

# DECOMPOSABLE FAMILIES OF ITEMSETS

Nikolaj Tatti and Hannes Heikinheimo



TEKNILLINEN KORKEAKOULU  
TEKNISKA HÖGSKOLAN  
HELSINKI UNIVERSITY OF TECHNOLOGY  
TECHNISCHE UNIVERSITÄT HELSINKI  
UNIVERSITE DE TECHNOLOGIE D'HELSINKI



# DECOMPOSABLE FAMILIES OF ITEMSETS

Nikolaj Tatti and Hannes Heikinheimo

Helsinki University of Technology  
Faculty of Information and Natural Sciences  
Department of Information and Computer Science

Teknillinen korkeakoulu  
Informaatio- ja luonnontieteiden tiedekunta  
Tietojenkäsittelytieteen laitos

Distribution:

Helsinki University of Technology  
Faculty of Information and Natural Sciences  
Department of Information and Computer Science  
P.O.Box 5400  
FI-02015 TKK  
FINLAND  
URL: <http://ics.tkk.fi>  
Tel. +358 9 451 1  
Fax +358 9 451 3369  
E-mail: [series@ics.tkk.fi](mailto:series@ics.tkk.fi)

© Nikolaj Tatti and Hannes Heikinheimo

ISBN 978-951-22-9369-8 (Print)  
ISBN 978-951-22-9382-7 (Online)  
ISSN 1797-5034 (Print)  
ISSN 1797-5042 (Online)  
URL: <http://www.otalib.fi/tkk/edoc/>

TKK ICS  
Espoo 2008

**ABSTRACT:** The problem of selecting a small, yet high quality subset of patterns from a larger collection of itemsets has recently attracted a lot of research. Here we discuss an approach to this problem using the notion of decomposable families of itemsets. Such itemset families define a probabilistic model for the data from which the original collection of itemsets was derived. Furthermore, they induce a special tree structure, called a junction tree, familiar from the theory of Markov Random Fields. The method has several advantages. The junction trees provide an intuitive representation of the mining results. From the computational point of view, the model provides leverage for problems that could be intractable using the entire collection of itemsets. We provide an efficient algorithm to build decomposable itemset families, and give an application example with frequency bound querying using the model. An empirical study show that our algorithm yields high quality results.

**KEYWORDS:** Itemsets, Decomposable models, Boolean query

# CONTENTS

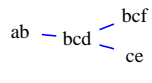
1	Introduction	1
2	Preliminaries and Notations	2
3	Decomposable Families of Itemsets	2
4	Finding Trees with Low Entropy	3
4.1	Definition of the Algorithm . . . . .	3
4.2	Model Selection . . . . .	6
4.3	Computing Multiple Decomposable Families . . . . .	6
5	Boolean Queries with Decomposable Families	7
6	Experiments	8
6.1	Datasets . . . . .	8
6.2	Generating Decomposable Families . . . . .	9
6.3	Reducing itemsets . . . . .	10
6.4	Boolean Queries . . . . .	11
7	Related Work	12
8	Conclusions and Future Work	12
A	Appendix	15

# 1 INTRODUCTION

Frequent itemset discovery has been a central research theme in the data mining community ever since the idea was introduced by Agrawal et. al [1]. Over the years, scalability of the problem has been the most studied aspect, and several very efficient algorithms for finding all frequent itemsets have been introduced, Apriori [2] or FP-growth [16] among others. However, it has been argued recently that while efficiency of the mining task is no longer a bottleneck, there is still a strong need for methods that derive compact, yet high quality results with good application properties [17].

In this study we propose the notion of decomposable families of itemsets to address this need. The general idea is to build a probabilistic model of a given dataset  $D$  using a small well-chosen subset of itemsets  $\mathcal{G}$  from a given candidate family  $\mathcal{F}$ . The candidate family  $\mathcal{F}$  may be generated from  $D$  using some frequent itemset mining algorithm. A special aspect of a decomposable family is that it induces a type of tree, called a junction tree, a well-known concept from the theory of Markov Random Fields [9].

As a simple example, consider a dataset  $D$  with six attributes  $a, \dots, f$ , and a family  $\mathcal{G} = \{bcd, bcf, ab, ce, bc, bd, cd, bf, cf, a, b, c, d, e, f\}$ . The family  $\mathcal{G}$  can be represented as the junction tree shown in Figure 1 such that the nodes in the tree are the maximal itemsets in  $\mathcal{G}$ . Furthermore, the junction tree defines a decomposable model of the dataset  $D$ .



$$p(abcdef) = \frac{p(ab)p(bcd)p(bcf)p(ce)}{p(b)p(bc)p(c)}$$

Figure 1: An example of a junction tree and the corresponding distribution decomposition.

Using decomposable itemset families has several notable advantages. First of all, the following junction tree graphs provide an extremely intuitive representation of the mining results. This is a significant advantage over many other itemset selection methods, as even small mining results of, say 50 itemsets, can be hard for humans to grasp as a whole, if just plainly enumerated. Second, from the computational point of view, decomposable families of itemsets provide leverage for accurately solving problems that could be intractable using the entire result set. Such problems include, for instance, querying for frequency bounds of arbitrary attribute combinations. Third, the statistical nature of the overall model enable to incorporated regularization terms, like BIC, AIC, or MDL to select only itemsets that reflect true dependencies between attributes.

In this study we provide an efficient algorithm to build decomposable itemset families while optimizing the likelihood of the model. We also demonstrate how to use decomposable itemset families to execute frequency bound querying, an intractable problem in the general case. We provide empirical results showing that our algorithm works well in practice.

The rest of the paper is organized as follows. Preliminaries are given in Section 2 and the concept of decomposable models are defined in Section 3. A greedy search algorithm is given in Section 4. Section 6 is devoted to experiments. We present the related work in Section 7 and conclude the paper with discussion in Section 8. The proofs for the theorems in this paper are provided in Appendix.

## 2 PRELIMINARIES AND NOTATIONS

In this section we describe the notation and the background definitions that are used in the subsequent sections.

A *binary dataset*  $D$  is a collection of  $N$  *transactions*, binary vectors of length  $K$ . The dataset can be viewed as a binary matrix of size  $N \times K$ . We denote the number of transactions by  $|D| = N$ . The  $i$ th element of a random transaction is represented by an *attribute*  $a_i$ , a Bernoulli random variable. We denote the collection of all the attributes by  $A = \{a_1, \dots, a_K\}$ . An *itemset*  $X = \{x_1, \dots, x_L\} \subseteq A$  is a subset of attributes. We will often use the dense notation  $X = x_1 \cdots x_L$ .

Given an itemset  $X$  and a binary vector  $v$  of length  $L$ , we use the notation  $p(X = v)$  to express the probability of  $p(x_1 = v_1, \dots, x_L = v_L)$ . If  $v$  contains only 1s, then we will use the notation  $p(X = 1)$ .

