An efficient Bayesian network approach for discovering interesting patterns

R. Malhas & Z. Al Aghbari
Department of Computer Science, University of Sharjah, UAE

Abstract

The main problem faced by all association rule/pattern mining algorithms is their production of a large number of rules which incurred a secondary mining problem; namely, mining *interesting* association rules/patterns. The problem is compounded by the fact that 'common knowledge' discovered rules are not interesting, but they are usually strong rules with high support and confidence levels – the classical measures.

In this paper, we present an efficient algorithm for discovering interesting (unexpected) patterns based on background knowledge, represented by a Bayesian network. A pattern/rule is unexpected if it is 'surprising' to the user. The algorithm profiles a pattern as interesting (unexpected), if the absolute difference between its support estimated from the dataset and the Bayesian network exceeds a user specified threshold (ε). Itemsets with the highest diverging supports are considered the most interesting. The efficiency of the Java implementation of the algorithm is verified experimentally.
Keywords: interesting patterns, association rules, frequent itemsets, Bayesian network, background knowledge.

1 Introduction

Since the inception of the classical Apriori algorithm [1] for mining association rules, development of interestingness measures has been a vigilant area of research to mine *interesting* patterns out of a sheer volume of obvious and irrelevant rules. The problem is compounded since obvious 'common knowledge' discovered rules are not interesting, but they are usually strong rules with high support and confidence levels - the classical measures in [1].

In this paper, we present an efficient algorithm that discovers interesting/unexpected patterns based on background knowledge, represented by

WIT Transactions on Information and Communication Technologies, Vol 37, © 2006 WIT Press
www.witpress.com, ISSN 1743-3517 (on-line)
doi:10.2495/DATA060111

a Bayesian network. The algorithm adopts a definition of interestingness proposed and implemented in [2], where a pattern (itemset) is considered interesting if *the absolute difference between its support estimated from the dataset and the Bayesian network* exceeds a user specified threshold (ε). Patterns with the highest diverging supports are considered the most interesting. For short, it is called the *diverging supports* definition. Our algorithm differs from [2] in the way it computes the support (joint probability) of a pattern from the Bayesian network, which is known to be computationally hard. The efficiency of the Java implementation of the algorithm is verified experimentally.

2 Related work and motivation

As the focus of this paper is on discovering interesting patterns based on background knowledge, only related work within this context is discussed. Three main approaches that use background knowledge in their discovery schemes were identified: 1) *syntactic* based as in [3–6]; 2) *logic* based as in [7, 8]; and 3) *probabilistic* based as in [2, 9–11].

Syntactic based approaches necessitate defining some kind of language for knowledge representation governed by a set of syntax rules, so that pair wise rule comparisons are conducted; one from the set of knowledge rules and the other from the data rules. If a syntax difference is captured- i.e. a similar rule body but a dissimilar rule head or vice versa- a data rule is considered interesting.

Logic based approaches are similar to the syntactic based ones because both adopt pair wise rule comparisons; but they look for logical contradictions instead of syntax differences between prior knowledge rules and data rules.

Probabilistic based approaches use belief systems for background knowledge representation, to be able to introduce uncertainty through assigning some *degree* or *confidence factor* to each belief. Although the authors of [9, 10] laid the ground for using a Bayesian or a Dempster-Shafer approach (reasoning under *uncertainty)*, their later algorithms in [7, 8] leveraged logical reasoning based on the user's *precise (certain)* knowledge! Two recent papers, [2] and [11], were the first to pick what was seeded in [7, 8] by adopting a discovery scheme that used a Bayesian network to represent background knowledge. But they adopted a different measure of interestingness.

In [9, 10], the proposed definition of *interestingness* should measure *how much a pattern affects the degrees of the beliefs in a belief system*; i.e. the more a pattern disagrees with the belief system the more unexpected and hence the more interesting it is. No formal algorithm was proposed in either paper.

