A new constraint for mining sets in sequences

Boris Cule* Bart Goethals† Céline Robardet‡

Abstract

Discovering interesting patterns in event sequences is a popular task in the field of data mining. Most existing methods try to do this based on some measure of cohesion to determine an occurrence of a pattern, and a frequency threshold to determine if the pattern occurs often enough. We introduce a new constraint based on a new interestingness measure combining the cohesion and the frequency of a pattern. For a dataset consisting of a single sequence, the cohesion is measured as the average length of the smallest intervals containing the pattern for each occurrence of its events, and the frequency is measured as the probability of observing an event of that pattern. We present a similar constraint for datasets consisting of multiple sequences. We present algorithms to efficiently identify the thus defined interesting patterns, given a dataset and a user-defined threshold. After applying our method to both synthetic and real-life data, we conclude that it indeed gives intuitive results in a number of applications.

1 Introduction

Discovering interesting episodes [8] is a popular area in temporal or sequential data mining, examples of which are mining text or protein sequences. In such data, the order in which the events appear is being analysed and the user's goal is to identify the regularities that may appear in the dataset, consisting of one or more sequences. The usual approach to episode discovery is to look for episodes consisting of events that frequently appear close to each other. Most of the current state-of-the-art methods first use a window of fixed length to find sufficiently cohesive episodes and then retrieve those that occur in more windows (or sequences) than a given minimum threshold [8]. The use of a window of fixed length is a major limitation of such approaches as no episodes longer than this window can ever be discovered. A different method that increases the window length proportionally to the size of the candidate set has been proposed in order to remove this limitation [4]. Still, in this proposal, the window length remains fixed for a particular candidate when counting its frequency in the sequence. Hence, when the episode occurs in the sequence, but in a time frame larger than the window size, then such occurrences will be disregarded. The high frequency of a set of events appearing close together gives no guarantee that a subset of that set will not sometimes appear far away from the rest of the set.

In this paper we propose a new constraint to select interesting sets of events. We focus on parallel episodes which are unordered sets of events, and these can thus be considered as sets of items, better known as itemsets. When looking at one sequence, we define the interestingness of an itemset based on a combination of how often the items in the candidate set appear in the sequence and how close to each other they appear on average. Later on, we present a similar definition for datasets consisting of many sequences. Hence, this approach does not use a fixed window length but instead also takes the occurrences of the items far from the rest of the set into account. It is precisely such occurrences that might sufficiently lower the cohesion of an itemset to render it uninteresting.

As we will show, the introduced interestingness measure has useful properties that allow us to develop an efficient algorithm to search for interesting itemsets, depending on a user-defined threshold. We present a divide-and-conquer method and prune the search space by computing an upper bound on the interestingness measure for potential candidate itemsets. We apply our approach to various datasets consisting of both a single sequence and of multiple sequences. We use synthetic and real-life datasets to test the intuitiveness of our method and the efficiency of our algorithms.

The paper is organized as follows. In Section 2, we present the most relevant related works and illustrate their weaknesses on a couple of toy examples. Section 3 gives the full description of our interestingness constraint for datasets consisting of a single sequence and, using the same examples, demonstrates its relevance and intuitiveness. Section 4 presents a short sketch of our algorithm including a pruning method. Section 5 demonstrates the effectiveness of our algorithm applied on synthetic and real-life datasets. In Section 6 we extend our definitions to multiple sequences,

*University of Antwerp, Belgium
†University of Antwerp, Belgium
‡Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205, F-69621, France

give an algorithm for this new setting, and present the results of further experiments. Finally, in Section 7 we present our conclusions and highlight some possibilities for future work.

2 Related works

Looking for frequent episodes in an event sequence was first proposed by Mannila et al. Each event is associated with a time stamp. Their Winepi algorithm [7, 8] finds all episodes that occur in a sufficient number of windows of fixed length. This window length is chosen by the user indicating how close to each other the events of an interesting episode should be. The frequency of an episode is defined as the number of windows in which the episode occurs divided by the total number of possible windows of chosen length.

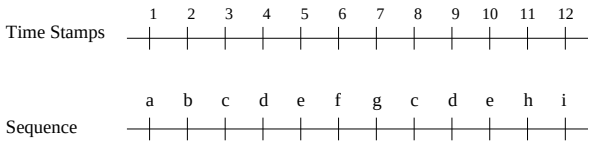


Figure 1: An illustrative example.

As was pointed out by Garriga [4], Winepi suffers from bias against longer episodes. Figure 1 shows a sequence, in which episode *cde* appears twice, as does its subepisode *cd*. Clearly, to a user, a longer episode would be more interesting than a shorter one, but applying Winepi (with a window length greater than 1) to this sequence will result in *cd* having a larger frequency than *cde*. For example, using a window of length 3 gives:

$$\begin{aligned} fr(cd) &= \frac{4}{14} \\ fr(cde) &= \frac{2}{14} \end{aligned}$$

This is clearly unintuitive. Garriga [4] proposes to solve this problem by increasing the window length proportionally to the episode length, using a parameter *tus* equal to the maximal gap allowed between two events in an episode. Therefore, using $tus = 2$ (equivalent to using a window of length 3 for an episode made of two event types) would result in a window of length 5 for episodes made of three event types. Frequencies of *cd* and *cde* are in this context

$$\begin{aligned} fr(cd) &= \frac{4}{14} \\ fr(cde) &= \frac{8}{16} \end{aligned}$$

For episodes of equal length, Winepi and Garriga methods give equal results when the chosen *tus* results

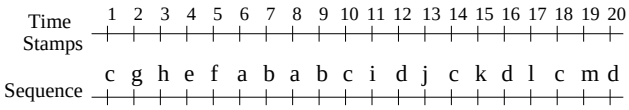


Figure 2: A second example

in the window length chosen for Winepi. However, these results are also not always intuitive. The example in Figure 2 illustrates that frequency alone is not sufficient to estimate an episode's interestingness. For example, event types c and d appear relatively close to each other more often than event types a and b, and yet the episode ab seems to be more interesting than cd since its event types always occur close to each other. At the same time, event types e and f also always occur close to each other but their low frequency makes episode ef less interesting than cd.