Given a binary dataset  $D$  we define  $q_D$  to be an *empirical distribution*,

$$q_D(A = v) = |\{t \in D; t = v\}|/|D|.$$

We define the frequency of an itemset to be  $fr(X) = q_D(X = 1)$ . The *entropy* of an itemset  $X = x_1 \cdots x_L$  given  $D$ , denoted by  $H(X; D)$ , is defined as

$$H(X; D) = - \sum_{v \in \{0,1\}^L} q_D(X = v) \log q_D(X = v), \quad (1)$$

where the usual convention  $0 \log 0 = 0$  is used. We often omit  $D$ .

We say that a family  $\mathcal{F}$  of itemsets is *downward closed* if each subset of a member of  $\mathcal{F}$  is also included in  $\mathcal{F}$ . An itemset  $X \in \mathcal{F}$  is *maximal* if there is no  $Y \in \mathcal{F}$  such that  $X \subset Y$ . We define  $m(\mathcal{F}) = \{|X|; X \in \mathcal{F}\}$  to be the maximal number of attributes in a single itemset.

## 3 DECOMPOSABLE FAMILIES OF ITEMSETS

In this section we define the concept of decomposable families. Itemsets of a decomposable family form a junction tree, a concept from the theory of Markov Random Fields [9].

Let  $\mathcal{G} = \{G_1, \dots, G_M\}$  be a downward closed family of itemsets covering the attributes  $A$ . Let  $H$  be a graph containing  $M$  nodes where the  $i$ th node corresponds to the itemset  $G_i$ . Nodes  $G_i$  and  $G_j$  are connected if  $G_i$  and  $G_j$  have a common attribute. The graph  $H$  is called the *clique graph* and the nodes of  $H$  are called *cliques*.

We are interested in spanning trees of  $H$  having a *running intersection property*. To define this property let  $\mathcal{T}$  be a spanning tree of  $H$ . Let  $G_i$  and  $G_j$  be two sets having a common attribute, say,  $a$ . These sets are connected in  $\mathcal{T}$  by a unique path. Assume that  $a$  occurs in every  $G_k$  along the path from  $G_i$  to  $G_j$ . If this holds for any  $G_i, G_j$ , and any common attribute  $a$ , then we say that the tree has a running intersection property. Such a tree is called a *junction tree*.

We should point out that the clique graph can have multiple junction trees and that not all spanning trees are junction trees. In fact, it may be the case that the clique graph does not have junction trees at all. If, however, the clique graph has a junction tree, we call the original family  $\mathcal{G}$  *decomposable*.

We label edge  $(G_i, G_j)$  of a given junction tree  $\mathcal{T}$  with a set of mutual attributes  $G_i \cap G_j$ . This label set is called a *separator*. We denote the set of all separators by  $S(\mathcal{T})$ . Furthermore, we denote the cliques of the tree by  $V(\mathcal{T})$ .



Given a junction tree  $\mathcal{T}$  and a binary vector  $v$ , we define the probability of  $A = v$  to be

$$p(A = v; \mathcal{T}) = \prod_{X \in V(\mathcal{T})} q_D(X = v_X) / \prod_{Y \in S(\mathcal{T})} q_D(Y = v_Y). \quad (2)$$

It is a known fact that the distribution given in Eq. 2 is actually the unique maximum entropy distribution [18, 10]. Note that  $p(A = v; \mathcal{T})$  can be computed from the frequencies of the itemsets in  $\mathcal{G}$  using the inclusion-exclusion principle.

It can be shown that the family  $\mathcal{G}$  is decomposable if and only if the maximal sets of  $\mathcal{G}$  is decomposable and that Eq. 2 for the maximal sets of  $\mathcal{G}$  and the whole  $\mathcal{G}$ . Hence, we usually construct the tree using only the maximal sets of  $\mathcal{G}$ . However, in some cases it is convenient to have non-maximal sets as the cliques. We will refer to such cliques as *redundant*. For a tree  $\mathcal{T}$  we define a family of itemsets,  $s(\mathcal{T})$  to be the downward closure of its cliques,  $V(\mathcal{T})$ . To summarize,  $\mathcal{G}$  is decomposable if and only if there is a junction tree  $\mathcal{T}$  such that  $\mathcal{G} = s(\mathcal{T})$ .

Calculating the entropy of the tree  $\mathcal{T}$  directly from Eq. 2 gives us

$$H(\mathcal{T}) = \sum_{X \in V(\mathcal{T})} H(X) - \sum_{Y \in S(\mathcal{T})} H(Y).$$

A direct calculation from Eqs. 1–2 reveals that  $\log p(D; \mathcal{T}) = -|D|H(\mathcal{T})$ . Hence, maximizing the log-likelihood of the data given  $\mathcal{T}$  (whose components are derived from the same data), is equivalent to minimizing the entropy.

We can define the maximum entropy distribution for any cover  $\mathcal{F}$  via linear constraints [10]. The downside of this general approach is that solving such a distribution is a **PP**-hard problem [26].

The following definition will prove itself useful in subsequent sections. Given two downward closed covers  $\mathcal{G}_1$  and  $\mathcal{G}_2$ . We say that  $\mathcal{G}_1$  *refines*  $\mathcal{G}_2$ , if  $\mathcal{G}_1 \subseteq \mathcal{G}_2$ .

**Proposition 1** *If  $\mathcal{G}_1$  refines  $\mathcal{G}_2$ , then  $H(\mathcal{G}_1) \geq H(\mathcal{G}_2)$ .*

## 4 FINDING TREES WITH LOW ENTROPY

In this section we describe the algorithm for searching decomposable families. To be more precise, given a candidate set, a downward closed family  $\mathcal{F}$  covering the set of attributes  $A$ , our goal is to find a decomposable downward closed family  $\mathcal{G} \subseteq \mathcal{F}$ . Hence our goal is to find a junction tree  $\mathcal{T}$  such that  $s(\mathcal{T}) \subseteq \mathcal{F}$ .

### 4.1 Definition of the Algorithm

We search the tree in an iterative fashion. At the beginning of each iteration round we have a junction tree  $\mathcal{T}_{n-1}$  whose cliques have at most  $n$  attributes, that is  $m(s(\mathcal{T})) = n$ . The first tree is  $\mathcal{T}_0$  containing only single attributes and no edges. During each round the tree is modified so that in the end we will have  $\mathcal{T}_n$ , a tree with cliques having at most  $n + 1$  attributes.

In order to fully describe the algorithm we need the following definition:  $X$  and  $Y$  are said to be  $n - 1$ -connected in a junction tree  $\mathcal{T}$ , if there is a path in  $\mathcal{T}$  from  $X$  to  $Y$  having at least one separator of size  $n - 1$ . We say that  $X$  and  $Y$  are 0-connected, if  $X$  and  $Y$  are not connected.

