Tree-Growth Based Sequential and Associative Pattern Discovery

M. Baumgarten, A.G. Büchner, J.G. Hughes
Northern Ireland Knowledge Engineering Laboratory
European Centre of Excellence for Web Mining
Faculty of Informatics, University of Ulster, Belfast, BT37 0QB, UK
Phone: +44 (0)28 9036 8394 Fax: +44 (0)28 9036 6068
{m.baumgarten, ag.buchner, jg.hughes}@ulster.ac.uk

Mining for frequent patterns is an active research area in which numerous algorithms have been proposed to discover different types of patterns based on associative and sequential structures. However, almost all methods are designed to discover a single type of pattern rather than a user selected combination. Within this paper, a novel, flexible algorithm called S&AD is presented, which is capable of discovering a multitude of inter-field and inter-record patterns. In addition the discovery of sequential and associative patterns is supported where the methods proposed for discovery only consist of a recognition phase, thus avoiding the time expensive task of candidate generation. The structure further allows the incorporation of user-driven constraints and domain knowledge at different levels of the discovery process.

Keywords: pattern mining, pattern discovery, association, sequential pattern

1 Introduction

Mining databases to discover patterns that are of interest has been a challenge for a vast number of researchers from various domains. Different techniques have been introduced to discover multiple types of patterns and their utilization within different domains. Two of the most important types are associations and sequences which can be used for various scenarios, such as basket analysis and behaviour analysis.

Traditional pattern discovery has concentrated on a single dimension to discover so called inter-field patterns. These are patterns in which all items belong to the same object, transaction, tuple etc. Recent endeavours have included a second dimension to discover patterns across objects, known as inter-record patterns. This essentially allows the discovery of patterns containing patterns per se, rather than single items. This is of particular interest when analyzing repeating actions of the same object. The data structure essentially contains two different yet related dimensions that need to be analyzed in tandem. Being able to select the desired type of pattern for each dimension enables the discovery of targeted pattern structures tailored for a given scenario.

The scope of this research is split into two parts. Firstly, the design and development of an efficient tree-based structure capable to store different types of patterns across two dimensions. Secondly, the development of an advanced discovery method efficiently discovering multiple types of patterns. An algorithm, called S&AD (Sequential & Associative Pattern Discovery) is introduced capable of discovering such associative and sequential based patterns across multiple dimensions. Avoiding time consuming candidate generation the proposed technique only consists of a recognition phase extending a structural representation of all *k*-patterns, found at a given state during the discovery process.

The paper is organized as follows. In Section 2, related work is reviewed and drawbacks are shown. Within Section 3, preliminary definitions are outlined and a tree structure is introduced to efficiently store patterns. Section 4 describes the algorithm and the incorporation of different types of domain knowledge. In Section 5 some experiments are performed and evaluated before Section 6 concludes the paper and outlines future work.

2 Related Work

Early approaches to discover frequent patterns are mostly based on traditional apriori-based algorithms, as presented in [1] and [2], which generate candidates to build up patterns in an iterative fashion. Such methods usually generate candidates of length *n* and then determine their occurrence through a recognition step. Due to the problems that are associated with each phase, traditional candidate generation methods can be very costly.

[4] introduces a method called *frequent pattern growth* that effectively mines patterns without candidate generation. This is done by utilizing methods which preserve the essential grouping of original data elements that are used for mining the desired patterns. The analysis phase focuses on counting the occurrence of the relevant data sets. A divide-and-conquer methodology is introduced to reduce the search space through the partitioning of the original data set.

In [5] a general formulation is introduced that unifies different pattern types and defines different counting methods to provide a more general representation of patterns, which allows the discovery of different pattern types.

Thresholds, in the form of numeric minima and maxima, are used during the discovery process to bias the search and the result space; such constraints are provided by almost all detection mechanisms [6]. Another way to bias the search is through the use of *domain* or *background knowledge.* As outlined in [8] the use of such knowledge is to guide and constrain the search for interesting patterns and knowledge. Popular types of domain knowledge are taxonomies [7] to discover generalized sequences and templates to specify the desired pattern [3]. However, most of the current discovery systems implement constraints and / or domain knowledge only on one dimension or on the overall pattern, respectively. This clearly limits the flexibility to search for patterns satisfying different constraints on different dimensions.