The above mentioned methods of discovering interesting episodes do not take this into account. If we apply the Winepi method on the example of Figure 2 with a window of length 3, we obtain the following frequencies:

$$\begin{aligned} fr(cd) &= \frac{5}{22} \\ fr(ab) &= \frac{4}{22} \\ fr(ef) &= \frac{2}{22} \end{aligned}$$

The ranking will be the same for all windows of length greater than 3. The same method with a window of length 2 would give:

$$\begin{aligned} fr(cd) &= 0 \\ fr(ab) &= \frac{3}{21} \\ fr(ef) &= \frac{1}{21} \end{aligned}$$

We can observe that no chosen window length gives the desired result. The method of Garriga [4] would give the same results depending on the used parameter. This is not a surprise, since both Winepi and Garriga methods are trying to find how likely we are to encounter an episode, while it may be more interesting to find how likely we are to encounter an event from an episode, and, once we encountered it, how likely we are to encounter the other events of the episode nearby.

Hence, both Winepi and Garriga ignore the first c in the example in Figure 2, while precisely that c plays a pivotal role in our method, indicating that, once we

encountered a c, we are not always likely to find a d nearby.

An often mentioned method when looking for interesting episodes in event sequences is MINEPI [6, 8]. In this method, a minimal occurrence of an episode is defined as a window in which the episode occurs such that the episode does not occur in any of its proper subwindows. The support of an episode is then defined as the number of such windows.

Two main disadvantages of MINEPI are the use of support instead of a frequency ratio making it difficult for the user to make the correct choice, and also the fact that the method completely disregards the distance between the items in an episode.

Looking for frequent patterns in multiple sequences was first proposed by Wang et al. in [9] and later extended by Agrawal and Srikant in [1]. However, the problem definitions in these approaches differ greatly from that presented here, and a meaningful comparison cannot be made.

3 Problem Setting

As mentioned in the introduction, parallel episodes can also be looked at as itemsets. We therefore consider an event to be a couple consisting of an item and a time stamp (i, t) where $i \in I$, the set of all possible items, and $t \in \mathbb{N}$. For the purposes of this paper, we consider all consecutive events to be equidistant and we assume that two events can never occur at the same time. In our notation, therefore, without any loss of generality, we consider the set of time stamps of an event sequence to be an uninterrupted sequence of natural numbers. We denote a sequence of such events by $\mathcal{S}$.

3.1 Definition of the constraint In this section, we introduce our interestingness measure that makes it possible to find itemsets that appear frequently in the sequence and consist of items that, on average, appear close to each other. The interestingness of an itemset will therefore depend on two factors: its coverage and its cohesion. Coverage measures how often an item of the itemset appears in the sequence, while cohesion measures how close together the items making up the itemset appear on average.

For a given itemset X, we denote the set of all occurrences of its items as

$$N(X) = \{t \mid (i, t) \in \mathcal{S} \text{ and } i \in X\}$$

The coverage of X can now be defined as:

$$P(X) = \frac{|N(X)|}{|\mathcal{S}|}$$

In order to calculate the cohesion, we must first evaluate the length of the shortest interval containing the itemset X for each position of $N(X)$:

$$W(X, t) = \min\{t_2 - t_1 + 1 \mid t_1 \leq t \leq t_2 \text{ and } \forall i \in X, \exists (i, t') \in \mathcal{S},\ t_1 \leq t' \leq t_2\}$$

We can now compute the average length of such shortest intervals:

$$\overline{W}(X) = \frac{\sum_{t \in N(X)} W(X, t)}{|N(X)|}$$

It is clear that $\overline{W}(X)$ is greater than or equal to the number of items in X. Furthermore, for a fully cohesive itemset, where no other events ever occur between the items making the itemset, $\overline{W}(X) = |X|$. To normalize, we want the cohesion of a fully cohesive itemset to be equal to 1, and the cohesion of other itemsets to be lower. Therefore, we define cohesion of X as

$$C(X) = \frac{|X|}{\overline{W}(X)}$$

We can now define the interestingness of an itemset X as

$$I(X) = C(X)P(X) = \frac{|N(X)| \times |N(X)| \times |X|}{\sum_{t \in N(X)} W(X, t) \times |\mathcal{S}|}$$

Note that both $C(X)$ and $P(X)$ are between 0 and 1, and the same is therefore true for $I(X)$. This makes it possible to apply the same interestingness threshold to any kind of dataset. Given a user defined threshold min_int, an itemset X is considered interesting if

$$I(X) \geq min\_int$$

In principle, the user can enter a coverage threshold (min_cov) and/or a cohesion threshold (min_coh) as separate parameters. In such a case, an interesting itemset must satisfy all of the following:

$$P(X) \geq min\_cov$$

$$C(X) \geq min\_coh$$

$$I(X) \geq min\_int$$

3.2 Computation example We will now apply our interestingness measure on the sequences given in Figures 1 and 2.

In the example of Figure 1, cd and cde are both fully cohesive, but cde clearly covers more of the sequence than cd, and will therefore be considered more interesting. According to our definition, $P(cd) = \frac{4}{12}$

and $P(cde) = \frac{6}{12}$. As cd and cde are both fully cohesive, $C(cd) = 1$ and $C(cde) = 1$. Therefore, $I(cd) = \frac{4}{12}$ and $I(cde) = \frac{6}{12}$.

In general, an itemset will always have a lower coverage than any of its supersets, and, provided that they are equally cohesive, will always have a lower interestingness than the superset.

On the other hand, we should note that the cohesion of an itemset alone is also not suitable for estimating its interestingness. In the example of Figure 2, ef is just as cohesive as ab, but its frequency is too small to be considered more interesting than cd. This is why our interestingness measure consists of combining both coverage and cohesion.

We shall now illustrate how our method computes the interestingness of itemset cd. The positions of the items c and d are in this case

$$N(cd) = \{1,\ 10,\ 12,\ 14,\ 16,\ 18,\ 20\}$$

which leads to the following coverage:

$$P(cd) = \frac{7}{20}$$

Then we compute the minimal interval around each position in $N(cd)$ containing both c and d:

$$\begin{aligned} W(cd,1) = length([1,12]) + 1 &= 12 \\ W(cd,10) = length([10,12]) + 1 &= 3 \\ W(cd,12) = length([10,12]) + 1 &= 3 \\ W(cd,14) = length([12,14]) + 1 &= 3 \\ W(cd,16) = length([14,16]) + 1 &= 3 \\ W(cd,18) = length([16,18]) + 1 &= 3 \\ W(cd,20) = length([18,20]) + 1 &= 3 \end{aligned}$$

The average length of these intervals is

$$\overline{W}(cd) = \frac{30}{7}$$

The cohesion is thus equal to

$$C(cd) = \frac{2 \times 7}{30} = \frac{7}{15}$$

Finally, the interestingness is

$$I(cd) = \frac{7}{15} \times \frac{7}{20} = \frac{49}{300}$$

The summary of the computation of the interestingness for the 3 itemsets previously considered in Section 2 is given in Table 1.