Each round of the algorithm consists of three steps. The pseudo-code of the algorithm is given in Algorithm 1–2.

1. **Generate:** We construct a graph  $G_n$  whose nodes are the cliques of size  $n$  in  $\mathcal{T}_{n-1}$ . We add all the edges to  $G_n$  having the form  $(X, Y)$  such that  $|X \cap Y| = n - 1$  and  $X \cup Y \in \mathcal{F}$ . We also set  $\mathcal{T}_n = \mathcal{T}_{n-1}$ . The weight of the edge is set to

$$w(e) = H(X) + H(Y) - H(X \cap Y) - H(X \cup Y).$$

2. **Augment:** We select the edge, say  $e = (X, Y)$ , having the largest weight and remove it from  $G_n$ . If  $X$  and  $Y$  are  $n - 1$ -connected in  $\mathcal{T}_n$  we add  $\mathcal{T}_n$  with a new clique  $V = X \cup Y$ . Furthermore, for each  $v \in V$ , we consider  $W = V - v$ . If  $W$  is not in  $\mathcal{T}_n$ , it is added into  $\mathcal{T}_n$ . Next,  $W$  and  $V$  are connected in  $\mathcal{T}_n$ . At the same time, the node  $W$  is also added into  $G_n$  and the edges of  $G_n$  are added using the same criteria as in Step 1 (Generate). Finally, a possible cycle is removed from  $\mathcal{T}_n$  by finding an edge with separator of size  $n - 1$ . Augmenting is repeated as long as  $G_n$  has no edges.
3. **Purge:** The tree  $V(\mathcal{T}_n)$  contains redundant cliques after augmentation. We remove these redundant cliques from  $\mathcal{T}_n$ .

To illustrate the algorithm we provide a toy example.

**Example 2** Consider that we have a family

$$\mathcal{F} = \{a, b, c, d, e, ab, ac, ad, bc, bd, cd, ce, abc, acd, bcd\}.$$

Assume that we are at the beginning of the second round and we already have the junction tree  $\mathcal{T}_1$  given in Figure 2(a). We form  $G_2$  by taking the edges  $(ab, bc)$  and  $(bc, cd)$ .

Consider that we pick  $ab$  and  $bc$  for joining. This will spawn  $ac$  and  $abc$  in  $\mathcal{T}_2$  (Figure 2(c)) and  $ac$  in  $G_2$  (Figure 2(d)). Note that we also add the edge  $(ac, cd)$  into  $G_2$ . Assume that we continue by picking  $(ac, cd)$ . This will spawn  $acd$  and  $cd$  into  $\mathcal{T}_2$ . Note that  $(bc, cd)$  is removed from  $\mathcal{T}_2$  in order to break the cycle.

The last edge  $(bc, cd)$  is not added into  $\mathcal{T}_2$  since  $bc$  and  $cd$  are not  $n - 1$ -connected. The final tree (Figure 2(f)) is obtained by keeping only the maximal sets, that is, purging the cliques  $bc, ab, ac, ad,$  and  $cd$ . The corresponding decomposable family is  $\mathcal{G} = \mathcal{F} - bcd$ .

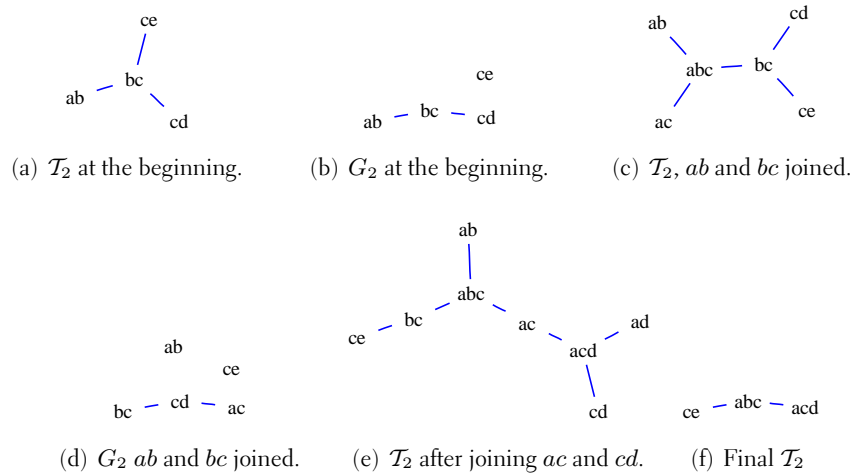


Figure 2: Example of graphs during different stages of SEARCHTREE algorithm.

The next theorem states that the **Augment** step does not violate the running intersection property.

---

**Algorithm 1** SEARCHTREE algorithm. The input is a downward closed cover  $\mathcal{F}$ , the output is a junction tree  $\mathcal{T}$  such that  $V(\mathcal{T}) \subseteq \mathcal{F}$ .

---

```

 $V(\mathcal{T}_0) \leftarrow \{x; x \in A\}$  { $\mathcal{T}_0$  contains the single items.}
 $n \leftarrow 0$ .
repeat
   $n \leftarrow n + 1$ .
   $\mathcal{T}_n \leftarrow \mathcal{T}_{n-1}$ .
   $V(G_n) \leftarrow \{X \in V(\mathcal{T}_n); |X| = n\}$ .
   $E(G_n) \leftarrow \{(X, Y); X, Y \in V(G_n), |X \cap Y| = n - 1, X \cup Y \in \mathcal{F}\}$ .
  repeat
     $e = (X, Y) \leftarrow \arg \max_{x \in E(G_n)} w(x)$ .
     $E(G_n) \leftarrow E(G_n) - e$ .
    if  $X$  and  $Y$  are  $n - 1$ -connected in  $\mathcal{T}_n$  then
      Call MODIFYTREE.
    end if
  until  $E(G_n) = \emptyset$ 
  Delete nodes marked by MODIFYTREE from  $\mathcal{T}_n$ , connect the incident nodes.
until  $G_n$  is empty
return  $\mathcal{T}$ 

```

---

**Algorithm 2** MODIFYTREE algorithm.

---

```

 $B \leftarrow X \cup Y$ .
 $V(\mathcal{T}_n) \leftarrow V(\mathcal{T}_n) \cup \{B\}$ . {Add new clique  $B$  into  $\mathcal{T}_n$ .}
for  $v \in B$  do
   $W \leftarrow B - v$ .
  Mark  $W$ .
  if  $W \notin V(G_n)$  then
     $V(G_n) \leftarrow V(G_n) \cup \{W\}$ .
     $E(G_n) \leftarrow E(G_n) \cup \{(W, Z); Z \in V(G_n), |X \cap Z| = n - 1, W \neq X \cup Z \in \mathcal{F}\}$ .
     $V(\mathcal{T}_n) \leftarrow V(\mathcal{T}_n) \cup \{W\}$ .
  end if
   $E(\mathcal{T}_n) \leftarrow E(\mathcal{T}_n) \cup (B, W)$ .
end for
Remove the possible cycle in  $\mathcal{T}_n$  by removing an edge  $(U, V)$  connecting  $X$  and  $Y$  and having  $|U \cap V| = n - 1$ .