The proposed algorithm overcomes this limitation by introducing a separate constraint and domain knowledge model for both inter-field and inter-record patterns. This also enables the handling of different pattern types for the different dimensions and the incorporation of separate domain knowledge.

3 Preliminaries

Source Data: The input data set can be characterized by four columns that build up the necessary structure for sequential or associative patterns. The first two columns are compulsory and contain a primary identifier (PID), splitting the data base into sets and a secondary identifier (SID), splitting each set into a number of sub-sets, as shown in Table 1. For instance, a customer (PID) and its shopping baskets (SID) bought during a given period. The last two columns (item and time) represent the item itself and a related timestamp which usually reflects the time of occurrence and therefore provides an order on the items. The timestamp is only relevant for patterns following a sequential structure; it can be left out for associative patterns. The nature of the input set can be defined as follows:

Items: Let E be a class of elementary item types, then an item is defined as a pair $p = (e, t)$, where $e \in E$ and t is a time related statement. An example is $p_n = (\textit{Milk}, \text{December:5:2002:22:34:11})$.

Data base: The database D contains a number of sets and is defined as $D = \{S_1, S_2, S_3, \dots\}$, where each $S \in D$ contains a start time S_t, an end time S^t and a number of objects Q such that $S = (S_t, S^t, Q_1, Q_2, Q_3, \dots)$. Each Q is defined as $Q = (Q_t, Q^t, p_1, p_2, p_3, \dots)$, where Q_t, and Q^t represent the object's start and end time and each p represents an item.

An example data base is shown in Table 1. In essence this structure contains two dimensions identified by PID and SID. Former splits the data into a number of sets, S, identifying a single object, while latter divides each of these sets into a number of sub objects, Q, containing a list of items. Contained patterns are identified as *inter-record patterns* Φ^R, where the content belongs to different Q's of a given S and *inter-field patterns* Φ^I where the content belongs to the same Q for a given S.

PID	SID	Item	Time
S_1	Q_1	A	May:5:2002:22:34:00
		C	May:5:2002:22:44:00
		B	May:5:2002:22:50:00
	Q_2	D	May:5:2002:22:54:00
		B	May:5:2002:23:04:00
		C	May:5:2002:23:14:00
S_2	Q_1	A	May:5:2002:23:11:00
		B	May:5:2002:23:15:00
	Q_2	D	May:6:2002:23:10:00

Table 1: Example Data Base

Pattern Types: In general, patterns can be defined as a collection of items, following a certain structure. Such a pattern becomes interesting if it fulfils given constraints such as minimum support or confidence. Two basic pattern types are sequences and associations.

Sequential Patterns Φ_S are defined as an ordered list of items which occur within a given data set. For such pattern the order of occurrence is important, such that $(A,B) \neq (B,A)$.

Associative Patterns Φ_A are defined as a set of items which occur within a given data set independent of their order or occurrence, such that $(A,B) = (B,A)$.
Depending on the combination of pattern types selected for inter-field and inter-record patterns different overall patterns Φ can be derived. To indicate the "contained in" relationship of a pattern the symbol $|=$ is used.

Overall Pattern Φ**:** Being able to choose the type of pattern for Φ^R and Φ^I allows four possible combinations for the overall pattern indicating the flexibility of the proposed approach.

Frequent Pattern: A pattern is called frequent if it occurs frequently enough, based on a user defined threshold, within a given data set. For inter-field patterns the frequency f_I is calculated based on the number of Q's, such that $f_I = o_I / |Q|$, where o_I represents in how many Q's a pattern occurs. For inter-record patterns the frequency f_R is calculated based on the number of sets in the data base, such that $f_R = o_R / |S|$, where o_R represents the occurrence of a pattern.

Pattern Representation: Taking advantage of the fact that patterns in general can be represented as directed acyclic graphs, a tree structure is used combining sub-patterns into single branches and therefore minimizing memory usage and processing efforts. Such a structure can be used for both associative and sequential based pattern types because it provides a structure that preserves the order of items as required for sequential structures, whereas for associative structures this order can be ignored.

