I10D100K	0.0012	0.99	1632.6	0.0012	846.0	0.00019
Accidents	0.2210	0.92	228.5	0.0908	664.0	0.00082
Pumsb	0.3379	1.00	132.6	0.1173	132.0	0.00136
Retail	0.0019	0.57	288.7	0.0017	71.1	0.00029

especially clear in Mushroom, T10I4D100K and Accidents where the lift values held poorly in the test sets. On the other hand, the dependencies by p_F held extremely well in all test sets ($\sqrt{MSE_\gamma}$ was about 7% from the average lift and $\sqrt{MSE_\delta}$ about 4% from the average leverage, while for the χ^2-measure, the proportions were about 122 and 11%).

In set Accidents, the χ^2-measure produced some harmful rules (about 0.6%), which expressed independence in the test sets. In all training sets, the harmful rule was the same negative dependency rule, which held in nearly all rows of data. In the corresponding test sets, it occurred on all rows or was missing only from a couple of rows. We note that other classical goodness measures, especially when combined to large minimum frequency thresholds, produce often a large number of harmful rules which express positive dependence in the learning set, but negative dependence or independence in the test sets [13].

Negative dependency rules were found among the 100 best rules in sets Mushroom (6.6% with p_F and 6.0% with χ^2), Chess (61.6 and 36.0%), Accidents (42.8 and 29.2%) and Pumsb (63.3 and 18.8%). The χ^2-measure found less and simpler negative dependency rules (typically 2-rules), while p_F produced also more complex rules. The reason is that the χ^2-measure gets its maximal value, whenever $m(XA = a) = m(X) = m(A = a)$. This occurs more often for simple and relatively infrequent rules, while the negative dependency rules were typically quite frequent.

6.3 Efficiency evaluation

In the efficiency evaluation, we compared the search with p_F and χ^2. With both goodness measures, we tested two versions of the Kingfisher algorithm: the original version and another one where the Lapis Philosophorum principle was disabled.

Table 8 Parameters for the efficiency comparison: max_M is the initial value for the threshold $ln(max_p)$, min_{fr} is the corresponding implicit minimum frequency threshold, and min_M is the minimal χ^2 threshold

Set	max_M	min_{fr}	min_M
Mushroom	−2,000	0.06745	8,000
Chess	−850	0.07541	2,200
T10I4D100K	−1,600	0.00227	93,000
T40I10D100K	−3,500	0.00569	95,000
Accidents	−73,000	0.05557	1,60,000
Pumsb	−18,000	0.12011	46,000
Retail	−350	0.00047	10,000

Table 9 Efficiency comparison of Kingfisher using p_F with and without the Lapis Philosophorum principle

Set	With LP				Without LP			
	l	w	$wsize$	t	l	w	$wsize$	t
Mushroom	6	3	454	0	7	3	959	0
Chess	15	8	23,09,751	169	≥10	≥10	≥3,63,98,778	≥1,200
T10I4D100K	4	2	1,399	11	8	2	3129	12
T40I10D100K	6	2	9,098	17	18	7	91112	92
Accidents	13	6	1,42,133	79	≥7	≥7	≥34,48,942	≥1,200
Pumsb	11	6	30,009	7	≥7	≥7	≥4,71,05,062	≥1,200
Retail	5	2	6,387	379	6	2	9,208	389

l last level, w widest level, $wsize$ size of the widest level, t execution time in seconds

With each experiment, we searched for the 100 best rules from the whole data set. For p_F, no minimum frequency thresholds were used, but the initial $ln(max_p)$-values and corresponding (implicit) minimum frequency thresholds are reported in Table 8. For the χ^2-measure, we used the same minimum frequency thresholds as for the quality evaluation. The initial minimum χ^2 thresholds are given in Table 8. When the Lapis Philosophorum principle was disabled, the program often got stuck and was halted after 20 min CPU time.

The results are represented in Tables 9 and 10. The size of the generated enumeration tree (traversed search space) is described by three variables: the last level, the widest level and the number of sets on the widest level. We recall that the enumeration tree is pruned after each level, and thus, the widest level is the bottleneck. In addition, we give the execution time in seconds.

