Finding Top-n Emerging Sequences to Contrast Sequence Sets

Osmar R. Zaïane
Department of Computing Science
University of Alberta, Canada
zaiane@cs.ualberta.ca

Kalina Yacef
School of Information Technologies
University of Sydney, Australia
kalina@it.usyd.edu.au

Judy Kay
School of Information Technologies
University of Sydney, Australia
judy@it.usyd.edu.au

Abstract

Comparing groups or sets is the main focal issue in statistics, and data mining research has also focused on automatically identifying values and instances that differ significantly across groups, known as contrast sets. Whether traditional statistics or the work on contrast sets, the comparison is made on nominal data. There is very little work on contrasting sets of event sequences. In this paper we introduce the notion of emerging sequences; sequences that when taken from a set of sequences A and put in a set of sequences B would be considered an abnormal outcast in B and thus distinguishes the set A from the set B. We present approaches for finding such emerging sequences efficiently and introduce an algorithm for discovering the top most emerging sequences.

1. Introduction

In many applications comparing groups is paramount and the fundamental goal of data analysis is to understand the differences and what makes these differences between contrasting groups. Applications such as comparing profitable and unprofitable transactions in business, comparing good customers from churning ones for customer relationship management, comparing populations where some drugs are effective or ineffective in medicine, comparing river floods in weather forecast, or comparing crop harvest from year to year, are typical examples where understanding differences is crucial and has certainly been studied for many years. While there has been significant work from statistics as well as data mining to study what contrasts groups, the comparison between groups is typically based on discrete or numerical attributes. Contrast Sets [3] (also called Emerging Patterns [6, 26]), for instance, are conjunctions of attribute-value pairs that are significantly more frequent in one group than another. In other words, the discovered contrast between groups is based on these attribute-value pairs. When instances in groups are not described by numerical or discrete attributes but rather by sequences of events, there are no known systematic ways to better help understand differences between these sequence-based groups. Examples of groups of sequences are abundant: contrasting the sequences of clicks performed by e-commerce site's visitors who purchase products against the sequences of clicks of visitors who leave the site abandoning their cart before checking out could give insight to marketing people and web designers to better entice buyers. Distinguishing the sequences of events performed by online students who succeed in their learning activities vis-à-vis the sequences of events performed by the less successful learners could provide insight to educators in order to improve or follow the learning activities as well as provide discriminating patterns for automatic recommendation systems. Comparing sequences of amino-acids in proteins of different classes could provide the sequences with the strongest discriminative power to determine protein location in cells or protein functions. The same can be said about the gene G-T-C-A sequences in bioinformatics as well as any application with sequence data. Our work for discovering emerging sequences has potential relevance in detecting differences in groups of event sequences in several application domains.

Understanding the contrast between groups can provide essential patterns which, when incorporated in a classifier, lead to high accuracy and predictive power. This has been demonstrated for the traditional contrast sets and emerging patterns [13, 14] and we believe there is the same potential for emerging sequences.

This paper presents two major contributions:

1. We introduce the notion of emerging sequences: A sequence from a group A of sequences that, whether existing or not in a group B, if added in the group B would be considered an outlier sequence and thus distinguishes A from B, is an emerging sequence.

2. We introduce two effective and efficient algorithms to find the most emerging sequence when contrasting one group with another based on a measure of distance. The algorithms are extended to find the top n emerging sequences.

The remainder of the paper is organized as follows: we present in the next section a brief survey of related work concerning contrasting groups. Section 3 introduces contrast sequences and a naïve approach to find them using a brute-force nested loop. Section 4 describes two algorithms that use simple but clever heuristics to prune the search space and significantly speedup the emerging sequence discovery process. The algorithm for finding the top n emerging sequences is also introduced in Section 4. Some experimental results are presented in Section 5 illustrating the order of magnitude speedup over the naïve approach and the scalability of our approaches. We suggest future work and present our conclusion in Section 6.

2. Related Work on Contrasting Groups

In statistics, myriad tests, such as the t-test and the chi-square test, have been developed to examine whether two different samples are dissimilar enough in some characteristics and whether the difference is statistically significant. Many elaborate methods were also contributed by the statistics community. For instance, the method of kernel discriminant analysis estimates the class conditional distributions using kernel density estimation to learn a discriminant function for each class [9] and hence emphasizes differences between groups. Comparing groups is the art of statistics and surveying the different methods is out of the scope of this paper. It suffices to mention that statistical methods are based on sampling, they assume some hypothesis and are solely considering nominal data (i.e. numerical and minimally categorical).