In order to discover both inter-field and inter-record patterns, one tree is introduced for each type. T^I holds the type of patterns selected for inter-field patterns where, each node represents an item itself. T^R holds the type of patterns selected for inter-record patterns as developed from T^I. Instead of storing items or sets of items, each node of T^R represents a reference to a node of T^I. Patterns can then be resolved using both the structure of T^I and T^R. An example for T^I and T^R, is shown in Figure 1 (based on Table 1 but representing only a fraction of the pattern contained in the example sets), where T^R is build upon T^I. Each node includes a node element containing a reference to an item of T^I, a reference to a node in T^I of T^R and a node count containing the occurrence of a pattern. The numeric values next to each node in T^I are used to visualize the nodes internal ID which is used to build T^R. To resolve a pattern in T^R the references of each node have to be resolved. For example, the node marked in T^R represents the pattern (A) (B, C) also marked in Table 1.

4 <u>S</u>equential <u>&</u> <u>A</u>ssociative Pattern <u>D</u>iscovery

Given a data base *D*, a pattern type and minimum frequency value for both inter-field and inter-record pattern, the goal is to find all frequent patterns Φ that are contained in *D* and satisfy the given constraints.

To discover patterns along both dimensions of the input set, the algorithm is split into two phases. Firstly, D^I, (lines 1 to 11 of Algorithm 1) is designed to discover all inter-field patterns of a selected type. This builds T^I which is required for the next phase. Within second phase, D^R, (lines 13 to 20), inter-record patterns are discovered based on the inter-field patterns discovered previously. After T^R is fully deployed it contains a set of both inter-field and inter-record patterns, thus representing the result set. The general construction of either of the trees does not depend on the selected pattern type but some specific constraints have to be satisfied to recognize the different patterns within the data set and to overlay them onto the tree structure.

```
à  D, f_I min, f_R min, pattern type for T^I and T^R.

      Sort elements in Q  // associative patterns only

D^I   1)      i := 0;
      2)      while (T^I is active) do
      3)          while(D.hasNextSet) do
      4)            while(S.hasNext_Q) do
      5)              extend T^I_{i+1} with (Q |= Φ^I) of size i +1
                      and set frequency f
      6)            od;
      7)          od;
      8)          remove nodes of T^I_{i+1} where f_Φ < f_I min. and
                  disable non-extendable branches
      9)          i := i + 1;
      10)     od;
      11)   // T^I is fully deployed and contains all Φ^I

D^R   12)     i := 0;
      13)     while (T^R is active) do
      14)         while(D.hasNextSet) do
      15)             S'_k = T^I (S_k |= Φ^I)
      16)             extend T^R_{i+1} with (S'_k |= Φ^R) of size i +1
                      and set frequency f_Φ
      17)         od;
      18)         remove nodes of T^R_{i+1} where f_Φ < f_R min. and
                  disable non-extendable branches
      19)         i := i + 1;
      20)     od;

ß  T^R containing all Φ
```

Algorithm 1: The S&AD Algorithm

4.1 Building Inter-Field Patterns

Building inter-field patterns concentrates on the discovery of patterns which contain single items. Therefore the input required for this phase is *Q* rather than *S*. For associative patterns all *Q*'s need to be sorted alphanumerically to guarantee the same order of items among all *Q*'s. This allows using the same discovery method as for sequential patterns. D^I extends iteratively the depth of T^I alternating between building and pruning phases that first build a new level on T^I discovering all patterns of size $i + 1$ or updates relevant occurrence values if a pattern already exists. After the entire data set has been scanned, all nodes that do not support $f_{I\,\min}$ are removed and branches that can no longer be extended are disabled. This process continues until the root node of the tree is disabled and T^I is marked as inactive.

