Finding Trees from Unordered 0–1 Data

Hannes Heikinheimo, Heikki Mannila, and Jouni K. Seppänen

HIIT Basic Research Unit, Lab. Computer and Information Science,
FI-02015 Helsinki University of Technology, Finland
{Hannes.Heikinheimo, Heikki.Mannila, Jouni.Seppanen}@tkk.fi

Abstract. Tree structures are a natural way of describing occurrence relationships between attributes in a dataset. We define a new class of tree patterns for unordered 0–1 data and consider the problem of discovering frequently occurring members of this pattern class. Intuitively, a tree T occurs in a row u of the data, if the attributes of T that occur in u form a subtree of T containing the root. We show that this definition has advantageous properties: only shallow trees have a significant probability of occurring in random data, and the definition allows a simple levelwise algorithm for mining all frequently occurring trees. We demonstrate with empirical results that the method is feasible and that it discovers interesting trees in real data.

1 Introduction

Frequent pattern discovery has been extensively studied, especially in the case of 0–1 data, where various algorithms exist for mining frequent itemsets and association rules. Here we propose a new class of co-occurrence patterns: trees. The idea is to search for hierarchies of general and more specific attributes. For example, consider document data and a tree with attribute A as the root and B and C as the children of A, and D as the child of B (see Figure 1). This means that A is the general concept, B and C are more specific terms related to A, and D is a further specialization of B.

An observed row u follows the hierarchy described by the tree if the attributes in the tree that are 1 in u form a subtree of T containing the root. In the example

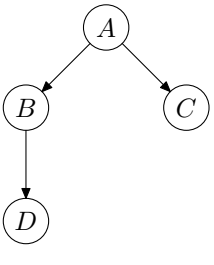

Fig. 1. Example tree

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): PKDD 2006, LNAI 4213, pp. 175–186, 2006.

tree of Figure 1 a row with $u(A) = u(B) = u(C) = 1$ and $u(D) = 0$ satisfies this condition, but a row with $u(A) = u(D) = u(C) = 1$ and $u(B) = 0$ does not.

We consider the task of finding trees T such that there are sufficiently few rows violating the subtree condition (few *conflicts*). To prevent trivial trees consisting only of attributes occurring very rarely, we additionally require that each attribute occurring in T has a high enough frequency in the data. The task we consider is, given two thresholds τ and σ, to find all trees that have at most τ conflicts and each attribute occurring in the tree has frequency at least σ.

While there has been lots of research on finding trees from tree- or graph-structured data (see, e.g., [1,2,3,4,5,6]), the crucial difference is that we start from unstructured 0-1 data, as in mining frequent sets and association rules. Hierarchical clustering or finding phylogenetic trees (see, e.g., [7]) looks for finding trees from 0-1 data, but typically the goal is to find one tree containing all the attributes of the data. The same is true for finding tree-structured Bayes nets from data. Another difference is that in a hierarchical clustering or phylogenetic tree all attributes of the data are in the leaves of the tree. We seek trees where all nodes of the tree are items from the data.

Another somewhat analogous class of patterns are approximate itemsets, such as error-tolerant or dense itemsets [8,9,10]. These are relaxed versions of frequent itemsets: the set is considered to occur in a row provided most of the attributes of the set are equal to 1 in the row. Similarly to these patterns, in this work a tree can be supported by rows that do not have all of the items of the tree. The tree structure reflects more closely the kinds of co-occurrence that are in fact present in the data. Fragments of order [11] are a type of directed itemsets: a fragment is violated by rows having two of its items but lacking at least one item that appear between the two. Fragments of order can be viewed as simple unrooted trees having only one branch.

The rest of this paper is structured as follows. In Section 2 we present the definition of tree patterns. We discuss their theoretical properties in Section 3. In Section 4 we give algorithms for discovering trees, and present two measures for selecting interesting trees. Empirical results demonstrating that the method is feasible and that it finds interesting trees are given in Section 5. Extensions to the pattern class are discussed in Section 6, and Section 7 is a short conclusion.

2 Tree Patterns and 0–1 Data

Let R be a set of 0-1 valued attributes (also called items). A 0-1 dataset D over R is a table of rows of 0s and 1s with R as the set of column headers. The dataset D can be also considered as an unordered multiset of rows u, where each row is a subset of R. We denote attributes by the letters A, B, and C, and by $u(A)$ the value 1 or 0, according to whether the attribute A is present in u or not. We denote by n the number of rows in D and by m the number of attributes, i.e., $|R|$. The *frequency* $f(A)$ of an attribute A is the fraction of rows u such that $u(A) = 1$.