There are also many works in data mining that are related to the study of differences. Discriminant rules [8] are based on Attribute-Oriented Induction [5] and provide patterns representing characteristics from two classes with high support differences. The attribute-oriented induction allows the counting of attribute-value pairs' occurrences at different levels of a given concept hierarchy.

The idea of contrast sets [3] was introduced at the same time and venue as emerging patterns [6]. They are essentially the same concept. Contrast sets or emerging patterns find rules that differentiate contrasting groups. The process consist of discovering attribute-value pairs or conjunctions of attribute-value pairs whose support (i.e. frequency) differs meaningfully across groups. A *significant* contrast set cs is one such $\exists\, i, j$ such that $support_{G_i}(cs) \neq support_{G_j}(cs)$ and a *large* contrast set cs is one such $max_{ij} \mid support_{G_i}(cs) - support_{G_j}(cs) \mid \geq \delta$, while a *jumping* contrast set cs is one such $\exists\, i, j$ such that $support_{G_i}(cs) = 0 \wedge support_{G_j}(cs) \neq 0$. Based on the above, it is clear that contrast sets (and emerging patterns) are tightly linked to association rules and the discovery of frequent itemsets. The most referenced work for discovering such patterns is STUCCO [3] which is based on and uses the same search strategy as Max-Miner [4] for finding maximal frequent patterns. The main difference between contrast sets and emerging patterns is that [6] introduces a *growthRate* to prune the patterns where $GrowthRate = \frac{support_{G_i}(cs)}{support_{G_j}(cs)}$ (0 if $support_{G_i}(cs) = 0$ and ∞ $if support_{G_j}(cs) = 0$) while STUCCO uses the Chi-square test to check the significance of the support differences [3]. Other variations of contrast sets with regard to statistical tests to prune the search space were also recently presented [10]. However, fundamentally they are simply frequent itemsets comprised of conjunctions of attribute-value pairs which are at least frequent in one group and the difference in their frequencies across groups is meaningful. While it is argued that contrast set mining is a new special purpose data mining task, it is demonstrated in [23] that in reality they are a special case of a general rule generation task and that they can straightforwardly be generated by mining for frequent itemsets in separate datasets (i.e. groups). The issue, however, remains the setting of a low support threshold to find those meaningful differences for rarely occurring contrast sets.

While [23] refers to applications of contrast sets in retail data, very few real world applications have been mentioned in the literature. One such example is presented in [15] where contrast sets are used to describe interesting characteristics of different segments of students on a web-based educational system. Their algorithm MCR identifies attributes characterizing patterns of performance disparity between various groups of students even with very low support. Despite the numerous works, it remains that contrast sets (and emerging patterns) are conjunctive rules characterizing only nominal attributes. Groups described by more complex data such as sequences, graphs, etc. cannot easily be contrasted using the above methods.

Recently, the focus has been the speedup of the emerging pattern discovery process [2] but also the extension to more complex data such as subgraphs [21] and sequences [11]. However, the work on sequences aiming at discovering *distinguishing subsequences* is mainly based on frequencies as contrast sets for nominal data [11], which is conceptually different from the notion of *emerging sequence* we introduce herein. To discover distinguishing subsequences, the algorithm presented in [11] assumes the subsequence exists in both groups and uses constraints to prune the search space while counting frequencies of occurrences in groups. Our approach to contrast groups of sequences does not count frequencies and does not assume co-existence of a sequence in both groups.

Another related work worth mentioning is the use of Hidden Markov Models (HMM) to represent sequences of events. For example, in [19] event sequences, in a collaborative online learning situation, are modeled using HMMs to assess how new concepts introduced in a group are assimilated. HMMs were used to classify instances of effective and ineffective knowledge sharing interaction. However, HMMs have not been used to contrast sets of sequences.

3. Emerging Sequences

To introduce the concept of emerging sequences that contrast a sequence group from another we will first start with illustrative examples.

Example 1: In an e-commerce web site selling products, the aim is to entice new buyers, keep buyers from churning and encourage more purchases. It is thus important to understand what makes buyers buy and non-buyers leave empty-handed. The click-stream visitation produces transactions of events chronologically ordered representing the clicks and the actions made by web visitors. Separating the buyers from the non-buyers is trivial and leads to two groups of event sequences. There is a plethora of work and solutions in web mining [16] studying web navigational behaviour to produce recommender systems, personalize web sites, generate analysis, etc. However, comparing groups at the sequence of event level has not been done. A relevant question could be “what is the sequence of events that may more likely lead to a purchase?” or “what is the sequence of events that may more likely delay or hinder a purchase?” Notice that these two questions are symmetric: one pertains to the purchasing group and the other to the non-purchasing group. The sequence that emerges from a group A to contrast the group from another group B is not necessarily the same sequence that may emerge from B to contrast it from A. Answering such questions and finding the sequence or sequences germane to the purchasing group or non-purchasing group is a valuable insight.