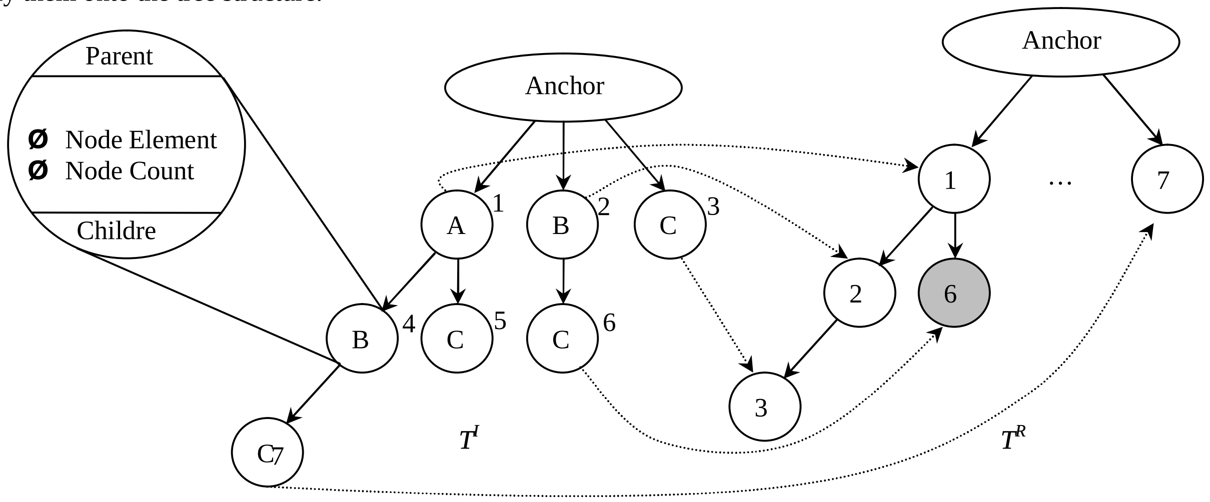

Figure 1: Example T^I and T

The root node of T^I is an anchor node not containing any values. To update occurrences or to extend a tree node the values of each Q are used and recursively overlaid onto the current tree structure. If there is a set of nodes forming a pattern of equal items and structure on T^I of size i, then the leaf node is extended with all remaining items in Q forming new patterns of size $i + 1$. If there already exists a pattern of size $i + 1$ then the node count is incremented. Reoccurring patterns within the same Q are only counted once. Figure 2 shows a fully deployed T^I based on the example set S_1 shown in Table 1. Nodes that are marked occur in both Q's. Note, that there exists no pattern (C, B) because this is covered by (B, C) = (C, B). Each node shown in Figure 2 has a numeric value attached to it representing a unique reference, which is required to construct T^R as described next.

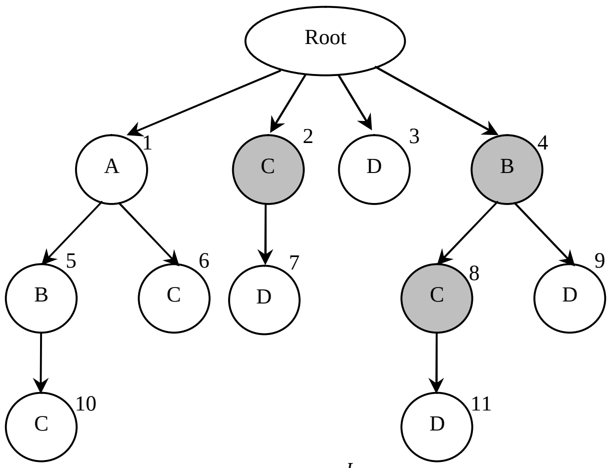

Figure 2: Example T^I (Associative)

4.2 Building Inter-Record Patterns

D^R works in a similar fashion as D^I, alternating between building and pruning phases to deploy T^R. However, inter-record patterns are built upon the results of the first discovery phase, that is T^R is deployed on the basis of T^I. This requires the re-discovery of all patterns that are contained in any Q of a given S, resulting in $S' = (R_1, R_2, R_3, \ldots)$, where each R is defined by the tuple (r, [Q]), where r is the node reference itself and [Q] represents a set of object identifiers where r occurres in, such as $[Q] = [Q_1, Q_2, Q_3, \ldots]$. For associative patterns S' also needs to be sorted by r to use the same discovery mechanism as for sequential based patterns. Figure 3 shows S' based on the example set S_1 given in Table 1.

$$S' = ([1, Q_1], [2, Q_1, Q_2], [4, Q_1, Q_2], [5, Q_1], [6, Q_1], [8, Q_1, Q_2], [10, Q_1], [3, Q_2], [7, Q_2], [9, Q_2], [11, Q_2])$$

Figure 3: S' based on S_1 of Table 1

Updating the tree structure to extend the depth of T^R building new patterns of size $i + 1$ (or to update the node counter) is similar to the update process of D^I. However, instead of discovering patterns on the original set structure the representation of S' is used and the following conditions need to be satisfied. For

ü Φ_S: For all newly added nodes the index k of at least one of the object identifiers in [Q] has to be greater than the index of the object identifier of the leaf node;

ü Φ_A: The index k of at least one of the object identifiers in [Q] of the new node has to be unequal to all other object identifiers of the same tree branch.