Comparing these results with those obtained in Section 2, it can be observed that our method gives more intuitive results than WINEPI and Garriga.

	$\lvert N(X)\rvert$	$\overline{W}(X)$	$C(X)$	$P(X)$	$I(X)$
ab	4	2	1	0.2	0.2
cd	7	4.29	0.47	0.35	0.16
ef	2	2	1	0.1	0.1

Table 1: Computation of interestingness for example of Figure 2

4 Algorithm sketch

To compute all the itemsets that are interesting, our algorithm generates candidates and uses a pruning technique based on an upper-bound of the constraint to reduce, as soon as possible, the search space. These two main steps are detailed below.

4.1 Candidate generation We generate candidates by applying the "divide-and-conquer" algorithm published in [2, 5, 3]. In a nutshell, the algorithm recursively enumerates, in depth-first manner, all itemsets containing an element, say a, and then all itemsets not containing a. During the enumeration process, a candidate $\langle X, Y\rangle$ is composed of two sets of items: one, denoted X, that contains the items contained in the candidate and its descendants (the ones obtained by recursive calls) and another one, denoted Y, that contains the items that still have to be enumerated.

Algorithm 1 DFS $(\langle X, Y\rangle)$

```
if UBI(⟨X,Y⟩) = false then
  if Y = ∅ then
    output X
  else
    Choose a in Y
    DFS(⟨X ∪ {a}, Y \ {a}⟩)
    DFS(⟨X, Y \ {a}⟩)
  end if
end if
```

The pseudo code of the DFS algorithm we have implemented is given in Algorithm 1. At the first line, the pruning function presented below in Subsection 4.2 is called to check if the recursion process can be stopped. If the recursion is not stopped, the second test evaluates if there are still elements to be enumerated. If not, the candidate is an interesting episode that is thus output. Otherwise, an element a is picked up from Y and the function is recursively called twice: once with a in X, and once without. In both calls, a is removed from Y. For the first call, X is empty and Y is equal to the set of all items appearing in the sequence.

For efficiency, we sort the items with respect to

their frequency in descending order. The logic behind this decision is that to prune a candidate $\langle X, Y\rangle$ most efficiently, we would need to encounter the set $X' \subseteq X \cup Y$ such that $\sum_{t\in N(X')} W(X', t)$ is sufficiently high to trigger the pruning. An heuristic to increase the chance of encountering it early is to generate candidates using items sorted with respect to their frequency in descending order, because if the frequency of an itemset is higher than that of another itemset then its corresponding sum of interval lengths is also likely to be higher.

This algorithm satisfies the following useful properties:

1. The time needed to generate an interesting itemset is, in the worst case, bounded by $O(T \times |I|^2)$, where I is the set of all possible items and T is the time complexity of computing the pruning function $UBI(\langle X, Y\rangle)$ as explained in Section 4.2. This function can be computed in $O(|\mathcal{S}|^2)$.
2. It supports pruning of monotonic and anti-monotonic functions, that is to say that additional monotonic constraints can be easily handled.

In addition to these advantageous features, the algorithm is easy to understand and implement.

4.2 Pruning Recall that an interesting itemset must satisfy the following:

$$I(X) = \frac{|N(X)| \times |N(X)| \times |X|}{\sum_{t\in N(X)} W(X, t) \times |\mathcal{S}|} \geq min\_int$$

Starting from a given candidate $\langle X, Y\rangle$, it is clear that for each Z such that $X \subseteq Z \subseteq X \cup Y$ the following inequalities stand:

$$\begin{aligned} |N(Z)| &\leq |N(X \cup Y)| \\ |Z| &\leq |X \cup Y| \\ \sum_{t\in N(X)} W(X, t) &\leq \sum_{t\in N(Z)} W(Z, t) \end{aligned}$$

It therefore follows that

$$I(Z) \leq \frac{|N(X \cup Y)| \times |N(X \cup Y)| \times |X \cup Y|}{\sum_{t\in N(X)} W(X, t) \times |\mathcal{S}|}$$

Because all such Z are generated by recursive calls of DFS starting from candidate $< X, Y >$, we can safely prune all itemsets Z that satisfy $X \subseteq Z \subseteq X \cup Y$ if

$$UBI(\langle X, Y\rangle) \equiv \frac{|N(X \cup Y)|^2 \times |X \cup Y|}{\sum_{t\in N(X)} W(X, t) \times |\mathcal{S}|} < min\_int$$

by suppressing further recursive calls.

If minimal cohesion is given as a parameter (min_coh), we can achieve further pruning. Using the above inequalities, we find that

$$C(Z) \leq \frac{|N(X \cup Y)| \times |X \cup Y|}{\sum_{t\in N(X)} W(X, t)}$$

We can therefore prune Z if

$$\frac{|N(X \cup Y)| \times |X \cup Y|}{\sum_{t\in N(X)} W(X, t)} < min\_coh$$

The crucial step in evaluating $UBI(\langle X, Y\rangle)$ is the computation of the intervals $W(X, t)$ for the relevant time stamps t, i.e. those time stamps at which an item of X occurred. In our implementation, we keep the set of intervals associated with X in a list. When we have generated a candidate $\langle X \cup \{a\}, Y \setminus \{a\}\rangle$ from $\langle X, Y\rangle$, we start its evaluation by updating, if necessary, the set of intervals corresponding to X, i.e. $\{W(X, t) \mid t \in N(X)\}$. In this step, the intervals that do not contain an occurrence of a are removed from the list and new minimal intervals for the corresponding time stamps are computed.

To find the minimal interval around position t containing all elements of $X \cup \{a\}$, we start by looking for the nearest occurrences of elements of X both left and right of position t. We then start reading from the side on which the furthest element is closest to t and continue by removing one element at a time and adding the same element from the other side. This process can stop when the interval on the other side has grown sufficiently to make it impossible to improve on the minimum we have found so far. When we have found this minimal interval, we can add it to the list.