A *tree* is a pair $T = (A, \mathcal{C})$, where A is an attribute and $\mathcal{C} = \{ T_1, T_2, \ldots, T_k \}$ is the set of subtrees, each of which is a tree. (For leaves of the tree the set $\mathcal{C}$

is empty.) We say that A is the root of the tree. Each T_i has the form $T_i = (A_i, \mathcal{C}_i)$; the attributes $A_1, \ldots, A_k$ are called the children of A (this phrasing is unambiguous as we require that each attribute appears in the tree at most once). An attribute B is a descendant of A (or, equivalently, A is an ancestor of B) either if B is a child of A or B is a descendant of a child of A. The set of nodes of the tree T are simply all attributes occurring in the tree.

The *conflict count* $c(T, D)$ of T in D is the number of rows $u \in D$ such that there is a node A and its descendant B such that $u(A) = 0$ and $u(B) = 1$. Given thresholds τ and σ, both in $[0, 1]$, the collection of trees $\mathcal{TP}(D, \tau, \sigma)$ consists of all trees T such that $c(T, D) \leq \tau n$ and $f(A) \geq \sigma n$ for all A occurring in T, where n is the number of rows in the dataset D.

The computational problem we consider in this paper is the following.

Problem 1. Given D, σ, and τ, compute $\mathcal{TP}(D, \tau, \sigma)$.

Note that in the definition of $\mathcal{TP}(D, \tau, \sigma)$ we use an *upper* bound τ on the number of conflicts, while frequent patterns typically are defined by a *lower* bound on the number of occurrences. Our parameter σ has this role, preventing attributes with very low frequency from being considered.

3 Basic Properties of Tree Patterns

In this section we consider the basic properties of tree patterns and the pattern collection $\mathcal{TP}(D, \tau, \sigma)$.

Monotonicity. The first observation is simple monotonicity property typical for frequent patterns. A tree S is a rooted subtree of tree T, if S can be obtained from T by a series of removals of leaves. The following proposition is immediate.

Proposition 1. *The pattern class* $\mathcal{TP}(D,\tau,\sigma)$ *is monotone with respect to rooted subtrees, i.e., if* $T \in \mathcal{TP}(D, \tau, \sigma)$ *and* S *is a subtree of* T*, then* $S \in \mathcal{TP}(D, \tau, \sigma)$.

Trees and Association Rules. Next we discuss the relationship of trees and association rules. For a simple tree T containing root A and a child B, the conflict count $c(T)$ is $n(f(B) - f(AB))$, where $f(AB)$ is the frequency of the attribute set AB, i.e., the relative number of rows u with $u(A) = u(B) = 1$. Noting that $\gamma = f(AB)/f(B)$ is the accuracy (confidence) of the association rule $B \to A$, we have that $c(T) = nf(B)(1 - \gamma)$. One can ask whether other, more complex trees could be reduced to association rules? Could we perhaps find all the trees in $\mathcal{TP}(D, \tau, \sigma)$ just by postprocessing a set of association rules between attributes? (See, e.g., [12,13,14,15,16] for interesting work on postprocessing collections of association rules.) This turns out not to be the case, however.

A row u satisfies the subtree condition for tree T if the rule $C \to D$ is true on u for all pairs (C, D) such that C is a descendant of D in T. However, there is no simple formula for the conflict count of a tree T given the accuracies of the rules $C \to D$ for attributes occurring in T. The reason is that a tree conflicts with a row u if at least one rule is violated: the frequency under which this happens

depends on the interaction of the different rules. As an example, consider the tree with root A, and B and C as the children of A. If $f(A) = f(B) = f(C) = 0.2$ and $f(AB) = f(AC) = 0.1$, the accuracies of the rules $B \to A$ and $C \to A$ are both 0.5. The conflict count of the tree T can, however, vary between $0.2n0.5$ and $2(0.2n0.5)$, depending on whether the conflicts are on the same rows or not. Hence there is no algorithm for computing $\mathcal{TP}(D, \tau, \sigma)$ that would take as input only the association rules between attributes.[1]

Number of Possible Trees. The number of rooted labeled trees on m vertices is m^{m-1}. This follows from a theorem of Cayley, which states that the number of labeled trees is m^{m-2}; see e.g. [17, Section 3.3] or [18, sequence A000169]. The number of possible roots is of course m, from which the result follows. This implies that it would be infeasible to consider all trees even for moderate values of m.