Example 2: In a controlled trial of a new drug after its initial production it is common to notice that there are significant side-effects for some individuals while others respond positively to the new drug. Are there genetic predispositions? Contrasting the genetic makeup of the two populations can allow the discovery of genetic signatures for predisposition or reaction to the drug.

Notice that in Example 1 the input consists of two groups materialized into transactions of chronologically ordered events. Each group is a set of transactions of ordered events (i.e. sequences) similar to the transactions for sequence analysis in [20] for market basket analysis. The sequences can be kept in their existing condition or, due to their excessive length, processed to extract frequent subsequences using existing algorithms such as GSP [20], Spade [25], PrefixSpan [17], or SPAM [1]. The result would still be two sets of sequences, all be it larger sets of short and long sequences.

In Example 2 the input is consisting of two very long sequences or two sets of very long sequences. This is common for proteins, genes or even natural language text. These long sequences need to be broken down into frequent subsequences. A frequent subsequence is a subsequence made up of consecutive elements that occurs in more than a certain fraction of the sequences in a group. There are efficient algorithms in bioinformatics to process such long sequences for frequent subsequences [7]. One such algorithm uses generalized suffix trees (GST) [22]. A GST is a trie-like structure designed for compactly representing a set of strings. Each suffix of the string is represented by a leaf in the GST. There are algorithms to build a GST in linear time [7]. Frequent subsequences are extracted by simple traversal of the GST. The end result would be, like in Example 1, two groups of sequences ready to be contrasted.

The application that motivated this work is related to e-learning and the understanding of collaborative team work [12]. In an educational project conducted in the context of a semester long software development course in which teams of students collaborated and interacted on-line to create substantial software artifacts, we collected data about their activities. We wanted to understand the sequences of events (i.e. collaborative behaviour) that lead a team to success and the sequences of events that prevented another team from succeeding. In other words our questions where: “what are the sequences of events that may more likely lead to success in student software creation?” and “what are the sequences of events that may more likely avert successful ending in collaborative software creation?” This is similar to Example 1, except that events expressed in terms of activities such as creating a task, modifying code, sending a message, distributing responsibilities, leading, etc., are articulated in a complex vocabulary taking into account repetitive activities and authors of the events in a team, resulting in a convoluted distance measure to evaluate similarity between sequences. This is out of the scope of this paper. However, distance measures are central to our approach for discovering emerging sequences.

3.1. The Notion of Emerging Sequence

Given two groups of sequences A and B, an emerging sequence contrasting A from B, $EmergeSequence(A, B)$, is a subsequence from A that, regardless of its frequency in A or B, is the furthest away from its closest match in B from any subsequence in A. In other words if $EmergeSequence(A, B) = x$, $x \in A$ and has the largest distance to its closest match in B. More explicitly, if x is added to B, x would be the most outlier in B based on distance. If x already existed in B, it is by definition different and distant from the other sequences in B. The problem thus consists of finding the closest neighbour in B for each subsequence in A and select the one with the largest distance to its closest match. Notice that the problem is not symmetric, that is we likely have $EmergeSequence(A, B) \neq EmergeSequence(B, A)$. This is because $EmergeSequence(A, B)$ may not initially exist in B, and even if it does it is not necessarily the most outlier within A.

Definition 1 (Most Emerging Sequence) *Given two distinct groups of sequences A and B, $EmergeSequence(A, B)$, the most emerging sequence contrasting A from B, is the sequence in A that has the largest distance to its nearest neighbour in B.*

The definition assumes the existence of a similarity or distance measure between sequences. There are many distance measures or metrics used in the literature: Euclidian, Manhattan, Mahalanobis distance, Cosine measure, Jaccard coefficient, etc. In this paper a given distance function $d(x, y)$ is assumed which can either be a standard metric or an application domain specific function defined to return the distance between two sequences. The distance of a sequence in a group from its nearest neighbour in another group provides the possibility of ranking and thus extracting a top list of most emerging sequences.