We then proceed by generating the intervals corresponding to each occurrence of a in the event sequence. These intervals are generated in exactly the same way as the new intervals in the previous step.

As many intervals occur many times, we store the list of distinct intervals together with a list of positions each one applies to.

In the worst case, the number of minimal intervals that need to be found can be equal to the number of items in the sequence, $|\mathcal{S}|$. To find an interval, we might need to read the whole sequence both to the left and to the right of the item. Therefore, the time needed to evaluate the pruning function is $O(|\mathcal{S}|^2)$. It is worth noting that this worst case can actually materialise only if we are evaluating the itemset consisting of all items that appear in the sequence, and even then only if items appearing at each end of the sequence do not appear anywhere else.

5 Experiments

In this section, we present the results of our experiments performed on various synthetic and real-life datasets, and analyse their implications.

5.1 Synthetic datasets Fully random datasets would not be suitable for our purposes because all itemsets of equal length would be likely to have similar interestingness values. A more suitable dataset would be one generated using a Markov chain model, which enables us to increase the likelihood of certain items appearing close together and thus forming interesting itemsets.

In our experiments, we use a memoryless model, also called a simple random walk, in which the next item depends only on its immediate predecessor. We can therefore easily describe such a model using a transition matrix. The transition matrix used to generate the dataset is given in Table 2. Our goal is to generate a sequence in which *abc* would be an interesting pattern hidden among occurrences of several other items denoted x in the table. When a transition leads to an occurrence of x, a random item is picked from a group of 25 items, not including a, b or c. Doing this ensures that none of these 25 items will form interesting itemsets.

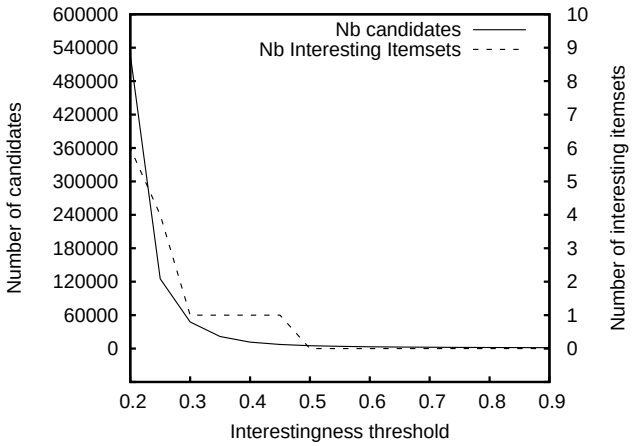
	a	b	c	x
a	0.2	0.35	0.35	0.1
b	0.35	0.2	0.35	0.1
c	0.35	0.35	0.2	0.1
x	0.05	0.05	0.05	0.85

Table 2: Transition matrix defining a Markov model

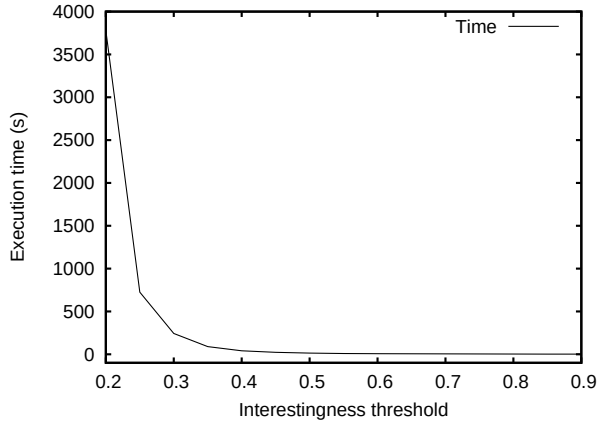
Using this model, we generated a sequence of 2000 events. We then ran our algorithm varying the interestingness threshold *min_int* from 0.9 down to 0.2, 0.05 at a time. In all experiments no thresholds were set for coverage and cohesion separately. The results of our experiments can be seen in Figures 3(a) and 3(b).

Figure 3(a) shows that as long as there are few interesting itemsets to be found, our pruning method works very efficiently. The most interesting itemset *abc* is found using a relatively high interestingness threshold of 0.45. No further interesting itemsets can be found without lowering the threshold below 0.3, which confirms the intuitiveness of our method. With the interestingness level of 0.25, itemsets *ab*, *ac* and *bc* are also discovered as interesting. Because of the uniform distribution of the remaining 25 items, the number of considered candidates grows significantly once the threshold is lowered further.

We can observe in Figure 3(b) that the execution time grows proportionally with the number of generated candidates indicating that the most interesting itemsets can be discovered in a reasonable amount of time. To determine the most appropriate interestingness threshold value for a given dataset, an inexperienced user can start with a high value and decrease it until the first results come through. Running the algorithm with a high interestingness threshold takes only few seconds.



(a) number of candidates and found itemsets



(b) execution time

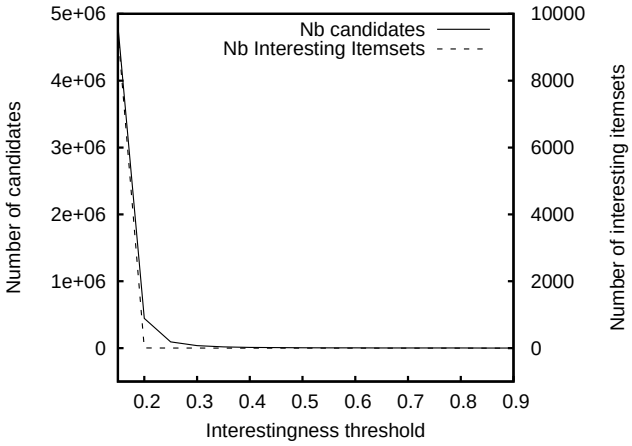
Figure 3: (a) Number of generated candidates and interesting itemsets for the Markov chain generated data, with varying *min_int*. (b) Execution time in seconds for the Markov chain generated data, with varying *min_int*.