Trees in Random Data. We next address the question how many trees are in the collection $\mathcal{TP}(D, \tau, \sigma)$ in random data, similarly to the discussion in [19] on frequent itemsets. Suppose D contains independent and identically distributed entries, with probability p of a 1 for each entry. Trees that only have a root and k children cause a conflict only if the value of the attribute of the root is 0; thus the conflict probability is fairly low.

For trees with longer branches it is straightforward to demonstrate that the probability q of a conflict grows fast. Using Chernoff bounds it is then easy to show that the probability that a tree with longer branches has less than, say, $nq/2$ conflicts is at most $\exp(-cnq)$ for some constant c. There are $P = \binom{m}{k} k^{k-1}$ trees with k nodes selected from the set of m attributes, implying that the expected number of trees with less than $nq/2$ conflicts is thus bounded by $P \exp(-cnq)$. We have $\log P \leq k(\log m + \log k) \leq 2k \log m$. Thus, if $2k \log m \leq cnq$, the expected number of trees (with sufficiently long branches) in $\mathcal{TP}(D, \tau, \sigma)$ is at most 1. We omit the details.

4 Generating the Collection $\mathcal{TP}(D, \tau, \sigma)$

4.1 The Levelwise Algorithm

Proposition 1 allows using a standard levelwise algorithm for computing trees in $\mathcal{TP}(D, \tau, \sigma)$, as when computing frequent itemsets: start from single attributes, and on every pass combine trees of size k into trees of size $k + 1$. However, the combination phase is not as simple as for itemsets.

One approach would be to try adding every attribute into every possible position in each tree, but with a large number of attributes this would force the algorithm to consider prohibitively many candidate trees. Another possibility is to try combining all pairs of trees, which takes quadratic time in the number

[1] The exponential collection of frequencies of *all* frequent sets for frequency threshold 0 specify the distribution of the data rows uniquely, so that exponential input would suffice to determine also the collection $\mathcal{TP}(D, \tau, \sigma)$.

of trees. Instead, we use the approach of Zaki [6]. Briefly, a tree is represented as a string by traversing it depth-first in preorder, recording the attribute in each node, and -1 when backtracking. For example, the tree in Figure 1 would be encoded as $(A, B, D, -1, -1, C, -1)$. In this encoding, it is sufficient to consider combining pairs of trees sharing the same $(k-1)$-prefix, which limits the quadratic behavior to much smaller sets than the complete set of k-trees. For details, see [6]; note that since we restrict attributes to occur at most once in each tree, not all combinations listed by Zaki are needed.

A drawback of the combination method is that each $(k+1)$-tree is generated multiple times as isomorphic copies: for example, the isomorphic trees $(A, B, -1, C, -1)$ and $(A, C, -1, B, -1)$ get both generated. We optimize the database pass by only accessing the database for trees where the children of each node are in alphabetical order, and using the same information for isomorphic trees. However, it is not possible to completely prune the copies. For example, the 4-tree $(A, C, -1, D, B, -1, -1)$ can only be generated from the 3-trees $(A, C, -1, D, -1)$ and $(A, C, -1, B, -1)$, the latter of which does not have the order property. Another possibility would be to work only with some canonical forms of trees, as in [4].

The size of the class $\mathcal{TP}(D, \tau, \sigma)$ is highly sensitive to the values of the two parameters. A way to ameliorate this problem is to use a top-k algorithm similar to that developed for dense itemsets [10].

We remarked in the previous section that in random data trees of small depth can have low conflict count. It is straightforward to modify the above algorithm to construct only, e.g., binary trees, thus guaranteeing longer branches. We omit the details.

4.2 Selecting Interesting Trees

The collection $\mathcal{TP}(D, \tau, \sigma)$ will in many cases be quite large, and tools are needed for selecting the most interesting trees.

We present two measures for selecting interesting trees: specificity and conflict ratio. The *specificity* $\phi(T)$ of a tree T is the size of the transitive closure of the tree, when it is viewed as a relation on the set of attributes. In other words, it is the number of (ancestor, descendant) pairs in the tree. A single-branch tree has maximal specificity and a shallow tree whose leaves are the children of the root has minimal specificity. The *conflict ratio* $E[c(T)]/c(T)$ of a tree T is obtained by comparing the number $c(T)$ of conflicting rows in the data to its expectation $E[c(T)]$ under the assumption that the attributes are independent and have the marginal frequencies observed in the data.