Definition 2 (Top n Emerging Sequence) *Given two distinct groups of sequences A and B, and a number n, the top n emerging sequences contrasting A from B are the n most emerging sequences from A. The ranking is based on the largest distance to their respective nearest neighbour in B.*

Since an emerging sequence is not symmetric, to contrast two groups of sequences one needs to apply the discovery of emerging sequences symmetrically. In other words find the emerging sequence contrasting A from B then find the emerging sequence contrasting B from A.

Definition 3 (Contrasting Emerging Sequences) *Given two distinct groups of sequences A and B, the contrasting emerging sequences is a pair $(EmergeSequence(A, B), EmergeSequence(B, A))$ of the most emerging sequence contrasting A from B and the most emerging sequence contrasting B from A.*

Given the duality, we will only concentrate on one direction: finding the most emerging sequence contrasting A from B ($EmergeSequence(A, B)$). The other direction is simply the reverse. Notice however that one may be more interested in one direction than the other. For example in the motivating scenario understanding success in online student collaboration, if we want to find success predictors we would contrast the successful group from the non successful. However, if we are more interested in makers for failure, we may wish to explore emerging sequences in that direction (i.e. contrasting the failed group from the other).

3.2. An Approach to Discover Emerging Sequences

We describe here the simple and naïve brute-force algorithm for finding an emerging sequence that contrasts a group of sequences from another.

The intuition behind the algorithm is that the emerging sequence from a group A vis-à-vis a group B would be an outlier sequence in B: an outlier. We need to do a pairwise comparison between all sequences in A and B to compute their distances. For each sequence in A we need to compute its distance to all sequences in B and keep the shortest one: its nearest neighbour. The one that has the furthest nearest neighbour is the outlier sequence and declared the most emerging sequence contrasting A from B. This is a nested loop as illustrated in Algorithm 1 in which for every $x \in A$ and every $y \in B$ the distance $d(x, y)$ is computed.

The algorithms looks for the sequence in A (initialized to NIL in line 2) that has the largest distance to its nearest neighbour in B. The outer loop scans A (line 3) and for each sequence looks for its nearest neighbour (line 7 to 9) in the inner loop scanning B (line 6). In lines 10 to 13, the neighbour with the largest distance is stored $ES.neighbour$ while the emerging sequence is stored in $ES.sequence$. The complexity of this simple naïve nested loop, Algorithm 1, is $\mid A \mid * \mid B \mid$ or N^2 assuming both groups have the same size N. Obviously, this is the worst case scenario and more efficient algorithms are possible.

4. Finding Emerging Sequences Efficiently

Computing the pairwise distance for all sequences in both groups is not necessary. Indeed, what we are looking for is the sequence in A that has the largest distance to its nearest match. Since $LargestDistance$ in line 11 of Algorithm 1 stores the largest distance to a sequence's nearest match so far found, as soon as we find a sequence $t \in B$ that

```
Algorithm NNL
INPUT: Group of Sequences A and B
OUTPUT: Sequence contrasting A with B - i.e.
sequence in A with largest distance to its nearest
neighbour in B
1  LargestDistance ← 0;
2  ES ← NIL;
3  for each $s \in A$ do
4      Nearest ← $\infty$ ;
5      Neighbour=NIL;
6      for each $t \in B$ do
7          if $d(s, t) < Nearest$ then
8              Nearest ← d(s, t);
9              Neighbour ← t;
           endif
       endfor
10     if $Nearest > LargestDistance$ then
11         LargestDistance ← Nearest;
12         ES.sequence ← s;
13         ES.nearestNeighbour ← Neighbour;
       endif
   endfor
14 Return (ES);
```

Algorithm 1: Naïve Nested Loop Algorithm to find an Emerging Sequence

```
Algorithm SNL
INPUT: Group of Sequences A and B
OUTPUT: Sequence contrasting A with B - i.e.
sequence in A with largest distance to its nearest
neighbour in B
1  LargestDistance ← 0;
2  ES ← NIL;
3  for each $s \in A$ do
4      Nearest ← $\infty$ ;
5      Neighbour ← NIL;
6      for each $t \in B$ do
7          if $d(s, t) < LargestDistance$ then
8              Break;
           endif
9          if $d(s, t) < Nearest$ then
10             Nearest ← d(s, t);
11             Neighbour ← t;
           endif
       endfor
12     if $Nearest > LargestDistance$ then
13         LargestDistance ← Nearest;
14         ES.sequence ← s;
15         ES.nearestNeighboutr ← Neighbour;
       endif
   endfor
16 Return (ES);
```