Analysing the generated dataset, we observed that the coverage of *abc* was around 0.6. Its interestingness was around 0.46, which means its cohesion was around 0.77. In order to show that the high interestingness of *abc* was not only due to its coverage, we investigate what is obtained in a randomly ordered sequence in which

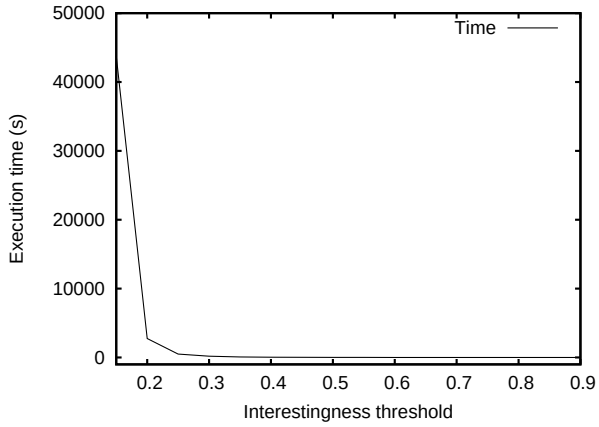
items have the same frequency as in the Markov Chain dataset. We generated such a sequence of 2000 events made of 28 items.

As we can see in Figure 4(a), we have to lower the interestingness threshold much further in order to get the first results. Itemset *abc* is found to have interestingness of around 0.3 indicating a cohesion of 0.5.

Figure 4(b) shows that, once again, the execution time is fully proportional with number of generated candidates.



(a) number of candidates and found itemsets



(b) execution time

Figure 4: (a) Number of generated candidates and interesting itemsets for the RANDOM dataset, with varying *min_int*. (b) Execution time in seconds for the RANDOM dataset, with varying *min_int*.

To test how much the execution time depends on the size of the sequence, we generated several Markov chain datasets of different sizes, varying between 1000 and 25000. For each size, we generated 10 datasets and applied our algorithm with $min\_int = 0.3$, because it is around this threshold that the first results usually appeared.

Figure 5 shows the obtained results. Each vertical line corresponds to one sequence length showing the mean execution time of the 10 algorithm runs and their standard deviation. It can be observed that the execution time depends much more on the structure of the dataset than on its size, even for such similarly structured datasets (all the datasets contained exactly the same number of different items that were distributed according to the same transition matrix).

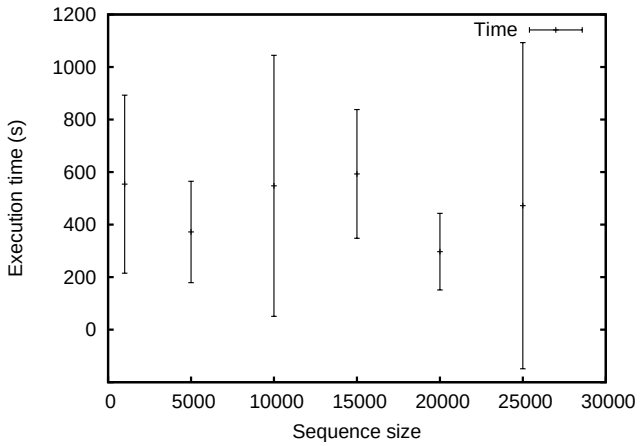


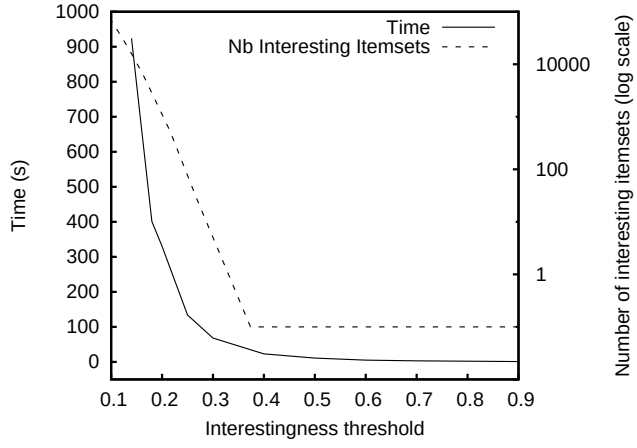
Figure 5: Execution time in seconds for the Markov chain generated data, when $min\_int = 0.3$ and the sequence length is varying.

5.2 Real-life datasets The first real-life dataset we used is one obtained from the amino-acid sequence from human alcohol dehydrogenase [10]. This enzyme has been heavily researched and its genomic sequence is widely available and therefore reliable[1]. The sequence consists of 20 different amino-acids making a chain of 375.

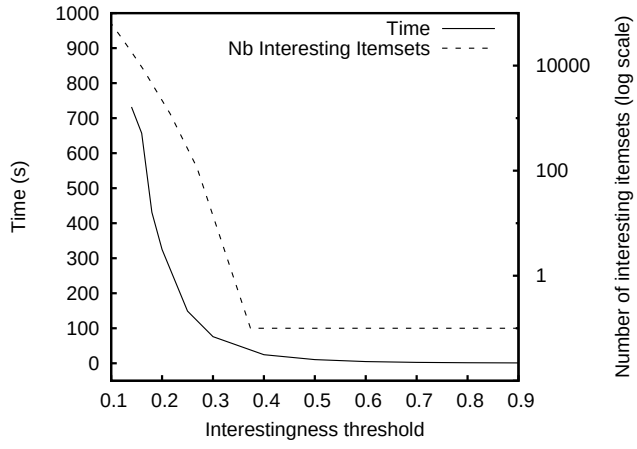
In Figure 6(a), we can see that we get some results with an interestingness value below 0.4, but no itemset really sticks out, suggesting a uniform distribution of items in the sequence. It turns out that the most interesting itemsets are quite large and can be found in a reasonable amount of time. These results were not surprising, as biologists confirmed that items were distributed in such a way. To double-check, we once again created a synthetic dataset, 375 items long, where items were distributed randomly, using their frequency

[1] http://www.expasy.org/cgi-bin/sprot-ft-details.pl?P07327@SEQUENCE@2@375

in the DNA dataset to determine their probability of occurrence. Figure 6(b) shows that the results obtained on this dataset are very similar to those in Figure 6(a).



(a) DNA dataset



(b) random DNA-like dataset

Figure 6: (a) Execution time in seconds and number of extracted itemsets (log-scale) for the DNA dataset, with varying *min_int*. (b) The same for the random DNA-like dataset.

Our second dataset consisted of music notes, collected from one track of a midi-file. We used the notes of the song Abide With Me[2], having first converted the midi-file into a text file, and then pruned all other elements (pitch, channel, velocity, etc.[3]) from the file. This left us with a sequence of 400 notes, made of 18 distinct notes.