The expectation $E[c(T)]$ can be computed recursively for each tree. For $T = (A, \mathcal{C})$, the probability of no conflict, under the independence assumption, is

$$\begin{aligned} 1 - \Pr(u \text{ conflicts with } T) = {} & \Pr(u(A) = 0) \prod_{B} \Pr(u(B) = 0) \\ & + \Pr(u(A) = 1) \prod_{S \in \mathcal{C}} (1 - \Pr(u \text{ conflicts with } S)), \end{aligned}$$

where the first product is taken over all attributes B represented in T, except for the root A, and the second over all child-trees S of T.

The conflict ratio will be high for trees that have much fewer conflicts than would be expected under the independence assumption, i.e., trees that capture interesting co-occurrence patterns in the data. The two interestingness measures can be used to rank the trees in various ways.

5 Experiments

In this section we report on the experimental results we have obtained on generated and real data. Due to space constraints we discuss the results only briefly.

5.1 Generated Data

We generated data using the following procedure. First, a number of disjoint trees with different values of the specificity measure were created by hand. The number of trees used for the experiment was varied by taking different subcollections of the trees. Given such a collection $\mathcal{S}$, data was produced as follows. A row u was generated by first making all attributes of u equal to 0. Then, each tree $T \in \mathcal{S}$ was selected with probability p. If T was selected, we sampled a subset X of the nodes of T by taking each node with probability q. Letting Y be the set of all ancestors of nodes in X, we let $u(A) = 1$ for all $A \in X \cup Y$. Finally, each bit in the dataset was flipped independently with probability r to create noise. The parameter values used were $p = q = 0.5$ and $r = 0.1$, and 1000 data rows were generated.

From the generated data, trees were mined using a Java implementation of the levelwise algorithm described in Section 4. The parameter σ was chosen to be 0.2, since each tree has a $p = 0.5$ chance of occurring, and each attribute in the tree may have as low as a $pq = 0.25$ chance of occurring. The parameter τ was chosen as large as possible so that a reasonable number of trees was still obtained; typically $\tau = 0.3$ was close to the limit. Selecting the parameters was also the reason that only disjoint trees were considered. With overlapping trees large numbers of shallow subtrees are generated, and the conflict threshold has to be quite low. This prevents the discovery of the more interesting trees. With disjoint trees, the result set $\mathcal{TP}$ was reasonably small, while containing all the trees in $\mathcal{S}$. In order to test how well these trees were positioned in the results with respect to the two interestingness measures, the mined trees were partitioned into classes by their specificity ϕ, and each class was sorted by the conflict ratio $E[c]/c$. From each specificity class, trees were selected into the final result set $\mathcal{R}$ in decreasing order of conflict ratio until all trees in $\mathcal{S}$ had been selected. The size $|\mathcal{R}|$ of the final result set is thus a measure of how close to the top the generating trees in $\mathcal{S}$ were. The results are shown in Table 1.

We see that in every case the number of trees one needs to examine in order to find the generating trees is fairly low. In fact, most of the extra trees are variations of the generating trees: for example, in the smallest data set, one of

the two generating trees is found immediately, and three of its simple variants precede the other generating tree in conflict ratio order. The sensitivity to the parameter is evident in that while we have $\sigma = 0.2$ and $\tau = 0.3$ in all the cases shown the size of the output $|\mathcal{TP}|$ varies non-monotonically in m.

5.2 Real Data

We used three real datasets in our experiments: data about terms used in NFS abstracts [20], a database about students and courses at the Computer Science Department of the University of Helsinki, and paleontological data [21,22]. We present the results only for the first two data sets; the results were similar for the third one.

Abstracts Data. The data set [20] consists of 128820 abstracts describing NSF awards for basic research. The observations correspond to the abstracts and the variables correspond to the terms occurring in them. For preprocessing we applied the Porter Stemming Algorithm [23] to merge variables corresponding to terms with a common stem. In addition, we reduced the dataset by taking a random 2% sample of the observations and choosing 66 subjectively interesting stem terms[2] for the experiments. The final preprocessed data set consisted of 2510 observations (rows) and 66 variables (columns), with a total of 11262 entries of 1s and an average of 4.5 1s per row.