Algorithm 2: Skip Nested Loop Algorithm to find an Emerging Sequence

has a shorter distance to the current sequence $s \in A$ than the current $LargestDistance$, there is no need to pursue with s and any other sequence in B after t. This is because if the distance $d(s,t) < LargestDistance$ then even if t ends up being the nearest match to s it does not have a distance larger than the larger distance found so far and thus there is no need to further investigate which one in B is the real nearest match to the actual s. Based on this observation it suffices to add a test in the inner loop to test whether the distance between the current sequence in A, s, and the current sequence in B, t, is smaller or equal than the largest distance to a neighbour found so far ($LargestDistance$). This test is what is added in Algorithm 2 lines 7 and 8. The "break" stops the inner loop as there is not need to continue scanning B for the current sequence s in the outer loop, hence the name *Skip Nested Loop*.

In the worst case scenario the break in line 8 will never be executed and the distance of all pairs (x, y), such that $x \in A$ and $y \in B$, would be computed resulting in a complexity of $O(N^2)$ assuming the same size N for A and B (Figure 1.left). However, if we are lucky enough, after the first sequence in A is compared against all sequences in B, the real largest distance to a nearest neighbour is found (i.e. the maximum $LargestDistance$ is found). In that case there is no need to compare the remaining sequences in A with all the remaining sequences in B if B is sorted in such a way that the nearest neighbour is always the first in B. Line 7 would always be true. This scenario is illustrated in the right of Figure 1. In that case B is scanned once for the first sequence in A but then the inner loop is always broken at the first opportunity for each sequence in A giving a linear complexity of $O(2N) \approx O(N)$.

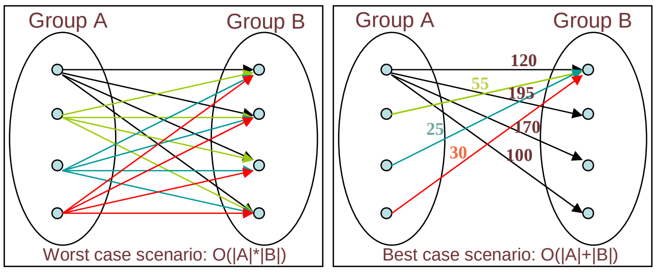
Group A
Group B
Worst case scenario: O(|A|*|B|)
Group A
Group B
120
55
195
25
170
30
100
Best case scenario: O(|A|+|B|)

Figure 1. Worst and best case scenarios for Skip Nested Loop algorithm.

An idealistic token example is illustrated on the right of Figure 1. The first sequence in A is compared to all sequences in B and its nearest neighbour is found to be the last sequence in B with a distance of 100. When entering the second iteration of the outer loop, $LargestDistance$ is 100 and the second sequence in A, s_2, is compared with the first sequence in B, t_1, and their distance is found to be $d(s_2, t_1) = 55$. 55 is smaller than $LargestDistance$ so whether t_1 is the nearest neighbour of s_2 or not, s_2 will never get a nearest neighbour that is more distant than $LargestDistance$. Thus, it is useless to look for the actual nearest neighbour of s_2 and check later if it is the largest. The same applies for the third iteration with s_3 since $d(s_3, t_1) = 25 < 100$ and in the fourth iteration since $d(s_4, t_1) = 30 < 100$.

To end up in this very idealistic scenario is very unlikely however. The order in which the sequences in A are considered is important. We want the first sequence to find a nearest neighbour that ends up the furthest away. Also the order in which sequences in B are considered is important. The earlier we examine a sequence t in B that has a smaller distance $d(s, t)$ than the current $LargestDistance$ the earlier we can stop looking for a suitable nearest neighbour for the current sequence s in A. Both groups A and B need to be sorted relative to the distance measure used and the nearest neighbours. However, sorting would assume knowing the nearest neighbours which defeats the purpose of looking for nearest neighbours in the first place. Nevertheless, some simple heuristics could be used for sorting the sequences. For instance, if the length of sequences is used in the distance function, simple sorting based on sequence length could improve performance. In addition to being more efficient than Algorithm 1 (NNL), Algorithm 2 (SNL) provides the opportunity to inject domain knowledge to further improve efficiency if application domain heuristics exist to sort either or both of the sequence groups.