```

---

**Theorem 3** Let  $\mathcal{T}$  be a junction tree with cliques of size  $n + 1$ , at maximum, that is,  $m(s(\mathcal{T})) = n + 1$ . Let  $X, Y \in V(\mathcal{T})$  be cliques of size  $n$  such that  $|X \cap Y| = n - 1$ . Set  $B = X \cup Y$ . Then the family  $s(\mathcal{T}) \cup \{C; C \subseteq B\}$  is decomposable if and only if  $X$  and  $Y$  are  $n - 1$ -connected in  $\mathcal{T}$ .

**Theorem 4** MODIFYTREE decreases the entropy of  $\mathcal{T}_n$  by  $w(e)$ .

Theorems 3–4 imply that SEARCHTREE algorithm is nothing more than a greedy search. However, since we are adding cliques in rounds we can state that under some conditions the algorithm returns an optimal cover for each round.

**Theorem 5** Assume that the members of  $\mathcal{F}$  of size  $n + 1$  are added to  $G_n$  at the beginning of the  $n$ th round. Let  $\mathcal{U}$  be a junction tree such that  $s(\mathcal{T}_n) \subseteq s(\mathcal{U})$  and  $m(s(\mathcal{U})) = n + 1$ . Then  $H(\mathcal{T}_{n+1}) \leq H(\mathcal{U})$ .

**Corollary 6** The tree  $\mathcal{T}_1$  is optimal among the families using the sets of size 2.

Corollary 6 states that  $\mathcal{G}_1$  is the Chow-Liu tree [8].

## 4.2 Model Selection

Theorem 1 reveals a drawback in the current approach. Consider that we have two independent items  $a$  and  $b$  and that  $\mathcal{F} = \{a, b, ab\}$ . Note that  $\mathcal{F}$  is itself decomposable and  $\mathcal{G} = \mathcal{F}$ . However, a more reasonable family would be  $\{a, b\}$  to reflect the fact that  $a$  and  $b$  are independent. To remedy this problem we will use model selection techniques such as BIC [24], AIC [3], and Refined MDL [14]. All these methods score the model by adding a penalty term to the likelihood.

We modify the algorithm by considering only the edges in  $G_n$  that improve the score. For BIC this reduces to considering only the edges satisfying

$$|D|w(e) \geq 2^{n-2} \log |D|,$$

where  $n$  is the current level of SEARCHTREE algorithm. Using AIC leads to the considering only the edges for which

$$|D|w(e) \geq 2^{n-1}.$$

Refined MDL is more troublesome. The penalty term in MDL is known as *stochastic complexity*. In general, there is no known closed formula for the stochastic complexity, but it can be computed for the multinomial distribution in linear time [19]. However, it is numerically unstable for data with large number of transactions. Hence, we will apply often-used asymptotic estimate [22] and define the penalty term

$$C_{\text{MDL}}(k) = \frac{k-1}{2} \log |D| - \frac{1}{2} \log \pi - \log \Gamma(k/2)$$

for  $k$ -multinomial distribution.

There are no known exact or approximative solution in a closed form of stochastic complexity for junction trees. Hence we propose the penalty term for the tree to be

$$\sum_{X \in \mathcal{V}(\mathcal{T})} C_{\text{MDL}}(2^{|X|}) - \sum_{Y \in \mathcal{S}(\mathcal{T})} C_{\text{MDL}}(2^{|Y|}).$$

Here we think that a single clique  $X$  is a  $2^{|X|}$ -multinomial distribution and we compensate the possible overlaps of the cliques by subtracting the complexity of the separators. Using this estimate leads to a selection criteria

$$|D|w(e) \geq C_{\text{MDL}}(2^{n+1}) - 2C_{\text{MDL}}(2^{n|}) + C_{\text{MDL}}(2^{n-1}).$$

## 4.3 Computing Multiple Decomposable Families

We can use SEARCHTREE algorithm for computing multiple decomposable covers from a single candidate set  $\mathcal{F}$ . The motivation behind this approach is that we get a sequence of covers, each cover holding partial information of the original cover  $\mathcal{F}$ . We will show empirically in Section 6.4 that the by exploiting the union information of these covers we are able to improve significantly bounds for boolean queries (see Section 5).

The idea is as follows. Set  $\mathcal{F}_1 = \mathcal{F}$  and let  $\mathcal{G}_1$  be the first decomposable family constructed from  $\mathcal{F}_1$  using SEARCHTREE algorithm. We define

$$\mathcal{F}_2 = \mathcal{F}_1 - \{X \in \mathcal{F}_1; \text{there is } Y \in \mathcal{G}_1, |Y| > 1, Y \subseteq X\}.$$

We compute  $\mathcal{G}_2$  from  $\mathcal{F}_2$  and continue in the iterative fashion until  $\mathcal{G}_k$  contains nothing but individual items.

## 5 BOOLEAN QUERIES WITH DECOMPOSABLE FAMILIES

One of our motivations for constructing decomposable families is that some computational problems that are hard for general families of itemsets reduce to tractable if the underlying family is decomposable. In this section we will show that the computational burden of a boolean query, a classic optimization problem [15, 6], reduces significantly, if we are using decomposable families of itemsets.

Assume that we are given a set of known itemsets  $\mathcal{G}$  and a query itemset  $Q \notin \mathcal{G}$ . We wish to find  $fr(Q; \mathcal{G})$ , the possible frequencies for  $Q$  given the frequencies of  $\mathcal{G}$ . It is easy to see that the frequencies form an interval, hence it is sufficient to find the maximal and the minimal frequencies. We can express the problem of finding the maximal frequency as a search for the distribution  $p$  solving

$$\begin{aligned} \max \quad & p(Q = 1) \\ \text{s.t.} \quad & p(X = 1) = fr(X), \text{ for each } X \in \mathcal{G}. \\ & p \text{ is a distribution over } A. \end{aligned} \quad (3)$$

We can solve Eq. 3 using Linear Programming [15]. However, the number of variables in the program is  $2^{|A|}$  and makes the program tractable only for small datasets. In fact, solving Eq. 3 is an NP-hard problem [26].

In the rest of the section we present a method of solving Eq. 3 with a linear program containing only  $2^{|Q|}|\mathcal{G}||A|$  variables, assuming that  $\mathcal{G}$  is decomposable. This method is an explicit construction of the technique presented in [27]. The idea behind the approach is that instead of solving a joint distribution in Eq. 3, we break the distribution into small component distributions, one for each clique in the junction tree. These components are forced to be consistent by requiring that they are equal at the separators. The details are given in Algorithm 3.