Alternatively, the approaches in [2, 11] adopt the *diverging supports* measure that was mentioned in the introduction (and formally presented in section 3). The major concern of both papers is the computation of the support (joint probability) of patterns from the Bayesian network, which is known to be a computationally hard problem. In [2], the approach uses the bucket elimination algorithm [12] only to compute the joint probability distributions of supersets of frequent itemsets discovered. It then marginalizes this distribution to get the joint probability distribution of smaller ones (i.e. joint probability distributions for the

subsets are derived from the joint probability distributions of their supersets). Thus, achieving a reduction in the number of times the, relatively costly, bucket elimination algorithm is called when the number of frequent itemsets is very large. However, for relatively smaller numbers of frequent itemsets, the feasibility of the algorithm becomes questionable, because the cost incurred due to computing the joint probability distributions for supersets might be higher than directly computing the joint probability for each frequent itemset.

In [11], the approach leverages a sequential sampling algorithm which approximates the computation of the joint probabilities from the Bayesian network to avoid exact inference adopted in [2]. This approach is a good alternative to [2] for very large databases or nets, because it works with a sample drawn from the database to get away from processing the entire database to discover the most interesting patterns. Nevertheless, as with any sampling approach, the risk of missing the most interesting pattern will always be there.

The work in [2, 11] has motivated the work in this paper in a reversed manner. We have implemented the *diverging supports* definition of interestingness, using Java, by directly computing the joint probability for each pattern from the Bayesian network to cater for the cases where a direct method might be more feasible. As for future work, we plan to develop an *inference optimizer* algorithm that profiles the database and the Bayesian network on hand – with respect to the number of attributes, domain size, user specified parameters etc – to choose the most feasible inference approach out of the three.

3 Definitions and notation

Using database notation:

- Let $A_1, A_2, A_3, ..., A_i$ be the attributes of a dataset. The domain of an attribute A_k is donated by $Dom(A_k)$. Only categorical and discrete attributes with finite domains are considered.
- Let $I, J, k, \ldots$ ... (uppercase letters) be the *attribute sets*, where an attribute set $I=A_1, A_2, A_3,..., A_k$ or $I=\{A_1, A_2, A_3,..., A_k\}$. The domain of an attribute set I is: $Dom(I)=Dom(A_1) \times Dom(A_2) \times ... \times Dom(A_k)$.
- Let i,j,... (lowercase letters) be the values from the domains of attributes and attribute sets, where $i \in Dom(I)$ is a value from the domain of the attribute set I.
- P_I denote the joint probability of the attribute set I, where $P_I(i)=Pr(I=i)$ is the probability that $I=i$. Noting that $\sum_{i \in Dom(I)} P_I = 1$.
- Let the pair (I,i) be an *itemset*, where I is an attribute set and $i \in Dom(I)$.
- The *support of an itemset* (I,i) in a dataset is defined as

$$supp_{Data}(I,i) = P_I(i). \quad (1)$$

  (I,i) is *frequent* if its support ≥ *minSupport*; a user specified minimum support.
- Let *BN* be a *Bayesian network* over a set of attributes $H = A_1, A_2, ...,A_n$. The *BN* is a directed acyclic graph, $BN = (V,E)$, with the set of vertices $V = V_{A1},V_{A2},...,V_{An}$, and a set of edges $E \subset V \times V$. Each vertex V_A has a conditional probability distribution $P_{A_i|par_i}$, where $par_i = \{A_j : (V_{Aj}, V_{Ai}) \in E\}$ is the set of attributes corresponding to the parents of V_{Ai}.

- A Bayesian network *BN* over *H* encodes a *joint probability distribution* (JPD) of *H* represented by

$$P_H^{BN} = \prod_{i=1}^{n} P_{A_i|par_i} \ . \tag{2}$$

- The *support of an itemset (I,i)* in a Bayesian network *BN* is defined as

$$supp_{BN}(I,i) = P_I^{BN}(i) \ . \tag{3}$$

where the probability is estimated from the JPD encoded in the *BN*. An itemset is *frequent* if its support ≥ *minSupport*; a user specified minimum support.