4.3 Constraints & Domain Knowledge

Constraints and domain knowledge are objective measurements that reflect the interestingness of a discovered pattern. The aim of introducing constraints and domain knowledge is to tailor the result set to include only patterns that satisfy certain facts and are therefore interesting to the user. This also narrows the search space, thus optimising the discovery process.

Because the proposed architecture is split into two stages, separate sets of constraint can be applied to each of these phases thus increasing the flexibility and the ability to tailor the discovery process. Eliminating unwanted inter-field patterns also increases the performance of the second phase where inter-record patterns are built upon inter-field patterns, which have satisfied all constraints. Domain knowledge, incorporated in the form of taxonomies and templates can also be applied to both phases of the discovery process. This is of particular interest for taxonomies since it enables the generalization of items as well as of inter-field patterns. Figure 4 (a) shows an example taxonomy generalising items. Whereas Figure 4 (b) shows a taxonomy generalising inter-field grouping pattern combinations into a higher group of generality.

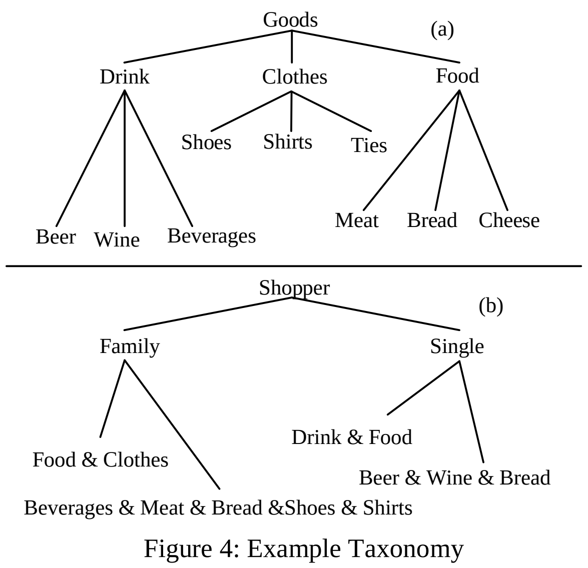

Figure 4: Example Taxonomy

5 Experimental Results

To evaluate the proposed algorithm several experiments have been performed on different artificial data sets.

Analysing how effective a tree structure can be deployed independent from other factors, two simple databases were created each containing a single set. The first includes one sub-set containing 20 items to analyze the deployment of T^I, whereas the second data set includes 20 sub-sets containing each a single item to analyze the deployment of T^R. Figure 5 shows the construction times for both trees and the number of nodes added at each iteration.

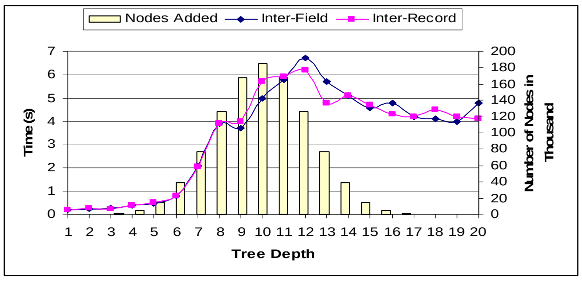

Figure 5: Tree Deployment Characteristics

Next the impact of an increasing sub-set size |*Q*| is analyzed leading to larger inter-field patterns and an increasing set size |S| leading to larger inter-record patterns.

A number of data sets have been created each containing 1000 sets, the average sets and sub-set size is 5 and the number of distinct items is 500. The minimum frequency is set to 0.3%. Figure 6 shows the overall runtime behaviour and the number of patterns discovered for associative (A) patterns as well as for sequential (S) patterns. As expected, the execution times behave in a similar way, ranging from 2 to 170 seconds and are connected to the number of patterns found.