---

**Algorithm 3** QUERYTREE algorithm for solving a query  $Q$  from a decomposable cover  $\mathcal{G}$ . The output is the interval  $fr(Q; \mathcal{G})$ .

---

```

 $\{\mathcal{T}_1, \dots, \mathcal{T}_M\} \leftarrow$  connected components of a junction tree of  $\mathcal{G}$ .
for  $i = 1, \dots, M$  do
   $Q_i \leftarrow Q \cap (\bigcup V(\mathcal{T}_i))$ . {Items of  $Q$  contained in  $\mathcal{T}_i$ .}
   $\mathcal{U} \leftarrow \arg \min_{\mathcal{S} \subseteq \mathcal{T}_i} \{ |V(\mathcal{S})|; Q_i \subseteq \bigcup V(\mathcal{S}) \}$ . {Smallest subtree containing  $Q_i$ .}
  while there are changes do
    Remove the items outside  $Q_i$  that occur in only one clique of  $\mathcal{U}$ .
    Remove redundant cliques.
  end while
  Select one clique, say  $R$  from  $\mathcal{U}$  to be the root.
   $R \leftarrow R \cup Q_i$ . {Augment the root with  $Q_i$ }
  Augment the rest cliques in  $\mathcal{U}$  so that the running intersection property holds.
  Let  $p_C$  be a distribution over each clique  $C \in V(\mathcal{U})$ .
   $\alpha_i \leftarrow$  the solution of a linear program

      
$$\begin{aligned} \min \quad & p_R(Q_i = 1) \\ \text{s.t.} \quad & p_C(X = 1) = fr(X), \text{ for each } C \in V(\mathcal{U}), X \in \mathcal{G}, X \subseteq C. \\ & p_{C_1}(C_1 \cap C_2) = p_{C_2}(C_1 \cap C_2), \text{ for each } (C_1, C_2) \in E(\mathcal{U}). \end{aligned}$$


   $\beta_i \leftarrow$  the solution of the maximum version of the linear program.
end for
 $fr(Q; \mathcal{G}) \leftarrow \left[ \max \left( \sum_i^M \alpha_i - (M - 1), 0 \right), \min_i (\beta_i) \right]$ .
```

---

To clarify the process we provide the following simple example.

**Example 7** Assume that we have  $\mathcal{G}$  whose junction tree is given in Figure 3(a). Let query be  $Q = adg$ . We begin first by finding the smallest sub-tree containing  $Q$ . This results in purging  $fh$  (Figure 3(b)). We further purge the tree by removing  $e$  since it only occurs in one clique (Figure 3(c)). In the next step we pick a root, which in this case is  $bc$  and augment the cliques with the members of  $Q$  so that the root contains  $Q$  (Figure 3(d)). We finally remove the redundant cliques which are  $ab$ ,  $cd$ ,  $fg$ . The final tree is given in 3(e). Finally, the linear program is formed using two distributions  $p_{abcdg}$  and  $p_{cfg}$ . The number of variables in this program is  $2^5 + 2^3 = 40$  opposed to the original  $2^8 = 256$ .

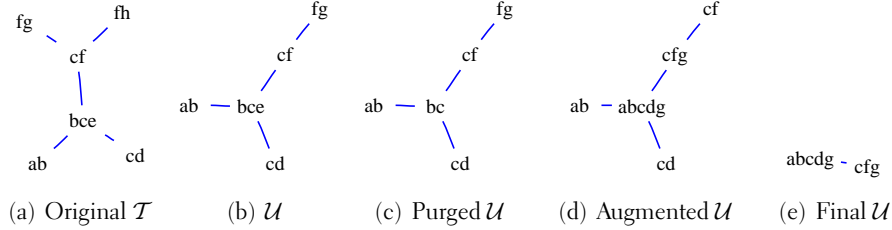


Figure 3: Junction trees during different stages of solving the query problem.

Note that we did not specify in Algorithm 3 which clique we selected to be the root  $R$ . The linear program depends on the root  $R$  and hence we select the root minimizing the number of variables in the linear program.

**Theorem 8** `QUERYTREE` algorithm solves correctly the boolean query  $fr(q; \mathcal{G})$ . The number of variables occurring in the linear programs is  $2^{|Q|} |\mathcal{G}| |A|$ , at maximum.

## 6 EXPERIMENTS

In this section we will study empirically the relationship between the decomposable itemset families and the candidate set, the role of the regularization, and the performance of boolean queries using multiple decomposable families.

### 6.1 Datasets

For our experiments we used one synthetic generated dataset, *Path*, and three real-world datasets: *Paleo*, *Courses* and *Mammals*. The synthetic dataset, *Path*, contained 8 items and 100 transactions. Each item was generated from the previous item by flipping it with a 0.3 probability. The first item was generated by a fair coin flip. The dataset *Paleo*<sup>1</sup> contains information of mammal fossils found in specific paleontological sites in Europe [13]. *Courses* describes computer science courses taken by students at the Department of Computer Science of the University of Helsinki. The *Mammals*<sup>2</sup> dataset consists of presence/absence records of current day European mammals [20]. The basic characteristics of the real-world data sets are shown in Table 1.

<sup>1</sup>NOW public release 030717 available from [13].

<sup>2</sup>The full version of the mammal dataset is available for research purposes upon request from the Societas Europaea Mammalogica ([www.european-mammals.org](http://www.european-mammals.org))

Dataset	# of rows	# of items	# of 1s	$\frac{\# \text{ of 1s}}{\# \text{ of rows}}$
<i>Paleo</i>	501	139	1980	16.0
<i>Courses</i>	3506	98	16086	4.6
<i>Mammals</i>	2183	124	54155	24.8

Table 1: The basic properties of the datasets.

## 6.2 Generating Decomposable Families

In our first experiment we examined the junction trees that were constructed for the *Path* dataset. We calculated a sequence of trees using the technique described in Section 4.3. As input to the algorithm we used an unconstrained candidate collection of itemsets (minimum support = 0) from *Path* and BIC as the regularization method. In Figure 4(a) we see that the first tree corresponds to the model used to generate the dataset. The second tree, given in Figure 4(b), tend to link the items that are one gap away from each other. This is a natural result since close items are the most informative about each other.

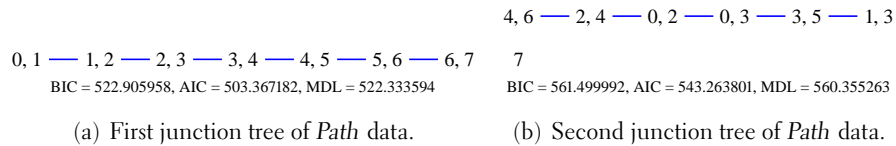


Figure 4: Junction trees for *Path*, a synthetic data in which an item is generated from the previous item by flipping it with 0.3 probability. The junction trees are regularized using BIC. The tree in Figure 4(b) is generated by ignoring the cliques of the tree in Figure 4(a).