Table 2 shows the number of trees obtained for different conflict thresholds τ and frequency thresholds σ. We see that the number of elements in the answer increases rapidly with decreasing frequency threshold σ and increasing conflict threshold τ. One should observe, however, that it is quite easy to iteratively find values of σ and τ that produce outputs of desired size.

When inspected visually, the resulting trees look intuitive. As an example, we found the tree depicted in Figure 2(a), at a conflict count frequency of 13.1% and a conflict ratio of 1.58. With another set of parameters, we found the very intuitive tree in Figure 2(b) at a conflict count frequency of 21.2% and a conflict ratio of 1.19. The tree in Figure 2(a) has specificity 5, whereas the tree in Figure 2(b) has specificity 6.

Course Enrollment Data. The data consists of course enrollment records for courses held at the Department of Computer Science at the University of Helsinki. The data set has 3506 observations corresponding to students and 98 variables corresponding to courses. The mean number of 1s per row is 4.6. The

[2] The terms chosen were algebra, algorithm, atom, behavior, biolog, carbon, cell, cellular, channel, chemistri, climat, code, comput, distribut, dna, document, earth, ecolog, ecosystem, educ, electron, energi, environment, enzym, evolut, genet, geolog, hardwar, internet, isotop, life, light, link, magnet, materi, mathemat, matter, metal, molecular, morpholog, natur, network, nonlinear, nuclear, numer, ocean, oxid, physic, pi, plasma, protein, quantum, record, scienc, semiconductor, social, softwar, statist, surfac, temperatur, theoret, topolog, transit, transport, water and web.

Table 1. Results on generated data. m: number of attributes; $|\mathcal{S}|$: number of trees used in the generating process; σ, τ: thresholds for frequency and conflicts; $|\mathcal{TP}|$: size of the output set; $|\mathcal{R}|$: size of the final result set obtained by taking enough trees to cover $\mathcal{S}$.

m	$\lvert\mathcal{S}\rvert$	σ	τ	$\lvert\mathcal{TP}\rvert$	$\lvert\mathcal{R}\rvert$
10	2	0.2	0.30	1343	5
14	3	0.2	0.30	1172	8
18	4	0.2	0.30	1965	21
20	5	0.2	0.30	2208	27
23	6	0.2	0.30	1674	45
28	7	0.2	0.30	5469	43

Table 2. Results for the abstracts data set. The number of trees in the collection $\mathcal{TP}(D, \tau, \sigma)$ for various values of τ and σ. $k = \max |T|$, the largest tree; $h = \max \phi(T)$ the maximal specificity.

τ	σ	$\lvert\mathcal{TP}\rvert$	k	h	cand	time/sec.
0.14	0.12	26	3	2	194	0.573
0.16	0.12	33	3	3	250	0.603
0.18	0.12	51	3	3	397	0.835
0.14	0.08	649	3	3	12514	5.284
0.16	0.08	1894	3	3	39091	19.22
0.18	0.08	3890	4	4	108825	96.05
0.14	0.06	6637	5	6	204741	339.1
0.16	0.06	14927	5	7	538334	1970
0.18	0.06	48176	6	7	1683841	30300

total number of 1s in the data is 16086. Table 5.2 shows the number of trees obtained for different conflict and frequency thresholds τ and σ.

For the course enrollment data there is an ordering in which the department recommends the students to take some of the courses. For instance, some courses require only basic understanding of programming concepts, whereas some of the courses have more specific prerequisites. As an example of this, we found the tree depicted in Figure 3 at a conflict threshold of 16.3% and a high conflict ratio of 2.20. The tree reflects the fact that the more advanced courses Data structures and Programming in C have Java programming as a prerequisite, whereas for Computer Organization the course Introduction to Programming suffices. This is also the order in which the department recommends these courses to be taken.

Outlier Detection. To evaluate the usefulness of discovered trees we experimented with their use in outlier detection. An observation that conflicts with several of the strongest tree patterns is likely to be an outlier. As a test of this, we performed the following experiment. We took the course enrollment data set,

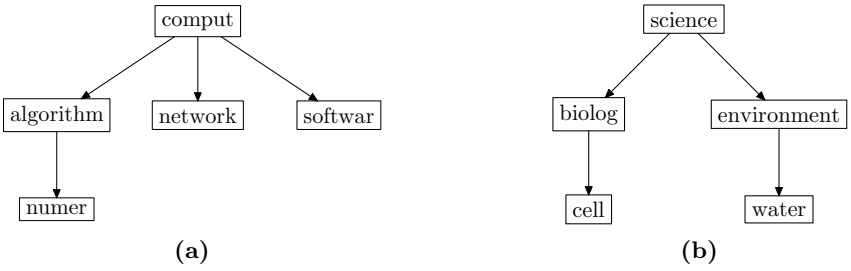

Fig. 2. Two example trees in abstracts data

Table 3. Results for the course enrollment data. The number of trees in the collection $\mathcal{TP}(D, \tau, \sigma)$ for various values of τ and σ. $k = \max |T|$, the largest tree; $h = \max \phi(T)$ the maximal specificity.