Figure 7 shows that no interesting itemsets were found using high interesting thresholds, but what was

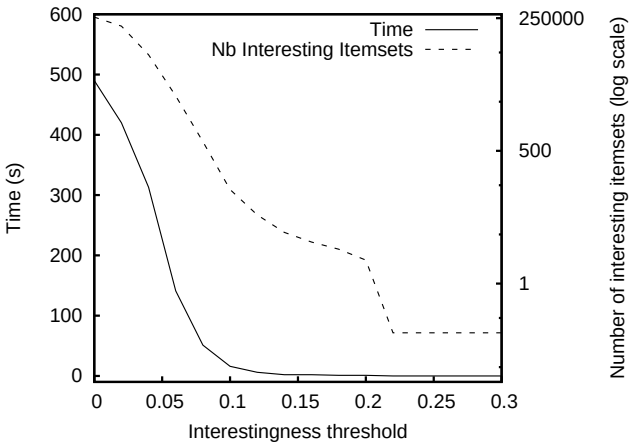


Figure 7: Execution time in seconds and number of interesting itemsets (log-scale) for the MUSIC dataset, with varying *min_int*.

encouraging was the fact that the execution time was minimal. We found the first three interesting itemsets at the level of 0.2: one consisting of four notes, one of five, and the third was a singleton, which was also included in the other two sets. This indicated that this note appeared most often in the sequence, and that the other four notes always appeared relatively close to it (certainly closer than the other 13 notes).

A larger number of itemsets was found to be interesting as we lowered the threshold further, growing consistently even for small decreases in the threshold.

6 Multiple Sequences

Another setting in which discovering interesting patterns can be applied is that of multiple sequences. Here, the data again consists of items, this time coming in many separate sequences. The frequency of an occurrence of a pattern would now be the number of different sequences in which the pattern occurs, regardless of how many times the pattern occurs in any single sequence. In other words, when looking for the frequency of a pattern alone, we can stop looking at a sequence as soon as we have encountered the first occurrence of the pattern.

To determine the interestingness of a pattern, however, it is not enough to know that the items making the pattern occur in a given sequence. We would also like to know how close they appear to each other. Once again, a possible method to do this would be to define a fixed window length as a measure of how close to each other the items need to occur before they can count as an occurrence of the pattern. We could then slide this window over each sequence and find whether the pattern

[2]http://www.tc.umn.edu/~sorem002/hymn_midi.html
[3]http://www.fourmilab.ch/webtools/midicsv/

appears in it or not. By then counting the sequences in which the pattern occurs within such a window, we can say that a pattern is interesting if the number of such sequences is high enough. In other words, a pattern is interesting if its items occur close enough to each other in a high enough number of sequences.

Once again, we would like to do more than that. We would like to give a guarantee not only that the items making up a pattern will appear close to each other in a high number of sequences, but also that if they do all appear in a sequence, then they will appear close to each other. To do this, we will once again define interesting patterns in terms of both frequency and cohesion.

6.1 Definition of the constraint Formally, a single sequence S_j is still defined as in Section 3, and we now denote the set of all sequences by $\mathcal{S}$. We can now redefine coverage, cohesion and interestingness given this new problem setting. This time, coverage will measure in how many sequences the itemset appears, while cohesion will measure how close to each other the items making up the itemset appear on average in their minimal occurrences for a given sequence.

For a given itemset X, we denote the set of all sequences in which all items of X can be found as

$$N(X) = \{j \mid \forall i \in X, \exists (i,t) \in \mathcal{S}_j\}$$

The coverage of X can now be defined as:

$$P(X) = \frac{|N(X)|}{|\mathcal{S}|}$$

In order to calculate the cohesion, we must first evaluate the length of the shortest interval containing the itemset X for each sequence in $N(X)$:

$$W(X,\, j) = \min\{t_2 - t_1 + 1 \mid t_1 \leq t_2 \text{ and } \forall i \in X, \exists (i,t) \in \mathcal{S}_j,\, t_1 \leq t \leq t_2\}$$

We can now compute the average length of such shortest intervals:

$$\overline{W}(X) = \frac{\sum_{j \in N(X)} W(X,\, j)}{|N(X)|}$$

Once again, it is clear that $\overline{W}(X)$ is greater than or equal to the number of items in X. Furthermore, for a fully cohesive itemset, $\overline{W}(X) = |X|$. Therefore, we can again define cohesion of X as

$$C(X) = \frac{|X|}{\overline{W}(X)}$$

We can now define the interestingness of an itemset X as

$$I(X) = C(X)P(X) = \frac{|N(X)| \times |N(X)| \times |X|}{\sum_{j \in N(X)} W(X,\, j) \times |\mathcal{S}|}$$

Given a user defined threshold min_int, an itemset X is considered interesting if

$$I(X) \geq min\_int$$

Again, minimal coverage and minimal cohesion can be used as separate thresholds.

6.2 Computation example To illustrate our interestingness measure, we will now apply it on the sequences given in Figures 1 and 2, this time looked at as two sequences in the same set of sequences.

Once again, we start by looking at itemsets ab, cd and ef. Items a and b can be found next to each other in both sequences, therefore $P(ab) = 1$, $C(ab) = 1$, and, as a result, $I(ab) = 1$. Exactly the same is true for ef, as this time the number of occurrences in any given sequence is of no importance. c and d, meanwhile, appear next to each other in the first sequence, but the minimal interval containing both c and d in the second sequence, $W(cd, 2)$, is of length 3. Therefore,$P(cd) = 1$, $\overline{W}(cd) = 2.5$, $C(cd) = 0.8$, and $I(cd) = 0.8$.

In other words, ab and ef are in this context equally interesting, while cd is slightly less interesting, as there exists a sequence in which both c and d appear, but never next to each other.

Let us now consider itemset eg. This itemset appears in both sequences, so $P(eg) = 1$. Further, we find that $W(eg, 1) = 3$, $W(eg, 2) = 3$, and therefore $\overline{W}(eg) = 3$. As a result, $C(eg) = 0.67$, and $I(eg) = 0.67$.

It may be worth noting that a sliding window method would either conclude that ab and eg are equally interesting (if the window size was greater than 2) or that eg is not interesting at all (if the window size was 2).