With *Courses* data one large junction tree of itemsets is produced with several noticeable components. One distinct component at one end of the tree contains introductory courses like *Introduction to Programming*, *Introduction to Databases*, *Introduction to Application design* and *Java Programming*. Respectively, the other end of the tree features several distinct components with itemsets on more specialized themes in computer science and software engineering. The central node connecting each of these components in the entire tree is the itemset node  $\{\textit{Software Engineering, Models of Programming and Computing, Concurrent systems}\}$ .

Figure 5 shows about two-thirds of the entire *Courses* junction tree, with the component related to introductory courses removed because of the space constraints. We see a concurrent and distributed systems related component in the lower left part of the figure, a more software development oriented component in the lower right quarter and a Robotics/AI component in the upper right corner of the tree. The entire *Courses* junction tree can be found in Appendix.

We continued our experiments by studying the behavior of the model scores in a sequence of trees induced by a corresponding sequence of decomposable families. For the *Path* data the scores of the two first junction trees are shown in Figure 4, with the first one yielding smaller values. For the real-world datasets, we computed a sequence of trees from each dataset, again, with the unconstrained candidate collection as input and using AIC, BIC, or MDL respectively as the regularization method. Computation took about 1 minute per tree. The corresponding scores are plotted as a function of the order of the corresponding junction tree (Figure 6). The scores are increasing in the sequence, which is expected since the algorithm tries

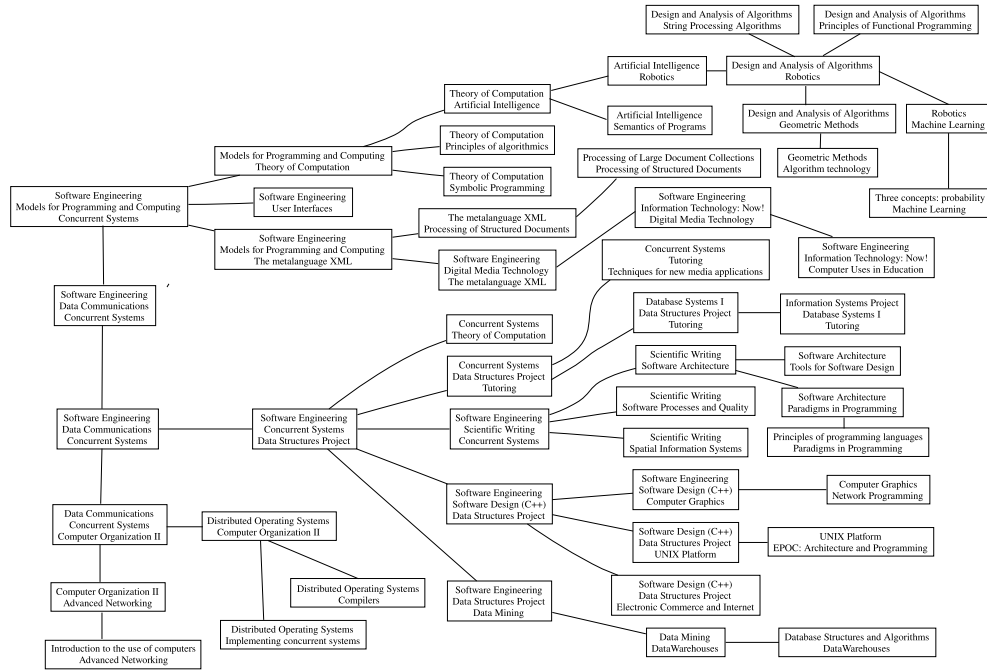


Figure 5: A part of the junction tree constructed from the *Courses* dataset. The tree was constructed using an unconstrained candidate family (min. support = 0) as input and BIC as regularization.

to select the best model and the subsequent trees are constructed from the left-over itemsets. The increase rate slows down towards the end since the last trees tend to have only singleton itemsets as nodes.

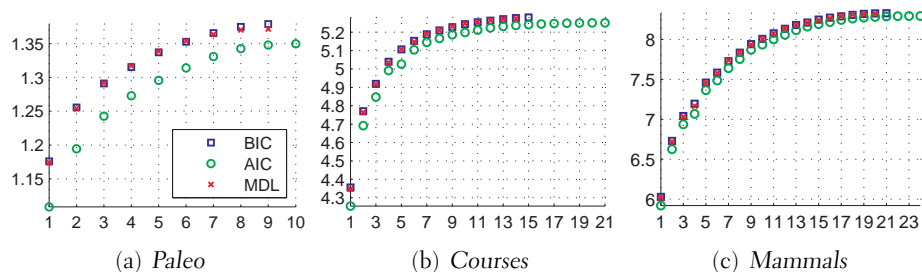


Figure 6: Scores of covers as a function of the order of the cover. Each cover is computed with an unconstrained candidate family (min. support = 0) as input and the corresponding regularization. The  $y$ -axis is the model score divided by  $10^4$ .

### 6.3 Reducing itemsets

Our next goal was to study the sizes of the generated decomposable families compared to the size of the original candidate set. As input for this experiment, we used several different candidate collections of frequent itemsets resulting from varying the support threshold, and generated the corresponding decomposable itemset families (Table 6.3).

From the results we see that the decomposable families are much smaller compared to the original candidate set, as a large portion of itemsets are pruned due to the running intersection property. The regularizations AIC, BIC, MDL prune the results further. The pruning is most effective when the candidate set is large.



Dataset	$\sigma$	$ \mathcal{F} $	First Family, $ \mathcal{G}_1 $				All Families, $ \bigcup \mathcal{G}_i $			
			AIC	BIC	MDL	None	AIC	BIC	MDL	None
<i>Mammals</i>	.20	2169705	221	213	215	10663	668	625	630	11103
<i>Mammals</i>	.25	416939	201	197	197	6820	535	507	509	7106
<i>Paleo</i>	.01	22283	339	281	290	5260	993	834	812	6667
<i>Paleo</i>	.02	979	254	235	239	376	463	433	429	733
<i>Paleo</i>	.03	298	191	190	190	210	231	228	228	277
<i>Paleo</i>	.05	157	147	147	147	151	149	149	149	156
<i>Courses</i>	.01	16945	217	202	206	4087	565	522	524	4357
<i>Courses</i>	.02	2493	185	177	177	625	354	342	342	751
<i>Courses</i>	.03	773	176	170	170	276	264	261	261	359
<i>Courses</i>	.05	230	136	132	132	158	167	164	164	186

Table 2: Sizes of decomposable families for various datasets. The second column is the minimum support threshold, the third column is the number of the frequent itemsets in the candidate set. The columns 4–7 contain the size of the first result family and the columns 8–11 contain the size of the union of the result families.