3.1 Diverging supports definition of interestingness

This section presents the *diverging supports* definition formally.

3.1.1 Interestingness of an itemset relative to a Bayesian network

Let *BN* be a Bayesian network over an attribute set *H*, and let the pair *(I,i)* be an itemset, such that $I \subseteq H$ and $i \in Dom(I)$. An itemset is considered interesting if the absolute difference between its support estimated from the dataset and the *BN* exceeds a user specified threshold ε. Thus, the *interestingness of an itemset (I,i) relative to* the Bayesian network *BN* is given by [2]

$$\mathcal{I}_{BN}(I,i) = |supp_{Data}(I,i) - supp_{BN}(I,i)|, \tag{4}$$

where $supp_{Data}(I,i)$ is the support of the itemset in the dataset as defined in eqn- 1; and $supp_{BN}(I,i)$ is the support (i.e. joint probability) of the itemset relative to the Bayesian network *BN* as defined in eqn-3.

3.1.2 Interestingness of an attribute set relative to a Bayesian network

The qualitative structural assumptions pertaining to the dependencies among the nodes of a *BN* are modeled using attributes not itemsets; hence, we agree with [2] that it would be logical to think in terms of discovering interesting attribute sets rather than interesting itemsets. The *interestingness of an attribute set* relative to a Bayesian network is given by [2]

$$\mathcal{I}_{BN}(I) = max_{i \in Dom(I)} \mathcal{I}_{BN}(I,i). \tag{5}$$

Attribute sets with the highest interestingness scores are deemed interesting. A user specified threshold ε can be used to prune attribute sets with $\mathcal{I}_{BN}(I) < \varepsilon$.

4 Implementation and algorithms

The *PatternMiner* Java class (Algorithm-1) utilizes *BNs* in two ways: as a causality/dependence representation of all attributes in the user's preliminary set of beliefs (background knowledge); and as a probabilistic inference engine. *PatternMiner* adopts the *diverging supports* interestingness measure presented in sections 3.1 and 3.2. *PatternMiner* relies extensively on the use of hashtables and caching to prevent dual computation of support values/joint probabilities.

Calculating the joint probability of an itemset from a *BN* is known to be computationally hard. NeticaJ, a Java API of Netica (an application for belief networks), is used to calculate the support (joint probability) of a pattern. Hence, *PatternMiner* differs from [2] in two ways: It directly computes the joint probability for each and every frequent itemset instead of supersets of sets in the positive border; and it uses the *join tree* algorithm [13] for the joint probability (support) computation through Netica's *getFindingsProbability()* method instead of the bucket elimination algorithm [12].

Algorithm-1: *PatternMiner*
Input: A BN, a user specified *minSupport* and interestingness threshold ε
Output: Interesting attribute sets.

1. Find the frequent itemsets L_{Data}=*{(I,i)}* in the dataset using *Apriori.*
2. Find the frequent itemsets L_{BN}=*{(J,j)}* in the *BN* using *AprioriBND.*
3. Let $L_{\cap} = \{ L_{Data} \cap L_{BN} \}$
4. Compute the support of L_{BN} (i.e. $supp_{Data}(J,j)$) from the dataset using
5. *computeSupportData*
6. Compute the support of L_{Data} (i.e. $supp_{BN}(I,i)$) from the *BN* using
7. *computeSupportBN*
8. $L = \{ L_{Data} \cup L_{BN}\} - L_{\cap}$, i.e. $(I,i)=\{ (I,i) \cup (J,j) - L_{\cap} : I \neq J \}$
9. Compute $\mathcal{I}_{BN}(I,i) = |supp_{Data}(I,i) - supp_{BN}(I,i)|$ for all $(I,i) \in L$
10. Prune from $\mathcal{I}_{BN}(I,i)$ itemsets $\{(I,i) : \mathcal{I}_{BN}(I,i) < \varepsilon\}$
11. Compute $\mathcal{I}_{BN}(I) = max\{\mathcal{I}_{BN}(I,i) : (I,i) \in L, i \in Dom(I)\}$
12. Prune from $\mathcal{I}_{BN}(I)$ *attribute sets* $\{I : \mathcal{I}_{BN}(I) < \varepsilon\}$
13. Output top interesting attribute sets sorted by interestingness scores $\mathcal{I}_{BN}(I)$