τ	σ	$\lvert\mathcal{TP}\rvert$	k	h	cand	time/sec.
0.14	0.12	68	3	3	581	0.524
0.16	0.12	124	4	3	991	0.795
0.18	0.12	220	4	5	1830	1.163
0.14	0.1	262	4	5	1929	1.318
0.16	0.1	484	5	6	4128	2.733
0.18	0.1	998	5	7	9367	6.373
0.14	0.08	3955	6	8	71982	145.6
0.16	0.08	11475	7	10	281398	3310
0.18	0.08	35398	8	12	1221143	93740

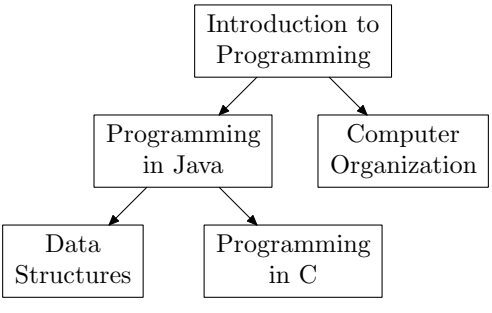

Fig. 3. Example tree found in course enrollment data

here denoted by D, and generated an additional data set N with independent attributes with the same marginal frequencies as in D. The size of N was 5% of the original data D. Then, frequent trees $\mathcal{TP} = \mathcal{TP}(E, \tau, \sigma)$ were mined from the augmented data $E = D \cup N$ with $\tau = 0.14$ and $\sigma = 0.08$. From each specificity class, the 30 trees with maximal conflict ratio were selected to form

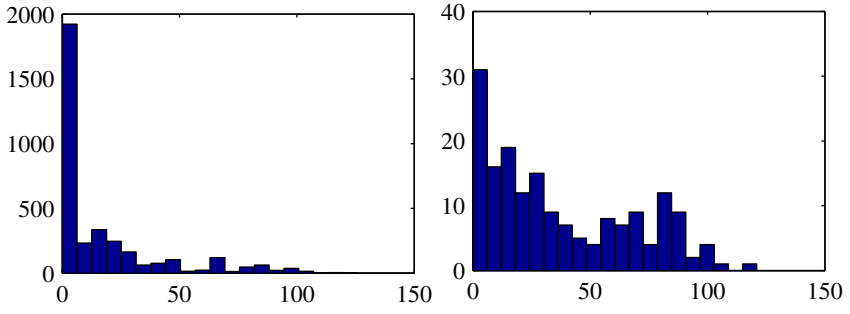

Fig. 4. Histogram for the conflict count between the rows in the data and the 164 generated trees. The figure on the left depicts the histogram for the real rows in the data (total 3506). The figure on the right depicts the histogram for the 5% added noise rows (total 175). The original data rows have fewer conflicts with the chosen 164 trees than the added rows. See the text for details.

a subset $\mathcal{S} \subset \mathcal{TP}$. The specificities ranged from 1 to 6, but since the generated number of trees with specificity 5 and 6 was 23 and 21 a total of 164 trees were selected to $\mathcal{S}$. For each row $u \in E$, we determined how many trees $T \in \mathcal{S}$ conflict with u.

The result was that for rows $u \in D$, the average conflict count was 16.8, with a standard deviation of 24.5, and for $u \in N$, the average was 37.8 with a standard deviation of 31.2. Figure 4 shows the histograms of the number of conflicts per row for the real data D and for the added rows N. Thus, the noise rows behave clearly differently from the real data rows from the viewpoint of tree patterns.

6 Extensions

A problem clearly seen in the experiments is that a large number of small, shallow trees crop up and slow the algorithm down before it has investigated more interesting trees. A possible solution is to restrict the class of trees considered. For example, if the number of children of each node is restricted to be at most two, a levelwise algorithm would still be possible, and there would be much fewer shallow trees to consider.