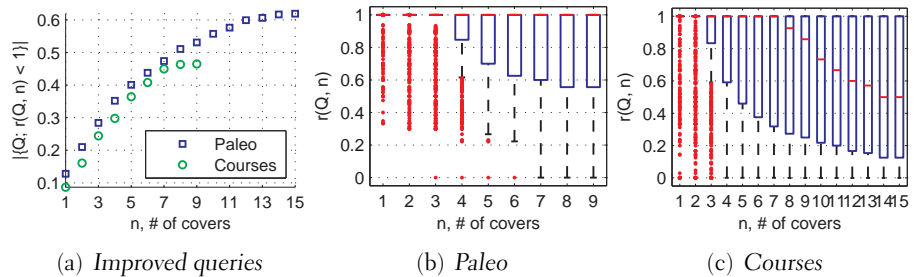


Figure 7: Boolean query ratios from *Paleo* and *Course* datasets. Figure 7(a) contains the percentage of queries having  $r(Q; n) < 1$ , that is, the percentage of queries improved over the singleton model as a function of the number of decomposable families. Figures 7(b)–7(c) are box plots of the ratios  $r(Q; n)$ , where  $Q$  is a random query and  $n$  is the number of decomposable families.

## 6.4 Boolean Queries

We conducted a series of boolean queries for *Paleo* and *Courses* datasets. For each dataset we pick randomly 1000 queries of size 5. We constructed a sequence of trees using BIC and the unconstrained (min. support = 0) candidate set as input. The average computation time for a single query was 0.3s. A portion (abt. 10%) of queries had to be discarded due to the numerical instability of the linear program solver we used.

A query  $Q$  for a decomposable family  $\mathcal{G}_i$  produces a frequency interval  $fr(Q; \mathcal{G}_i)$ . We also computed the frequency interval  $fr(Q; \mathcal{I})$ , where  $\mathcal{I}$  is a family containing nothing but singletons. We studied the ratios  $r(Q; n) = |\bigcap_1^n fr(Q; \mathcal{G}_i)| / |fr(Q; \mathcal{I})|$  as a function of  $n$ , that is, the ratio between the tightness of the bound using  $n$  families and the singleton model.

From the results given in Figure 7 we see that the first decomposable family in the sequence yields in about 10 % of the queries an improved bound with respect to the singleton family. As the number of decomposable families increases, the number of queries with tighter bounds goes from 10% up to 60%. Also, in general the absolute

bounds become tighter for the queries as we increase the number of decomposable families. For *Courses* the median of the ratio  $r(Q; 15)$  is about 0.5.

## 7 RELATED WORK

One of the main uses of our algorithm is in reducing itemset mining results into a smaller and a more manageable group of itemsets. One of the earliest approaches on itemset reduction include close itemsets [21] and maximal frequent itemset [23]. Also more recently, a significant amount of interesting research has been produced on the topic [7, 28, 25, 5]. Yan et al. [28] proposed a statistical model in which  $k$  representative patterns are used to summarize the original itemset family as well as possible. This approach has, however, a different goal to that of ours, as our model aims to describe the data itself. From this point of view the work by Siebes et al. [25] is perhaps the most in concordance to ours. Siebes et al. propose an MDL based method where the reduced group of itemsets aim to compress the data as well as possible. Yet, their approach is technically and methodologically quite different and does not provide a probabilistic model of the data as our model does. Furthermore, non of the above approaches provide a naturally following tree based representation of the mining results as our model does.

Traditionally, junction trees are not used as a direct model but rather as a technique for decomposing directed acyclic graph (DAG) models [9]. However, there is a clear difference between the DAG models and our approach. Assume that we have 4 items  $a, b, c$ , and  $d$ . Consider a DAG model  $p(a)p(b; a)p(c; a)p(d; bc)$ . While we can decompose this model using junction trees we cannot express it exactly. The reason for this is that the DAG model contains the assumption of independence of  $b$  and  $c$  given  $a$ . This allows us to break the clique  $abc$  into smaller parts. In our approach the cliques are the empirical distributions with no independence assumptions. DAG models and junction tree models are equivalent for Chow-Liu tree models [8].

Our algorithm for constructing junction trees is closely related to EFS algorithm [11, 4] in which new cliques are created in a similar fashion. The main difference between the approaches is that we add new cliques in a level-wise fashion. This allows a more straightforward algorithm. Another benefit of our approach is Theorem 5. On the other hand, Corollary 6 implies that our algorithm can be seen also as an extension of Chow-Liu tree model [8].

## 8 CONCLUSIONS AND FUTURE WORK

In this study we applied the concept of junction trees to create decomposable families of itemsets. The approach suits well for the problem of itemset selection, and has several advantages. The naturally following junction trees provide an intuitive representation of the mining results. From the computational point of view, the model provides leverage for problems that could be intractable using generic families of itemsets. We provided an efficient algorithm to build decomposable itemset families, and gave an application example with frequency bound querying using the model. Empirical results showed that our algorithm yields high quality results. Because of the expressiveness and good interpretability of the model, applications such as classification using decomposable families of itemsets could prove an interesting avenue for future research. Even more generally, we anticipate that in the future decomposable models could prove computationally useful with pattern mining ap-

plications that otherwise could be hard to tackle.

## References

- [1] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.
- [2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, pages 307–328, 1996.
- [3] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.
- [4] Stephan M. Altmueller and Robert M. Haralick. Practical aspects of efficient forward selection in decomposable graphical models. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 710–715, Washington, DC, USA, 2004. IEEE Computer Society.
- [5] B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *IEEE International Conference on Data Mining*, 2007.
- [6] Artur Bykowski, Jouni K. Seppänen, and Jaakko Hollmén. Model-independent bounding of the supports of Boolean formulae in binary data. In Pier Luca Lanzi and Rosa Meo, editors, *Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries*, LNCS 2682, pages 234–249. Springer Verlag, 2004.
- [7] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2002.
- [8] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968.
- [9] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and Davig J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag, 1999.
- [10] Imre Csiszár. I-divergence geometry of probability distributions and minimization problems. *The Annals of Probability*, 3(1):146–158, Feb. 1975.
- [11] Amol Deshpande, Minos N. Garofalakis, and Michael I. Jordan. Efficient step-wise selection in decomposable models. In *Conference in Uncertainty in Artificial Intelligence*, pages 128–135, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [12] Adrian Dobra and Stephen E. Fienberg. Bounds for cell entries in contingency tables given marginal totals and decomposable graphs. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 97(22):11885–11892, Oct. 2000.
- [13] Mikael Fortelius. Neogene of the old world database of fossil mammals (NOW). University of Helsinki, <http://www.helsinki.fi/science/now/>, 2005.