PatternMiner takes as input the *BN* representing the background knowledge, which is learned from the dataset. It starts by finding the frequent itemsets in the dataset and the *BN* using a classical *Apriori* and *AprioriBND* (Algorithm-2), respectively. *AprioriBND* (D stands for Direct) is similar to the *AprioriBN* algorithm proposed in [2], but differs in the support computation of an itemset.

In step-4, frequent itemsets found in the *BN* (L_{BN}) are passed to *computeSupportData* (Algorithm-3) to compute their support in the dataset. Likewise, in step-6, frequent itemsets found in the dataset (L_{Data}) are passed to *computeSupportBN* (Algorithm-4) to compute their support in the *BN*. To enhance performance, *computeSupportData* (in step-3) and *computeSupportBN* (in step-4) were allowed to search hashtables of cached support values constructed during execution of *Apriori* and *AprioriBND*, to avoid dual computation of support values of common frequent itemsets ($L_{Data} \cap L_{BN}$).

Frequent itemsets in the dataset and the *BN* are merged in step-8 of Algorithm-1, before computing their interestingness scores in step-9. Only itemsets with an interestingness score $\geq \varepsilon$ are qualified to enter the competition for finding the attribute set with the maximum interestingness score in step-11.

Finally, *PatternMiner* sorts the attribute sets by their interestingness scores after pruning those < ε. Top ranked attribute sets are considered the most interesting.

Algorithm-2: *AprioriBND*
Input: *BN* and *minSupport*.
Output: Frequent itemsets $L_{BN}=\{(J,j)\}$ whose $supp_{BN}(J,j) \geq minSupport$; and $hashtable_{BN}$ for $L_{BN}=\{(J,j)\}$ containing respective $supp_{BN}(J,j)$

1. $L_0=\phi$; $k = 1$
2. $C_1=\{(J,j):|J|=1\}$
3. Compute the support $supp_{BN}(J,j)$ for all $(J,j) \in C_1$ using
4. *getFindingsProbability()* of Netica
5. Prune from C_1 itemsets $\{(J,j): supp_{BN}(J,j) < minSupport\}$
6. while $C_k \neq \phi$
7. $L_k = \{(J,j) \in C_k\}$
8. Generate new candidate set C_{k+1} *by using* L_k
9. Compute the support $supp_{BN}(J,j)$ for all $(J,j) \in C_{k+1}$ using
10. *getFindingsProbability()* of Netica
11. Prune from C_{k+1} itemsets $\{(J,j) : supp_{BN}(J,j) < minSupport\}$
12. $k=k+1$
13. $L_{BN} = \cup_k L_k$
14. Construct $hashtable_{BN}$ for $L_{BN}=\{(J,j)\}$ containing respective $supp_{BN}(J,j)$

Algorithm-3: *ComputeSupportData*
Input: $L_{BN}=\{(J,j)\}$, $L_\cap$ and $\text{hashtable}_{\text{Data}}$, constructed by the classical Apriori, for $L_{Data}=\{(I,i)\}$, which contains $supp_{Data}(I,i)$
Output: $supp_{Data}(L_{BN}=\{(J,j)\})$ *and* $cachedSupp_{BN}(L_\cap)$