Finally, let's take a look at two more itemsets, dh and ck. Items d and h appear in both sequences, but never near each other, while items c and k appear together only in one sequence, but then next to each other. Intuitively, itemset ck seems to be more interesting. The results are given in Table 3.

	$P(X)$	$W(X,1)$	$W(X,2)$	$\overline{W}(X)$	$C(X)$	$I(X)$
dh	1	8	11	9.5	0.21	0.21
ck	0.5	-	2	2	1	0.5

Table 3: Computation of interestingness for itemsets dh and ck

Once again, all these results confirm the intuitiveness of our method.

6.3 Algorithm Sketch We use the same divide-and-conquer algorithm presented in Section 4 to generate candidates, but now the pruning technique is different. Starting from a given candidate $\langle X, Y\rangle$, for each Z such that $X \subseteq Z \subseteq X \cup Y$ the following inequalities stand:

$$\begin{aligned} |N(Z)| &\leq |N(X)| \\ |Z| &\leq |X \cup Y| \\ \sum_{j \in N(X)} W(X, j) &\leq \sum_{j \in N(Z)} W(Z, j) \end{aligned}$$

It therefore follows that

$$I(Z) \leq \frac{|N(X)| \times |N(X)| \times |X \cup Y|}{\sum_{j \in N(X)} W(X, j) \times |\mathcal{S}|}$$

Because all such Z are generated by recursive calls of DFS starting from candidate $< X, Y >$, we can safely prune all itemsets Z that satisfy $X \subseteq Z \subseteq X \cup Y$ if

$$UBI(\langle X, Y\rangle) \equiv \frac{|N(X)|^2 \times |X \cup Y|}{\sum_{j \in N(X)} W(X, j) \times |\mathcal{S}|} < min\_int$$

by suppressing further recursive calls.

Note that the first inequality is different from its counterpart in Section 4, as an occurrence of an itemset X in a sequence directly implies an occurrence of all its subsets in that same sequence, and each sequence appears only once in $N(X)$ regardless of how many times items making up the itemset appear in it. The coverage of a subset can thus never be smaller than the coverage of the set itself. As a result, if minimal coverage is given as a parameter (min_cov), we can achieve further pruning using

$$P(Z) \leq P(X)$$

meaning we can prune Z if

$$P(X) < min\_cov$$

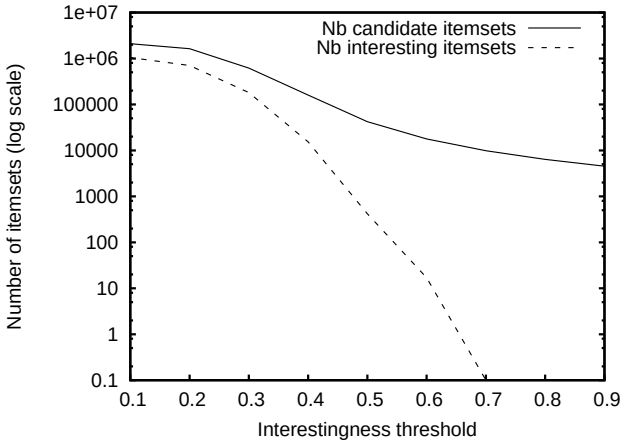
If minimal cohesion is given as a parameter (min_coh), we can achieve even more pruning. Using the above inequalities, we find that

$$C(Z) \leq \frac{|N(X)| \times |X \cup Y|}{\sum_{t \in N(X)} W(X, t)}$$

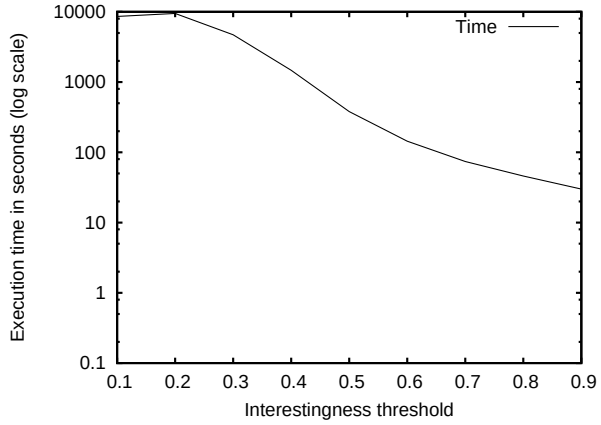
We can therefore prune Z if

$$\frac{|N(X)| \times |X \cup Y|}{\sum_{t \in N(X)} W(X, t)} < min\_coh$$

6.4 Experiments We ran our algorithm on a dataset consisting of 4343 protein sequences of the E. Coli bacteria, containing 22 different items. The most interesting itemsets turned out to be singletons, which was to be expected, since singletons always have cohesion equal to 1, and the coverage of a singleton can never be smaller than the coverage of any of its supersets. After the singletons, the most interesting itemsets were sets of two items consisting of the most interesting singletons, which was not surprising, as the items were all well spread within the sequences, increasing the likelihood that two items would appear close to each other in those sequences in which they both appeared at least once.



(a) number of candidates and found itemsets



(b) execution time

Figure 8: (a) Number of generated candidates and interesting itemsets of size 4 or higher (log-scale) for the E. Coli dataset with varying min_int. (b) Execution time in seconds (log-scale) for the E. Coli dataset, with varying min_int.

To obtain more meaningful results, we decided to look only for itemsets of size 4 or higher. As we can see in Figure 8(a), the number of discovered interesting itemsets still rises sharply as the interestingness threshold declines. The most interesting itemsets were still combinations of the most interesting singletons. Figure 8(b) shows that the execution time for this dataset was proportional to the number of generated candidate itemsets.

Our second dataset consisted of 56 abstracts of papers accepted at the 2008 ECML PKDD conference. We preprocessed these abstracts to obtain a stemmed version and remove all the stop words (such as articles or prepositions) using the Porter stemming algorithm[4]. This resulted in 56 sequences containing 1111 different items (stems). To further speed up our algorithm, we substituted all stems that appeared fewer than 6 times with a dummy. This left us with 56 sequences containing 200 different items. When running the algorithm, we disregarded any candidates containing the dummy.