With the original Kingfisher using p_F, the whole search space could be traversed, and therefore, the discovered rules were globally optimal. The most demanding data set was Retail, where the number of attributes is extremely large. In addition, all dependencies are relatively weak, and therefore, Kingfisher could not determine any effective minimum frequency from the maximal p_F-value requirement. Most of the execution time was spent on level 2, where the program had to determine lower bounds for over 17 million attribute combinations.

The implicit minimum frequency thresholds explain the efficiency of Kingfisher only on the first levels. After that, the Lapis Philosophorum principle begins to play a more important role. When the principle was not used, the program got stuck with data sets Chess, Accidents and Pumsb. The sparsest data sets could be handled without Lapis Philosophorum, but the enumeration tree was still significantly larger. In the densest data sets, the widest level was

Table 10 Efficiency comparison of Kingfisher using χ^2 with and without the Lapis Philosophorum principle

Set	With LP				Without LP			
	l	w	$wsize$	t	l	w	$wsize$	t
Mushroom	8	4	6,486	0	9	4	22,004	1
Chess	19	10	46,38,560	914	≥ 9	≥ 8	$\geq 7{,}07{,}03{,}089$	$\geq 1{,}200$
T10I4D100K	4	2	604	23	10	3	55,338	182
T40I10D100K	7	4	2,413	28	≥ 3	≥ 2	$\geq 3{,}18{,}909$	$\geq 1{,}200$
Accidents	15	7	13,72,901	1,038	≥ 4	≥ 3	$\geq 7{,}27{,}536$	$\geq 1{,}200$
Pumsb	≥ 8	≥ 7	$\geq 1{,}09{,}62{,}956$	$\geq 1{,}200$	≥ 4	≥ 3	$\geq$ 41,38,149	$\geq$ 1,200
Retail	5	2	7,598	335	6	2	8,424	319

l last level, w widest level, $wsize$ size of the widest level, t execution time in seconds

at least 15–1570 times as large as with the Lapis Philosophorum principle and none of these experiments could have been run in an ordinary desktop computer.

Search with the χ^2-measure was remarkably slowlier. Chess, Accidents, Pumsb and Retail were the most demanding data sets for the χ^2-measure, in spite of minimum frequency thresholds. The problem is that the minimal χ^2-value requirement does not define any minimum frequency threshold, but any rule—even one occurring on just one row—can gain the maximal value. Therefore, the upper bounds used for pruning tend to be large and the search continues deep.

When the Lapis Philosophorum principle was used, the program got stuck only with set Pumsb, on level 8. For curiosity, we tested the most difficult data set, Pumsb, also without any minimum frequency threshold or maximal rule length. The search continued to level 20, and the widest level was 11 with over 60 million sets. The execution took 411 min. The interesting phenomenon was that now all the 100 best rules were found on levels 2 and 3, but the program was not able to decide that no more rules are to be found on deeper levels.

When the Lapis Philosophorum principle was disabled, the program got soon stuck in all data sets, except the sparsest—Mushroom, T10I4D100K and Retail (where $min_{fr} = 0.0005$ was required). In Accidents and Pumsb, only the first three levels could be processed in 20 min CPU time. The biggest surprise was set T40I10D100K, which was easy to handle without any extra restrictions, when Lapis Philosophorum was used, but without it just the first two levels could be finished in 20 min.

7 Conclusions and future research

Searching for statistical dependencies from high-dimensional data is a fundamental problem in all empirical science. The problem is computationally very demanding, and so far, there have not been any scalable solutions to the general problem. Heuristic or suboptimal solutions are often insufficient, because the new scientific theories should be based on valid information. In practice, the scientists want to find the most significant dependencies which hold also in future data. For this purpose, we need an efficient search algorithm, which optimizes some statistical significance measure like Fisher's p-value, without any other restrictions like minimum frequency requirements. Pruning redundant dependencies is also important, because scientists want to find the real causes of dependencies, without any weakening or occasional extra factors.

In this paper, we have introduced an effective solution for a special case, where dependency rules are searched for from binary data. The rules can be of the form $X \rightarrow A$, $X \rightarrow \neg A$, $\neg X \rightarrow A$ or $\neg X \rightarrow \neg A$, where X is a set of positive-valued attributes, and A can be any attribute. We recall that any categorical data can be represented in such a form by creating a new attribute for each attribute value in the original data (numerical data would require discretization first).