1. while $L_{BN} \neq \phi$
2. (J,j) = an itemset from L_{BN}
3. if $(J,j) \in hashtable_{Data}(L_{Data}= \{(I,i)\})$ /* i.e. $(J,j) \in L_\cap$ */
4. $supp_{Data}(J,j) = supp_{Data}(I,i)$ from $hashtable_{Data}$: $(J,j) = (I,i)$
5. Cach $supp_{BN}(J,j)$ in $cachedSupp_{BN}(L_\cap)$
6. else /* $(J,j) \notin L_\cap$ */
7. Calculate $supp_{Data}(J,j)$ through a complete pass over the dataset

Algorithm-4: *ComputeSupportBN*
Input: Frequent itemsets $L_{Data}=\{(I,i)\}$, $L_\cap$, $hashtable_{BN}$ and $cachedSupp_{BN}(L_\cap)$
Output: $supp_{BN}(L_{Data}=\{(I,i)\})$

1. Prune from $hashtable_{BN}$ entries $\notin cachedSupp_{BN}(L_\cap)$
2. while $L_{Data} \neq \phi$
3. (I,i) = an itemset from L_{Data}
4. if $(I,i) \in hashtable_{BN}(L_\cap)$ /* i.e. $(I,i) \in L_\cap$ */
5. $supp_{BN}(I,i) = supp_{BN}(J,j)$ from $hashtable_{BN}$: $(I,i) = (J,j)$
6. else /* $(I.i) \notin L_\cap$ */
7. Enter (I,i) to the *BN* as findings using *enterFinding()* of Netica
8. Calculate $supp_{BN}(I,i)$ *using getFindingsProbability()* of Netica

The computational complexity of *PatternMiner* is governed by:

- The number of frequent itemsets $|L|$ generated by the Apriori and *AprioriBND*, which is highly governed by the user specified *minSupport* parameter and the maximum size (max_k) allowed for an itemset/attribute set; i.e. the number of items/attributes within a pattern.
- The size of the Bayesian network, i.e. the number of nodes N in the net.
- The size of the dataset.

The time and space complexity of the *join tree* algorithm is exponential in the worst case (so is the bucket elimination algorithm, according to [12]). Hence, the most costly part of *PatternMiner* is *getFindingsProbability()*; it is called in step-9 of *AprioriBND* for every frequent itemset $(J,j) \in L_{BN}$; and in step-8 of *ComputeSupportBN* for every frequent itemset $\{(I,i) : (I,i) \in L_{Data}, (I,i) \notin L_{\cap}\}$. This cost escalates with large values of $|L|$ and N; but with $|L|$ being the major overhead even for small values of N. For this reason, the *minSupport* parameter should be specified wisely so as not to explode the number of frequent itemsets worthlessly. Experimenting with different *minSupport* values witnessed a vast reduction in computation time (reaching 92.0%!) when *minSupport* was increased, and the same top most interesting patterns were still discovered.

Table 1: Time performance of *PatternMiner* and the marginalization algorithm.

Dataset	Max Size	minSupp	ε	*PatternMiner* Java Class			Marginalization Alg.		
				#Joint dist	Time [s]	Max $\mathcal{I}_{BN}$	#Joint dist	Time [s]	Max $\mathcal{I}_{BN}$
KSL (N=9)	5	0.1	0.01	1228	**0.21**	0.032	-	-	-
	5	0.05	0.01	3309	**0.62**	0.032	-	-	-
	5	0.01	0.01	13120	**2.55**	0.032	382	**1.12**	0.032
Breast-Cancer (N=10)	5	0.1	0.01	1274	**0.30**	0.82	-	-	-
	5	0.05	0.01	3643	**0.63**	0.82	-	-	-
	5	0.01	0.01	27469	**5.57**	0.82	638	**3.49**	0.082
Lympho-graphy (N=19)	3	0.1	0.067	13845	**2.93**	0.123	-	-	-
	3	0.067	0.067	**24826**	**5.34**	**0.123**	1160	**29.12**	**0.123**
	4	0.1	0.067	26161	**7.40**	0.126	-	-	-
	4	0.067	0.067	**53988**	**18.6**	**0.126**	5036	**106.1**	**0.126**
Splice (N=61)	3	0.1	0.01	32845	**13.7**	0.028	-	-	-
	3	0.075	0.01	93969	**57.6**	0.028	-	-	-
	3	0.05	0.01	**1521332**	**1540.7**	0.036	-	-	-
	3	0.01	0.01	-	-	-	37882	**8456**	0.036