In the case of random ordering of the sequences a better strategy to improve the nested loop is to not only skip sequences in B but also in A whenever possible and count on randomness. The idea is to read blocks of sequences at a time from A then find the nearest neighbour from B of each sequence in the block and select the largest one. Once $LargestDistance$ of previous blocks is known, when considering a new block from A, one does not need to compute the distance $d(s, t)$ for all sequences s in the block and sequences t in B. Indeed, as soon as we find a distance $d(s, t)$ smaller than $LargestDistance$, s has no chance to get its nearest neighbour larger than $LargestDistance$ since it already has a sequence in B that is closer (i.e. t). Thus, we can safely omit checking distances between s and the remaining sequences in B. We simply drop s from the block and the block shrinks. The Block Nested Loop approach is depicted in Algorithm 3. As in previous algorithms, initially $LargestDistance$ is unknown and is set to 0 (line 1). Also, the most emerging sequence is unknown and is initialized with NIL (line 2). The outer loop reads sequences from A one block at a time (line 3). In each iteration the nearest neighbours for all sequences in the block are searched before moving on the the next block. Once a block is in main memory, the first inner loop scans all sequences in B and computes the distances between the sequences in the block and the sequences in B and stores the nearest neighbour for each (lines 13-14). However, if the distance is shorter than the largest distance to a nearest neighbour found in previous blocks (i.e. $LargestDistance$), the sequence is removed from the block as we can abandon the search for a nearest neighbour to that sequence (lines 10-11). In lines 15 to 19 is another loop to compare the nearest neighbours found for the sequences in the current block searching for the maximum distance.

```
   Algorithm BNL
   INPUT: Group of Sequences A and B
   OUTPUT: Sequence contrasting A with B - i.e.
   sequence in A with largest distance to its nearest
   neighbour in B
1  LargestDistance ← 0;
2  ES ← NIL;
3  while blockA ← getNextBlock(A) do
4      for all s ∈ blockA do
5          s.Nearest ← {};
6          s.DistanceToN ← 0;
       endfor
7      for each t ∈ B do
8          for each s ∈ blockA do
9              Distance ← d(s, t);
10             if Distance ≤ LargestDistance then
11                 Remove s from blockA;
               endif
12             if s.Nearest ← {} or
               Distance < s.DistanceToN then
13                 s.Nearest ← t;
14                 s.DistanceToN ← Distance;
               endif
           endfor
       endfor
15     for each s ∈ blockA do
16         if s.DistanceToN > LargestDistance then
17             LargestDistance ← s.DistanceToN;
18             ES.sequence ← s;
19             ES.nearestNeighbour ← s.Nearest;
           endif
       endfor
   endw
20 Return (ES);
```

Algorithm 3: Block Nested Loop Algorithm to find an Emerging Sequence

In the worst case scenario no sequence is ever removed from the block in line 11. This means that all pairs (x, y), such that $x \in A$ and $y \in B$, would be computed resulting in a complexity of $O(N^2)$ assuming the same size N for A and B (Figure 2.left). So the worst case scenario is the same as the worse case scenario for the *Skip Nested Loop*, which is the complexity of the brute-force *Naïve Nested Loop*. However, in a randomized set it is very likely that distances between some sequences in a block and some sequences in B are shorter than the previously found largest distance in other blocks. In those cases, sequences are removed from some blocks reducing the overall number of pairwise comparisons. In the best case scenario (illustrated in the right of Figure 2), the sequences from A in the first read block are all compared for distance computation with all sequences in B. Once the furthest nearest neighbour is found for the first block, it turns out to be larger than any first comparison in all subsequent blocks. Thus, the complexity for the best case is $O(b|B| + |A|)$ where b is the size of a block. Since b is relatively small, the linear complexity of the best case is $O(2bN) \approx O(N)$. This is the case when the furthest nearest neighbour is found in the first block of A.

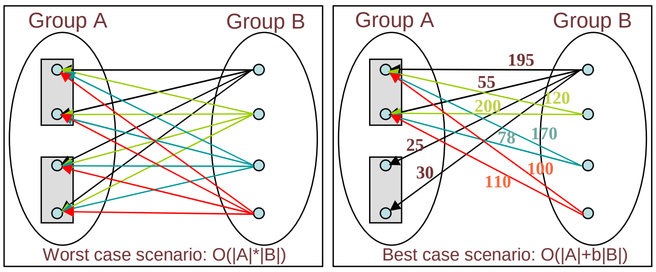
Group A
Group B
Worst case scenario: O(|A|*|B|)
Group A
Group B
195
55
200
120
78
170
25
30
100
110
Best case scenario: O(|A|+b|B|)

Figure 2. Worst and best case scenarios for Block Nested Loop algorithm.