The results were very satisfactory. As expected, the stems that appeared in most abstracts had the highest interestingness value, followed by some sets of two or more stems. The stems that appeared in most abstracts were *use*, *show* and *method*, yet the highest ranked set of two stems was {*real*, *world*} (see Table 4 and Table 5 for details and further examples). The stem *use* not only appears in more abstracts than *world*, but it appears more often in the same abstract with *real* than *world* does, and yet the set {*real*, *world*} was recognised as much more interesting than the set {*use*, *real*}. In fact, the set {*use*, *real*} is so uninteresting that we did not find it even when we lowered the interestingness threshold to 0.03. This shows that our algorithm recognises that when both *real* and *world* appear in an abstract, they tend to appear near each other, which cannot be said for *use* and *real*.

X	$\|N(X)\|$	$P(X)$	$C(X)$	$I(X)$
use	31	0.55	1	0.55
show	31	0.55	1	0.55
method	31	0.55	1	0.55
algorithm	30	0.54	1	0.54
paper	30	0.54	1	0.54
propos	30	0.54	1	0.54
base	30	0.54	1	0.54
data	29	0.52	1	0.52

Table 4: The most interesting stems identified in the abstracts of papers accepted at the 2008 ECML PKDD conference

[4]http://tartarus.org/∼martin/PorterStemmer/index.html

X	$\|N(X)\|$	$P(X)$	$C(X)$	$I(X)$
$\{real, world\}$	9	0.16	1	0.16
$\{dimension, reduct\}$	6	0.11	1	0.11
$\{semi, supervis\}$	6	0.11	1	0.11
$\{state, art\}$	6	0.11	1	0.11
$\{experiment, result\}$	6	0.11	1	0.11
$\{learn, task\}$	7	0.13	0.78	0.1
$\{dimension, low\}$	5	0.09	1	0.09

Table 5: The most interesting pairs of stems identified in the abstracts of papers accepted at the 2008 ECML PKDD conference

X	$\|N(X)\|$	$P(X)$	$C(X)$	$I(X)$
$\{algorithm, show\}$	17	0.3	0.19	0.06
$\{use, algorithm\}$	16	0.29	0.14	0.04
$\{propos, method\}$	20	0.36	0.11	0.04
$\{use, method\}$	18	0.32	0.12	0.04
$\{method, base\}$	15	0.27	0.13	0.04
$\{algorithm, data\}$	16	0.29	0.12	0.03

Table 6: Some examples of uninteresting pairs of stems in the abstracts of papers accepted at the 2008 ECML PKDD conference

Table 5 also shows that while stems *reduct* and *low* do not appear very frequently, when they do appear they appear right next to stem *dimension*, making the sets {*dimension*, *reduct*} and {*dimension*, *low*} interesting. Table 6, meanwhile, shows that our algorithm recognises that while some stems frequently appear in the same abstract, they are not interesting unless they regularly appear close to each other. We can safely conclude that our algorithm correctly identifies interesting itemsets and does not output any spurious results.

7 Conclusion and future work

In this paper we presented a new constraint to identify interesting itemsets in one or many event sequences. For one sequence, such itemsets consist of frequent items that on average appear close to each other. Unlike previous approaches that only look for items that appear close to each other frequently enough, our constraint gives a guarantee that when an item from an interesting itemset is encountered, the remainder of the set will be found nearby.

We also provide a sound and complete algorithm to extract itemsets having an interestingness value above a user given threshold. This algorithm takes advantage of certain properties of the defined interestingness that allow us to efficiently prune large numbers of candidates.

We ran experiments on both synthetic and real-life datasets and the results proved the intuitiveness of our measure. The synthetically generated Markov chain datasets confirmed that our algorithm detects the expected interesting itemsets and, unless the threshold is set too low, does not output any spurious itemsets. The random dataset confirmed that there is no spurious output. The experiments made on real-life datasets provided further proof of the efficiency of our pruning technique.

We extended our definitions to a situation where the dataset consists of multiple sequences. Here, an itemset is deemed interesting if its items appear in many sequences close to each other. We developed another algorithm that efficiently looks for such sets in multiple sequences. Experiments demonstrated the method's efficiency and intuitiveness.

In future work, we plan to examine the possibilities of using separate coverage and cohesion thresholds differently, possibly in combination with a minimal set size threshold, in an attempt to improve both the definition of interestingness and the efficiency of the algorithm. We further plan to extend our research into ordered patterns, or serial episodes, as they are often referred to.

References

[1] R. Agrawal and R. Srikant, *Mining Sequential Patterns*, Proc. 11th Int. Conf. on Data Engineering, Washington DC: IEEE Comput. Soc., 1995

[2] J. Besson, C. Robardet, J.-F. Boulicaut and S. Rome, *Constraint-based concept mining and its application to microarray data analysis*, IDA, volume 9(1), pp. 59-82, IOS Press, 2005

[3] L. Cerf, J. Besson, C. Robardet and J.-F. Boulicaut, *Data-Peeler: Constraint-based Closed Pattern Mining in n-ary Relations*, Proc. SIAM Int. Conference on Data Mining (SDM), pp. 37-48, 2008

[4] G. C. Garriga, *Discovering unbounded episodes in sequential data*, Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-03), pp. 83–94, 2003

[5] A. Gély, *A generic algorithm for generating closed sets of a binary relation*, In Proc. of 3rd Int. Conf. on Formal Concept Analysis (ICFCA), volume 3403, pp. 223-234, Springer, 2005

[6] H. Mannila and H. Toivonen, *Discovering Generalized Episodes Using Minimal Occurrences*, Proc. Int. Conference on Knowledge Discovery and Data Mining (KDD), pp. 146-151, 1996

[7] H. Mannila, H. Toivonen and A. I. Verkamo, *Discovering frequent episodes in sequences*, International Conference on Knowledge Discovery and Data Mining (KDD), pp. 210-215, 1995

[8] H. Mannila, H. Toivonen and A. I. Verkamo, *Discovery of Frequent Episodes in Event Sequences*, Data Mining and Knowledge Discovery, volume 1(3), pp. 259–289, 1997

[9] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha and K. Zhang, *Combinatorial pattern discovery for scientific data: some preliminary results*, Proc. 1994 ACM SIGMOD Int. Conf. on Management of Data, pp. 115-125, Minneapolis, Minnesota, 1994

[10] G. Wu, *Frequency and markov chain analysis of the amino-acid sequence of human alcohol dehydrogenase alpha-chain*, Alcohol and Alcoholism, volume 35(3), pp. 302-306, 2000