5 Performance evaluation

The performance of *PatternMiner* is evaluated using four datasets used previously in [2]: the KSL dataset of Danish 70-year olds, Breast Cancer, Lymphography and Splice datasets. The KSL dataset is distributed with the DEAL package, while the other three are from the UCI Machine Learning repository. The respective *BNs* for those datasets were thankfully provided by the authors of [2] to facilitate a smooth performance comparison. Table 1 below, presents the results obtained along with a comparison with the time performance of the algorithm in [2], which is referred to as the 'marginalization algorithm'.

The 'max size' column indicates the user specified size limit of frequent itemsets generated by the *AprioriBND* and *Apriori* algorithms. The 'minSupport' column was allowed to vary for the same dataset, the same 'max size' and the same interestingness threshold 'ε', to show the great reduction in computation time achieved, with the same maximum interestingness values (max $\mathcal{I}_{BN}$) still discovered. Naturally, the number of joint distributions ('#Joint dist' column) processed, and the computation time ('Time[s]' column), both depend on the respective 'minSupport' and 'max size' parameters specified. The '#Joint dist' grows (and sometimes explodes) with low 'minSupport' values and large 'max size' values. The time reported in seconds does not include the execution of the *Apriori* algorithm (to comply with [2]), but it does include *AprioriBND*.

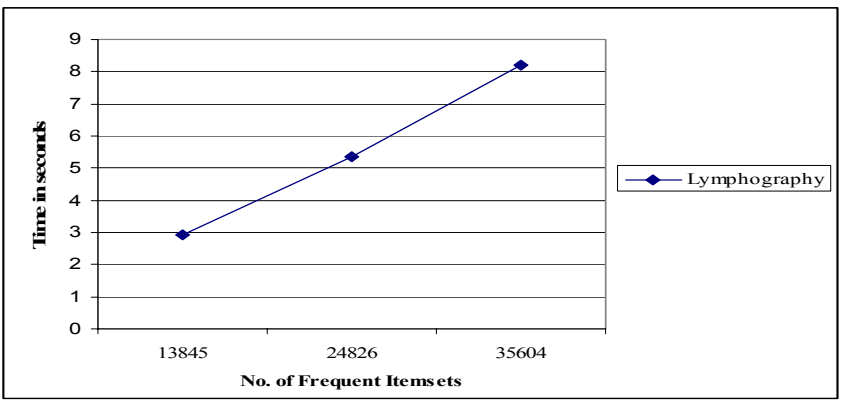

Figure 1: Computation time of *PatternMiner* Vs. the number of frequent itemsets for the Lymphography dataset, N=19.

Figure 1 shows how *PatternMiner* scales linearly, using the Lymphography dataset (19 attributes), against the growing number of frequent itemsets as a result of decreasing the 'minSupport' parameter (.1, .067 & .05). Unfortunately, such a linear behaviour is not guaranteed all the time, because there are many factors governing the computational complexity, which include the *BN* structure, no. of attributes, size of the domain, size of the dataset and the max size of the frequent itemsets. This is expected because *PatternMiner* leverages the *join tree* algorithm whose complexity is exponential in the worst case. Figure 2 below

resembles such a case using the Splice dataset (61 attributes), where *PatternMiner* at first scales linearly for relatively smaller numbers of frequent itemsets, but exponentially for exploding numbers of frequent itemsets, as the 'minSupport' parameter is decreased from 0.1 to .05 (refer to Table 1).