- [14] Peter D. Grünwald. *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.
- [15] Theodore Hailperin. Best possible inequalities for the probability of a logical function of events. *The American Mathematical Monthly*, 72(4):343–359, Apr. 1965.
- [16] J. Han and J. Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explorations Newsletter*, 2(2):14–20, 2000.
- [17] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1), 2007.
- [18] Radim Jiroušek and Stanislav Přeušil. On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics and Data Analysis*, 19:177–189, 1995.
- [19] Petri Kontkanen and Petri Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007.
- [20] A J Mitchell-Jones, G Amori, W Bogdanowicz, B Krystufek, P J H Reijnders, F Spitzenberger, M Stubbe, J B M Thissen, V Vohralik, and J Zima. *The Atlas of European Mammals*. Academic Press, 1999.
- [21] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.
- [22] Jorma Rissanen. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, 42(1):40–47, 1996.
- [23] Jr. Roberto J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD international conference on Management of data*, pages 85–93, New York, NY, USA, 1998. ACM.
- [24] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.
- [25] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *SIAM Conference on Data Mining*, pages 393–404, 2006.
- [26] Nikolaj Tatti. Computational complexity of queries based on itemsets. *Information Processing Letters*, pages 183–187, June 2006.
- [27] Nikolaj Tatti. Safe projections of binary data sets. *Acta Informatica*, 42(8–9):617–638, April 2006.
- [28] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: A profile-based approach. In *ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 2005.

## A APPENDIX

### Proof of Theorem 3

The theorem is trivial for the case  $n = 1$ . Hence we assume that  $n > 1$ .

Assume that  $X$  and  $Y$  are  $n-1$ -connected in  $\mathcal{T}$ , and let  $P$  be the path connecting  $X$  and  $Y$ . If  $(X, Y) \notin P$ , then there exists  $f = (Z, W) \in P$  such that  $|Z \cap W| = n-1$ . Since  $\mathcal{T}$  is a junction tree, we must have  $Z \cap W = X \cap Y$ . Hence by removing  $f$  and adding  $(X, Y)$  does not violate the running intersection property. Thus we can assume that  $(X, Y) \in E(\mathcal{T})$ . Adding  $V$  between  $X$  and  $Y$  does not violate the running intersection property and hence  $V(\mathcal{T}) + V$  remains decomposable.

To prove the other direction assume that  $X$  and  $Y$  are not  $n-1$  connected. This implies that the path  $P$  from  $X$  to  $Y$  consists of  $n+1$  cliques with  $n$  separators. Assume that  $V(\mathcal{T}) + V$  is decomposable and hence there is a junction tree  $\mathcal{U}$  having  $V(\mathcal{U}) = V(\mathcal{T}) + V$ . We can modify  $\mathcal{U}$  such that the edges of the path  $P$  occur in  $\mathcal{U}$ . Let  $P_s$  be the first clique in  $P$  and let  $P_l$  be the last. Note that  $V \subset P_s \cup P_l$ . Let  $P_e$  be the first clique in  $P$  along the path from  $V$  to  $P_s$ . Since  $|P_s \cap V| = n$ , we must have  $P_e \cap V = P_s \cap V$ . The path from  $V$  to  $P_l$  must also go through  $P_e$ , hence we must have  $P_s \cap V = P_l \cap V$ . This implies that either  $V = P_s$  or  $V = P_l$ , which is a contradiction. This completes the proof.

### Proof of Theorem 4

Assume that the edge  $e = (X, Y) \in E(\mathcal{T}_n)$ . Let  $\mathcal{T}'_n$  be the tree after adding  $X \cup Y$ . The entropy of the original tree is

$$H(\mathcal{T}_n) = H(X) + H(Y) - H(X \cap Y) + B,$$

where  $B$  is the impact of the rest nodes. The entropy of the new tree is

$$H(\mathcal{T}'_n) = H(X \cup Y) + B.$$

Hence we have  $H(\mathcal{T}_n) - H(\mathcal{T}'_n) = w(e)$ .

### Proof of Theorem 5

It is easy to see that the cliques  $X$  and  $Y$  are  $n-1$ -connected if and only if they are not connected by the previous edges from  $G_n$ . Hence, the algorithm reduces to Kruskal's algorithm in finding the optimal spanning tree of  $G_n$ , thus returning the optimal spanning tree.

Let  $\mathcal{U}$  be a junction tree refined by  $\mathcal{T}_n$  and containing the cliques of size  $n+1$ , at maximum. The cliques of size  $n+1$  occur in  $G_n$ . Let  $H$  be the corresponding edges in  $G_n$ . To prove the theorem we need to show that  $H$  contains no cycles.

Assume othewise, and consider adding the edges in  $H$ , one at the time. When the first cycle occurs, the corresponding family is not decomposable by Theorem 3. The argument in the proof of Theorem 3 holds even if we keep adding cliques of size  $n+1$ , hence the final family cannot be decomposable. Thus  $H$  cannot contain cycles.

### Proof of Theorem 8

Theorem 6 in [12] guarantees that breaking  $\mathcal{G}$  into connected components and computing  $fr(Q; \mathcal{G})$  from  $\alpha_i$  and  $\beta_i$  produce an accurate result as long as  $\alpha_i$  and  $\beta_i$  are

accurate. Theorem 7 in [27] states that taking the smallest subtree containing  $Q_i$  and removing attributes occurring in only one clique does not change  $\alpha_i$  and  $\beta_i$ .

Finally, we need to prove that the linear program of the algorithm produce the same  $\alpha_i, \beta_i$  as the linear program in Eq. 3. Let  $p$  be a distribution satisfying the conditions in Eq. 3. Clearly, we can break  $p$  into components satisfying the conditions of the linear program given in the algorithm. On other hand, assume that  $\{p_C\}$  now satisfy the conditions of the linear program given in the algorithm. Since components are equal at the separators we can combine this into one joint distribution  $p$  satisfying the condition of Eq. 3. This implies that the outcome of both programs are equivalent.

To prove the bound for the number of variables, note that for any clique  $C$  we have  $2^{|C|} \leq |\mathcal{G}|$ . We can have  $|A|$  cliques at most. Augmenting can increase the size of the cliques by  $|Q|$ , at maximum. This implies that the number of variables is  $\sum_i 2^{|Q|+|C_i|} = 2^{|Q|} \sum_i 2^{|C_i|} \leq 2^{|Q|} |\mathcal{G}| |A|$ .









TKK REPORTS IN INFORMATION AND COMPUTER SCIENCE

ISBN 978-951-22-9369-8 (Print)

ISBN 978-951-22-9382-7 (Online)

ISSN 1797-5034 (Print)

ISSN 1797-5042 (Online)