Another way to approach the problem is to change the search strategy from the levelwise, breadth-first search. In the case of frequent itemsets, there are depth-first algorithms for mining the maximal frequent itemsets without considering all their subsets (see, e.g, [24] and [25]). Adapting such algorithms for trees is an interesting direction for future research.

If more efficient search strategies are developed, it would also be interesting to broaden the class of patterns: directed acyclic graphs would be a natural generalization of trees.

7 Conclusion

We have introduced the idea of mining trees from unordered 0–1 data and shown that this pattern class is distinct from traditional frequent itemsets or association rules. We have shown empirically that the levelwise algorithm can find interesting trees in both generated and real data. In real data, our experiments show that there are interesting co-occurrence patterns that are naturally captured as trees.

References

1. Chi, Y., Muntz, R.R., Nijssen, S., Kok, J.N.: Frequent subtree mining – an overview. Fundamenta Informaticae **66** (2005) 161–198
2. Chi, Y., Yang, Y., Muntz, R.R.: Indexing and mining free trees. In: Proceedings of the Third IEEE International Conference on Data Mining (ICDM). (2003) 509 – 512
3. Chi, Y., Yang, Y., Muntz, R.R.: Mining frequent rooted trees and free trees using canonical forms. Technical Report CSD-TR No. 030043, UCLA Computer Science Department (2003) `ftp://ftp.cs.ucla.edu/tech-report/2003-reports/030043.pdf`.
4. Chi, Y., Yang, Y., Muntz, R.R.: HybridTreeMiner: an efficient algorithm for mining frequent rooted trees and free trees using canonical forms. In: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM). (2004) 11 – 20
5. Nijssen, S., Kok, J.N.: Efficient discovery of frequent unordered trees. In: First International Workshop on Mining Graphs, Trees and Sequneces (MGST). (2003) 55 – 64
6. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (2002) 71 – 80
7. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc., Sunderland, MA (2004)
8. Pei, J., Tung, A.K., Han, J.: Fault-tolerant frequent pattern mining: Problems and challenges. In: Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD). (2001) 7–12
9. Yang, C., Fayyad, U., Bradley, P.S.: Efficient discovery of error-tolerant frequent itemsets in high dimensions. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (2001) 194 – 203
10. Seppänen, J.K., Mannila, H.: Dense itemsets. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (2004) 683–688
11. Gionis, A., Kujala, T., Mannila, H.: Fragments of order. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (2003) 129–136
12. Tuzhilin, A., Adomavicius, G.: Handling very large numbers of association rules in the analysis of microarray data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (2002) 396–404
13. Lent, B., Swami, A.N., Widom, J.: Clustering association rules. In: Proceedings of the 13th International Conference on Data Engineering (ICDE). (1997) 220–231

14. Liu, B., Hsu, W., Ma, Y.: Pruning and summarizing the discovered associations. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (1999) 125–134
15. Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., Verkamo, A.I.: Finding interesting rules from large sets of discovered association rules. In: Proceedings of the Third International Conference on Information and Knowledge Management (CIKM). (1994) 401–407
16. Jaroszewicz, S., Simovici, D.A.: Pruning redundant association rules using maximum entropy principle. In: Proceedings of the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD). (2002) 135–147
17. Kreher, D.L., Stinson, D.R.: Combinatorial Algorithms: Generation, Enumeration and Search. Discrete mathematics and its applications. CRC Press (1999)
18. Sloane, N.J.A.: The on-line encyclopedia of integer sequences (2006) `http://www.research.att.com/~njas/sequences/`.
19. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining. AAAI Press (1996) 307–328
20. Hettich, S., Bay, S.D.: The UCI KDD archive (1999) Irvine, CA: University of California, Department of Information and Computer Science. `http://kdd.ics.uci.edu`.
21. Fortelius, M. (coordinator): Neogene of the old world database of fossil mammals (NOW) (2006) University of Helsinki. `http://www.helsinki.fi/science/now/`.
22. Fortelius, M., Gionis, A., Jernvall, J., Mannila, H.: Spectral ordering and biochronology of european fossil mammals. Paleobiology **32** (2006) 206–214
23. Porter, M.F.: An algorithm for suffix stripping. Program **14** (1980) 130–137
24. Bayardo, R.: Efficiently mining long patterns from databases. In: Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD). (1998) 85–93
25. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000)