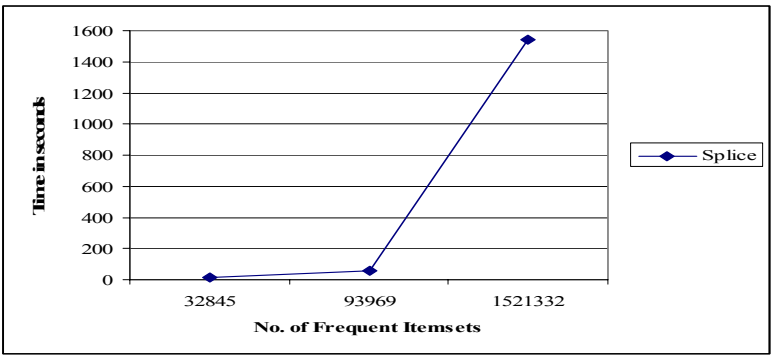

Figure 2: Computation time of *PatternMiner* Vs. the number of frequent itemsets for the Splice dataset, N=61.

For the Splice dataset, *PatternMiner* was able to produce a result for minSupport=.05 only when Java's heap size was increased at run time. It computed 1,521,332 joint distributions in 1541 seconds as opposed to 8456 seconds needed by the marginalization algorithm to compute 37,882 joint distributions; i.e. *PatternMiner* slashed the computation time by ~82% for a dataset of 61 attributes! Such an excelling performance is ascertained by Figure 3, where the computation time of *PatternMiner* is compared to the that of the marginalization algorithm, using the Lymphography dataset, for the same values of 'max size'; noting that this value masks a large difference in the numbers of joint distributions each algorithm has to process. For example, for 'max size'=4, *PatternMiner* has computed 53,988 joint distributions in 18.61 seconds as opposed to 5,036 joint distributions in 106.13 seconds by the marginalization algorithm. Thus, again a vast reduction in computation time was achieved reaching ~82% with the same maximum interestingness values still discovered!

However, the scalability of *PatternMiner* remains a concern. It couldn't produce a result for the Splice dataset when minSupport=0.01 (refer to last row in Table 1); while the marginalization and approximation [11] algorithms could. Nevertheless, *PatterMiner* exhibited an excellent performance that outperformed not only the marginalization algorithm, but also the approximation algorithm as shown in Table 2 below; it consumed ~1541 seconds for the Splice dataset as opposed to 1795 seconds by the approximation algorithm.

6 Conclusion

An efficient Java algorithm, *PatternMiner*, was presented that discoveres interesting unexpected patterns based on background knowledge represented by

a Bayesian network (BN). *PatternMiner* profiles a pattern as *unexpected,* if the absolute difference between the pattern's support estimated from the dataset and the BN exceeds a user specified threshold. Patterns with the highest diverging supports are considered the most interesting.

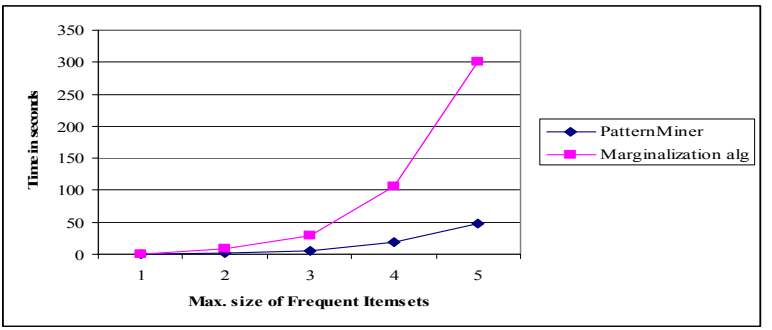

Figure 3: Computation time of *PatternMiner* Vs. the marginalization algorithm, using the Lymphography dataset, N=19.

Table 2: Time performance of *PatternMiner* compared to the approximation algorithm.