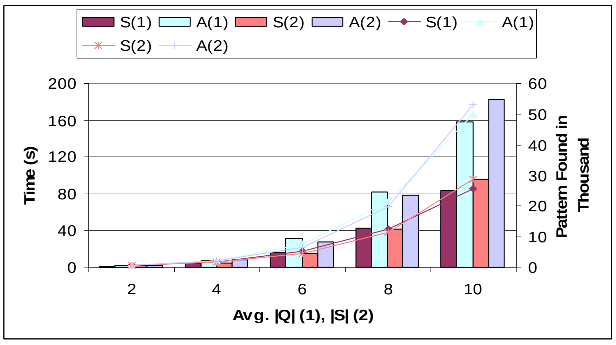

Figure 6: Increasing |Q| and |S|

Next, the impact of an increasing number of sets and a decreasing $f_{\min}$ is analyzed. In Figure 7, 5 data sets were evaluated where the number of sets was increased from 10,000 to 50,000. In order to exclude other factors all data sets contained only one set, which was repeated *n* times. Execution times where between 8 and 45 minutes increasing in a linear fashion.

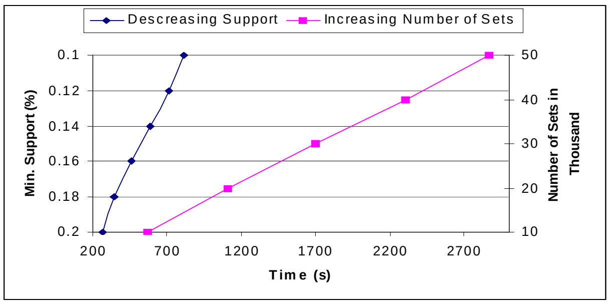

Figure 7: Scale up Characteristics

Also in Figure 7 the scale-up characteristics for a decreasing $f_{\min}$. The data set used contained 5,000 sets with an average set and sub-set size of 5 and 100 distinct items. The analysis shows that the runtime increases approximately linear with respect to $f_{\min,}$ resulting in execution times between 4 and 13 minutes.

6 Conclusions and Further Work

A tree-growth based method has been presented to effectively discover patterns across two dimensions. The proposed tree structure provides a flexible and efficient method to store and process patterns and it is easily extendable towards different patterns types and additional pattern dimensions. Incorporating constraints and domain knowledge on both dimensions enables the proposed algorithm to discover tailored patterns as desired by the user. The runtime evaluation undertaken has shown that the architecture is stable and efficient.

To reduce the number of database scans required two methods can be used. Firstly, the current tree is extended by more than one level for each run over the database. Secondly, the second tree storing inter-record patterns is extended simultaneously to the tree holding inter-field pattern.

Another interesting extension to the proposed algorithm is the discovery of patterns incorporating more than two dimensions. This can be achieved by introducing a new tree for each dimension. Initial work has been carried out to extend the proposed algorithm to dynamically add new pattern dimensions and preliminary results are encouraging.

References

[1] R. Agrawal, S. Srikant: Fast Algorithms for Mining Association Rules, Proc. Of the 20th VLDB Conference, Santiago, Chile, 1994

[2] R. Agrawal, R. Srikant. Mining Sequential Patterns, Proc. 11th Int'l Conf. On Data Eng., pp. 3-14, 1995.

[3] M. Baumgarten, A.G. Büchner, S.S. Anand, M.D. Mulvenna, J.G. Hughes. Navigation Pattern Discovery from Internet Data, B. Masand, M. Spiliopoulou (eds.) Advances in Web Usage Analysis and User Profiling, Springer-Verlag, 2000.

[4] J.Han, J.Pei; Mining Frequent Patterns by Pattern-Growth: Methodology and Implications; ACM SIGKDD Explorations, Dec. 2000.

[5] A Universal Formulation of Sequential Patterns; Proc. of the KDD'2001 workshop on Temporal Data Mining, San Francisco, August 2001.

[6] W. Klösgen, J. Żytkow (eds.). Handbook of Data Mining, Oxford University Press, 2002.

[7] R. Srikant, R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements, 5th Int'l Conf. On Extending Database Technology, pp. 3-17, 1996.

[8] G. Piatetsky-Shapiro: Knowledge Discovery in Databases, AAI/MIT Press, 1991