The new Kingfisher algorithm can be used to search for either the K best, non-redundant dependency rules or to enumerate all sufficiently good, non-redundant rules. The search algorithm itself is applicable to any statistical goodness measure, given the required upper or lower bounds, but in this paper, we have focused on the search with Fisher's exact test and the χ^2-measure. For this purpose, we have introduced new, tight lower bounds for Fisher's p_F-value. In addition, we have introduced several general pruning principles, which enable to restrict the search into areas, where the most significant non-redundant dependencies are to be found.

According to our experiments, the algorithm is extremely well-scalable, even to densest and largest-dimensional data sets, when p_F is used as a search criterion. No minimum frequency thresholds or other restrictions were required. Surprisingly, the χ^2-measure turned out to be much less effective, and small minimum frequency thresholds were needed with the most demanding data sets. Another surprising result was the relatively poor quality of rules, when the χ^2-measure was used. With p_F, the results were very accurate, and we can conclude that Kingfisher offers a robust and efficient tool for scientists.

Finally, we note that in this research, we have taken a very careful attitude against losing significant rules and pruned out only clearly redundant rules. In reality, some dependency rules may appear as non-redundant only in the given data, but in future data, they would be redundant. Clearly, they could be pruned out and thus the efficiency improved, but first one should develop the statistical theory for the significance of improvement. On the other hand, some of the discovered significant dependency rules may be specious, just by-products of other dependency rules, and not interesting as such. Detecting specious rules would also require more statistical investigation as well as new algorithmic insights for efficient pruning. Another important challenge for future research is to develop an efficient algorithm for searching for general dependency rules (containing an arbitrary number of negations). All these problems are likely to require new Philosopher's stones or other magic tricks to be solved.

Appendix A: Proofs for the three lower bounds on Fisher's p_F

Theorem 3 *Let us notate* $p_{abs} = \frac{m(A)!m(\neg A)!}{n!}$. *For any attribute* $A \in R$ *and* $X \subseteq R \setminus \{A\}$, $p_F(X \rightarrow A) \geq p_{abs}$ *and* $p_F(X \rightarrow \neg A) \geq p_{abs}$.

Proof p_F can be expressed as

$$p_F(X \rightarrow A) = p_{abs} \sum_{i=0}^{J} \binom{m(X)}{m(XA)+i} \binom{m(\neg X)}{m(\neg X \neg A)+i}.$$

Since the sum is always ≥ 1, the minimum value is p_{abs}. Case $p_F(X \rightarrow \neg A)$ is similar. □

Theorem 4 *For all $X \subsetneq R,\ A \in R \setminus X,\ a \in \{0,1\}$ and $Q \subseteq R \setminus (X \cup \{A\})$ such that $m(X) \leq m(A=a)$ holds*

$$p_F(XQ \to A=a) \geq \frac{m(\neg X)!m(A=a)!}{n!(m(A=a)-m(X))!}.$$

Proof $p_F(XQ \to A = a)$ is of the form $t_0 + \dots t_J$, where $t_i = p_{abs}\binom{m(XQ)}{m(XQA=a)+i}\binom{n-m(XQ)}{m(\neg(XQ)A \neq a)+i}$. Therefore, $p_F \geq t_J$, where $J = \min\{m(XQA \neq a), m(\neg(XQ)A = a)\}$. Because $m(XQ) \leq m(X) \leq m(A=a)$, $J = m(XQA \neq a)$ and p_F have a lower bound $t_J = p_{abs}\binom{n-m(XQ)}{m(A \neq a)}$. Because $\binom{m}{l}$ is an increasing function of m, $t_J \geq p_{abs}\binom{m(\neg X)}{m(A \neq a)}$, which is equal to $\frac{m(\neg X)!m(A=a)!}{n!(m(A=a)-m(X))!}$. □

Theorem 5 *For all $X \subsetneq R,\ A \in R \setminus X,\ a \in \{0,1\}$ and $Q \subseteq R \setminus (X \cup \{A\})$ holds*

$$p_F(XQ \to A=a) \geq p_{abs}\binom{n-m(XA=a)}{m(A \neq a)}.$$

Proof In this proof, we use the fact that for positive dependency rule $X \to A = a$ holds $m(XA=a) > \frac{m(X)m(A=a)}{n}$ (the lift is > 1). We notice that it is enough to show that $p_F(X \to A=a) \geq p_{abs}\binom{n-m(XA=a)}{m(A \neq a)}$, because for any $Q \subseteq R \setminus (X \cup \{A\})$ holds $\binom{n-m(XQA=a)}{m(A \neq a)} \geq \binom{n-m(XA=a)}{m(A \neq a)}$.