Dataset	N	Max Size	minSupp	ε	*PatternMiner*		Approximation alg	
					Time [s]	Max $\mathcal{I}_{BN}$	Time [s]	Max $\mathcal{I}_{BN}$
KSL	9	5	0.01	0.01	**2.55**	0.032	**55**	0.032
Lympho-graphy	19	3	0.067	0.067	**5.34**	0.123	**43**	0.099
		4	0.067	0.067	**18.6**	0.126	**83**	0.123
Splice	61	3	0.05	0.01	**1540.7**	0.036	**-**	-
		3	0.01	0.01	**-**	-	**1795**	0.036

Computing the support (joint probability) of a pattern from a BN is known to be computationally hard. *PatternMiner* adopted the *joint tree* algorithm for probabilistic inference using NeticaJ, a Java API of Netica. For moderately-sized datasets and BNs, *PatternMiner* has significantly outperformed other existing algorithms [2, 11] adopting the 'diverging supports' measure, where it was able to mark a slash in computation time reaching 82%.

As the scalability of *PatternMiner* remains a concern, it would be a worthwhile future direction to benefit from the more scalable approaches in [2, 11] when needed, by equipping *PatternMiner* with an inference optimizer that profiles the BN and dataset on hand before choosing the most feasible inference approach out of the three. Implementing the algorithms in [2, 11] using Java is an important starting point to assess how much Java's efficient environment can contribute to boosting their performance.

Acknowledgement

We would like to thank Szymon Jaroszewicz, one of the authors of [2, 11] for his feedback and provision of the nets used in [2], to facilitate a smooth comparison.

References

[1] Agrawal, R., Imielinski, T. & Swami, A., Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD Conf. on Management of Data*, pp. 207 216, 1993.

[2] Jaroszewicz, S. & Simovici, D., Interestingness of frequent itemsets using Bayesian networks as background knowledge. *In Proc. of the 2004 ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-2004)*, pp. 178-186, 2004.

[3] Chen. S., Hsu, W. & Liu., B., Using general impressions to analyze discovered classification rules. *In Proc. of the 3rd Int'l Conf. on Knowledge Discovery and Data Mining (KDD-1997)*, pp. 31, 1997.

[4] Chen, S., Hsu, W., Liu, B. & Ma, Y., Analyzing the subjective interestingness of association rules. *IEEE Intelligent Systems*, v.5: pp. 47 55, 2000.

[5] Klemettinen, M., Mannila, H., Ronkainen, P. & Verkamo, A.I., Finding interesting rules from large sets of discovered association rules. *In Proc. of the 3rd Int'l Conf. on Information & Knowledge Manag.*, pp. 401 407, 1994.

[6] Sahar, S., Interestingness via what is not interesting. *In the 5th Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 332 336, 1999.

[7] Padmanabhan, B. & Tuzhilin, A., A belief-driven method for discovering unexpected patterns. *In Proc. of the 4th Int'l Conf. on Knowledge Discovery and Data Mining*, 1998.

[8] Padmanabhan, B. & Tuzhilin, A., Small is beautiful: Discovering the minimal set of unexpected patterns. *In Proc. of the 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 54 63, 2000.

[9] Silberschatz, A. & Tuzhilin, A., On subjective measures of interestingness in knowledge discovery. *knowledge Discovery & Data Mining*, 1995.

[10] Silberschatz, A. & Tuzhilin, A., What makes patterns interesting in knowledge discovery systems. *IEEE Trans. In Knowledge and Data Engineering. Special issue on Data Mining*, v.5, no.6: pp. 970 974, 1996.

[11] Jaroszewicz, S. & Scheffer, T., Fast discovery of unexpected patterns in data, relative to a Bayesian network. *In Proc. Of the 2005 ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-2005)*, 2005.

[12] Dechter, R., Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, v.113, no.1&2: pp. 41 85, 1999.

[13] Cowell, R.G., Dawid, A.P., Lauritzen, S.L. & Spiegelhalter, D.J., Bayesian analysis in expert systems. *Statistical Science*, v.8, no.3: pp. 219 283, 1993.