An idealistic token example for the best case of Algorithm 3 is illustrated on the right of Figure 2. Given a block size of 2, the distances for the two sequences in the first block are computed with all sequences in B. The distance to the nearest neighbour of the first sequence in the block is $min(195, 120, 170, 100) = 100$ and the distance to the nearest neighbour of the second sequence in the block is $min(55, 200, 78, 110) = 55$. Thus, the largest distance is $LargestDistance = max(100, 55) = 100$. When the second block is read, as soon as we compute the distance between the first sequence t_1 of B and the first sequence s_1 of the block and obtain 25 we de facto know that we can abandon s_1 from the block. This is because since 25 is smaller than the current $LargestDistance$ 100 and the nearest neighbour to s_1 is at least 25, the nearest neighbour of s_1 (whether t_1 or other) will never be larger than $LargestDistance$ (i.e. the furthest nearest neighbour found so far in previous blocks). This saves computation with the remaining sequences in B. The same applies to the second sequence in the block since the distance $d(s_2, t_1) = 30$ (that is < 100).

```
Algorithm TopES
INPUT: Group of Sequences A and B; n, the number
of top contrasting sequences to return
OUTPUT: Top n most contrasting sequences from A
w.r.t. B - i.e. sequences in A with largest distance to
their nearest neighbour in B
Let minDist(T) return the minimum distance between
elements in T and their respective nearest neighbours
1  T[1..n] ← NIL;
2  while blockA ← getNextBlock(A) do
3      for all s ∈ blockA do
4          s.Nearest ← {};
5          s.DistanceToN ← 0;
       endfor
6      for each t ∈ B do
7          for each s ∈ blockA do
8              Distance ← d(s, t);
9              if Distance ≤ minDist(T) then
10                 Remove s from blockA;
               endif
11             if s.Nearest={} or
               Distance < s.DistanceToN then
12                 s.Nearest ← t;
13                 s.DistanceToN ← Distance;
               endif
           endfor
       endfor
14     for each s ∈ blockA do
15         if s.DistanceToN > minDist(T) then
16             Seq.sequence ← s;
17             Seq.Nearest ← s.Nearest;
18             Seq.distance ← s.DistanceToN;
19             T ← Top(T ∪ Seq, n);
               //keep only the top n sequences based on
               distance
           endif
       endfor
   endw
20 Return (T);
```

Algorithm 4: Top-n Algorithm to find Top-n Emerging Sequences

4.1. Finding Top n Emerging Sequences

The previous algorithms contrast a group of sequences A from a group of sequences B. To contrast B from A, the groups need to be reversed and the same algorithms can be applied again. To highlight differences between two groups, both directions need investigation, hence the pair in Definition 3.

The most emerging sequence is one sequence from a group A that contrasts A from a group B. It is probably not the only sequence that highlights difference, but it has the furthest distance to its nearest neighbour in B. It would be the most outlying sequence if it was inside B. This presumes an ordering in the emerging sequences. The second most emerging sequence is the one that has the second furthest distance to its nearest neighbour in B and can be found after removing the most emerging sequence from A and applying the algorithm again. However, the previous algorithms can easily be modified to find a list of ordered emerging sequences. Algorithm 4 extends Algorithm 3 to find the top n most emerging sequences contrasting A from B.

An array stores the top most emerging sequences found and is initialized to NIL in line 1. Instead of the variable $LargestDistance$ to keep track of the furthest distance to a nearest neighbour, the function $minDist()$ returns the smallest distance among the distances of the emerging sequences so far collected in the array. The strategy reading blocks from A remains the same as in Algorithm 3. The last loop in line 14 checks the nearest neighbours of the sequences of the block. If anyone is larger than those already collected in the array, the new emerging sequence is inserted in the array and the one with the shortest distance to its neighbour is eliminated (line 19). The complexity of this algorithm remains the same as Algorithm 3.

Theoretically, both algorithms SNL and BNL have a complexity between $O(N^2)$ and $O(N)$. However, we would expect their performance is near linear and by far they outperform NNL.

5. Experimental Evaluation

To show the efficiency of our algorithms we compared NNL, SNL and BNL on a collection of datasets. Evaluation on real datasets is not reported as the interpretation of the results by the domain experts is still underway. However, the efficiency and scalability results are similar to the results obtained from synthetic data. We generated sets of sequences of the same size in the number of sequences and compared the execution time of the three algorithms as well as the number of pairwise distance computations processed by the three contenders on each dataset. The distances between sequences were randomly generated so that the distance function had simply to consult a matrix for distance calculation. The distance matrix was loaded in main memory. In addition, we evaluated the effect of the size of the block in BNL.