Let us notate $p_F = p_{abs}p_X$. Because p_{abs} is a constant, when the consequence is fixed, it can be omitted. For clarity, we present the proof for case $a = 1$. The same result is achieved for $a = 0$, when A and $\neg A$ are reversed.

If $m(X) = m(XA)$, then $p_X(X \to A)$ contains just one term, which is equal to its lower bound. Otherwise, $p_X(X \to A)$ is a sum of several terms, but it is enough to show that for the first term t_0 holds: $t_0 = \binom{m(X)}{m(XA)}\binom{m(\neg X)}{m(\neg X \neg A)} \geq \binom{n-m(XA)}{m(\neg A)} \Leftrightarrow \frac{m(X)!m(\neg A)!}{m(XA)!m(X\neg A)!(m(\neg A)-m(X\neg A))!} \geq \frac{(n-m(XA))!}{m(\neg X)!} \Leftrightarrow [m(X) \cdot \ldots \cdot (m(XA)+1)][m(\neg A) \cdot \ldots \cdot (m(\neg X \neg A)+1)] \geq [(n-m(XA)) \cdot \ldots \cdot (m(\neg X)+1)]m(X\neg A)! \Leftrightarrow \Pi_{i=0}^{m(X\neg A)-1}(m(X)-i)(m(\neg A)-i) \geq \Pi_{i=0}^{m(X\neg A)-1}(n-m(XA)-i)(m(X\neg A)-i)$.

This is true, because for all $i = 0, \ldots, m(X\neg A)-1$ holds $(m(X)-i)(m(\neg A)-i) \geq (n-m(XA)-i)(m(X\neg A)-i) \Leftrightarrow m(X)m(\neg A) - im(X) - im(\neg A) + i^2 \geq m(X)(n-m(XA)) - m(XA)(n-m(XA)) - i(n-m(XA)) - im(X) + im(XA) + i^2 \Leftrightarrow -m(X)m(A) + im(A) + m(X)m(XA) + nm(XA) - m(XA)^2 - 2im(XA) \geq 0$.

Because $nm(XA) > m(X)m(A)$ and $im(A) - 2im(XA) \geq -im(XA)$, it is sufficient to show that $m(X)m(XA) - m(XA)^2 - im(XA) \geq 0 \Leftrightarrow m(XA)m(X\neg A) \geq im(XA) \Leftrightarrow m(X\neg A) \geq i$, which was true. □

References

1. Aggarwal C, Yu P (1998) A new framework for itemset generation. In: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS 1998). ACM Press, New York, pp 18–24

2. Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data. ACM Press, New York, pp 207–216
3. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large data bases (VLDB'94). Morgan Kaufmann, Los Altos, pp 487–499
4. Agresti A (1992) A survey of exact inference for contingency tables. Stat Sci 7(1):131–153
5. Antonie M-L, Zaïane OR (2004) Mining positive and negative association rules: an approach for confined rules. In: Proceedings of the 8th European conference on principles and practice of knowledge discovery in databases (PKDD'04). Springer, Berlin, pp 27–38
6. Blanchard J, Guillet F, Gras R, Briand H (2005) Using information-theoretic measures to assess association rule interestingness. In: Proceedings of the Fifth IEEE international conference on data mining (ICDM'05). IEEE Comput Soc, pp 66–73
7. Borgelt C (2010) Apriori v5.14 software. http://www.borgelt.net/apriori.html. Retrieved 7.6. 2010
8. Cormen T, Leiserson C, Rivest R (1990) Introduction to algorithms. The MIT Press, Cambridge
9. Fisher R (1925) Statistical methods for research workers. Oliver and Boyd, Edinburgh
10. Hämäläinen W (2009) Lift-based search for significant dependencies in dense data sets. In: Proceedings of the workshop on statistical and relational learning in bioinformatics (StReBio'09), in the 15th ACM SIGKDD conference on knowledge discovery and data mining (KDD'09). ACM Press, New York, pp 12–16
11. Hämäläinen W (2010a) Efficient discovery of the top-K optimal dependency rules with Fisher's exact test of significance. In: Proceedings of the 10th IEEE international conference on data mining (ICDM 2010). IEEE Computer Society, Wahington, pp 196–205
12. Hämäläinen W (2010b) Efficient search for statistically significant dependency rules in binary data. PhD thesis, Department of Computer Science, University of Helsinki, Finland. Series of Publications A, Report A-2010-2
13. Hämäläinen W (2010) Statapriori: an efficient algorithm for searching statistically significant association rules. Knowl Inf Syst Int J (KAIS) 23(3):373–399
14. Hämäläinen W, Nykänen M (2008) Efficient discovery of statistically significant association rules. In: Proceedings of the 8th IEEE international conference on data mining (ICDM'08), pp 203–212
15. Koh Y, Pears R (2007) Efficiently finding negative association rules without support threshold. In: Advances in artificial intelligence, proceedings of the 20th Australian joint conference on artificial intelligence (AI 2007), vol 4830 of lecture notes in computer cience. Springer, Berlin, pp 710–714
16. Koh Y, Rountree N, O'Keefe R (2008) Mining interesting imperfectly sporadic rules. Knowl Inf Syst 14(2):179–196
17. Li J (2006) On optimal rule discovery. IEEE Trans Knowl Data Eng 18(4):460–471
18. Liu B, Hsu W, Ma Y (1999) Pruning and summarizing the discovered associations. In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'99). ACM Press, New York, pp 125–134
19. Morishita S, Sese J (2000) Transversing itemset lattices with statistical metric pruning. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS'00). ACM Press, New York, pp 226–236
20. Nijssen S, Guns T, Raedt LD (2009) Correlated itemset mining in ROC space: a constraint programming approach. In: Proceedings the 15th ACM SIGKDD conference on knowledge discovery and data mining (KDD'09). ACM Press, New York, pp 647–656
21. Nijssen S, Kok J (2006) Multi-class correlated pattern mining. In: Proceedings of the 4th international workshop on knowledge discovery in inductive databases, vol 3933 of lecture notes in computer science. Springer, Berlin, pp 165–187
22. Thiruvady D, Webb G (2004) Mining negative rules using GRD. In: Advances in knowledge discovery and data mining, proceedings of the 8th Pacific-Asia conference (PAKDD 2004), vol 3056 of lecture notes in computer science. Springer, Berlin, pp 161–165
23. Webb G (2006) Discovering significant rules. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06). ACM Press, New York, pp 434–443
24. Webb G (2007) Discovering significant patterns. Mach Learn 68(1):1–33
25. Webb G (2008) Layered critical values: a powerful direct-adjustment approach to discovering significant patterns. Mach Learn 71(2–3):307–323
26. Webb G (n.d.) MagnumOpus software. http://www.giwebb.com/index.html. Retrieved 10.2. 2009
27. Webb G, Zhang S (2005) K-optimal rule discovery. Data Mining Knowl Discov 10(1):39–79
28. Wu X, Zhang C, Zhang S (2002) Mining both positive and negative association rules. In: Proceedings of the nineteenth international conference on machine learning (ICML '02). Morgan Kaufmann Publishers Inc., San Francisco, pp 658–665

29. Wu X, Zhang C, Zhang S (2004) Efficient mining of both positive and negative association rules. ACM Trans Inf Syst 22(3):381–405
30. Xiong H, Shekhar S, Tan P-N, Kumar V (2004) Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'04). ACM Press, New York, pp 334–343
31. Yates F (1984) Test of significance for 2 × 2 contingency tables. J Roy Stat Soc Ser A (General) 147(3):426–463
32. Zhang S, Wu X (2011) Fundamentals of association rules in data mining and knowledge discovery. Wiley Interdiscip Rev: Data Mining Knowl Discov 1(2):97–116

Author Biography

Wilhelmiina Hämäläinen received a M.Th. degree from the University of Helsinki, Finland, in 1998, a M.Sc. degree from the University of Helsinki, Finland, in 2002, a Ph.Lic. degree from the University of Joensuu, Finland, 2006, and a Ph.D. degree (Computer Science) from the University of Helsinki, Finland, in 2010. She has worked as teacher, lecturer, and researcher in the Department of Computer Science both in the University of Helsinki and University of Joensuu since 1996. Currently, she is working as a researcher in the Department of Biosciences, University of Eastern Finland. Her interests include statistically sound data mining, machine learning, algorithmics and mathematics.