Experiments were executed on a Pentium IV 3.00 GHz machine with 1 GB memory running Windows XP Professional and the algorithms were implemented in C.

In all experiments both sets of sequences had the same size in number of sequences. Sequences are not of the same length so files are not of the same size in bytes. Figure 3 illustrates the scalability of our algorithms.

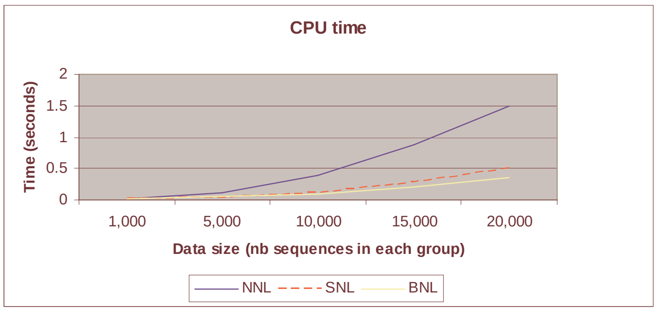

Figure 3. Scalability Comparison between algorithms.

When both groups are of a small size all three algorithms perform similarly. However, as the sizes increase, both SNL and BNL clearly outperform NNL. With 20,000 sequences in each group, NNL needs three times more time to find the emerging sequence than SNL or BNL. BNL performs slightly better than SNL and its efficiency is even more pronounced as the size of the groups increases. Further randomization of the sets can improve BNL's performance even more. On the plant protein dataset the difference between SNL and BNL was more pronounced.

The times reported in Figure 3 do not include the run time for computing distances between sequences as in our experiments the distance function simply accessed a pre-computed distance matrix. Some distance functions can be expensive and thus reducing the number of calls to such a function is critical. In Figure 4 we report the number of pairwise distance computations, which the heuristics in SNL and BNL attempt to reduce. We use a logarithmic scale as the differences are noteworthy. Clearly, NNL computes N^2 distance computations, N being the size of each of the two groups. SNL needs significantly less distance computations and BNL even less. With an expensive distance function, this difference in the number of pairwise computations would translate into major time savings with BNL when mining large sets.

BNL reads blocks of sequences at a time from the first group of sequences A. All sequences of the first block are compared with all sequences in the second group B. This means that in the extreme case when the block size is equivalent to the whole group A we would have as many distance computations as with NNL (i.e. N^2). Having a very small block size, for instance 1, is also not advantageous since

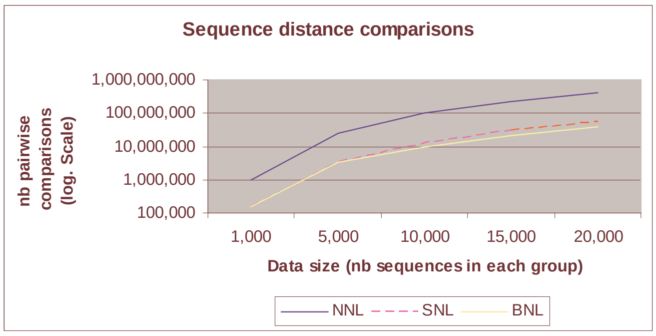

Figure 4. Number of Pair-wise sequence comparison for each algorithm.

it would assume that the emerging sequence is early in the group A and it would not take advantage of the randomness of the data. A relatively small block size, however, is still beneficial and certainly more valuable than a large one due to the complete crossover with the first block and the second group of sequences. We performed tests by varying the block size from 0.2% to 6% of the size of the group and did not notice significant difference. As the size increase beyond 10 and 15% the performance drops.

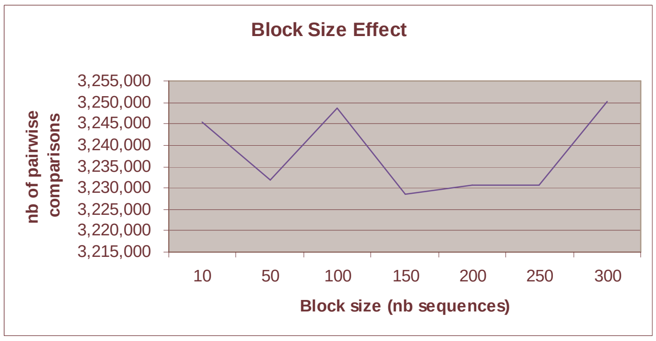

Figure 5. Effect of block size in BNL on pair-wise comparisons

Figure 5 depicts the number of distance computations for two sequence sets of size 5,000 and varying the block size from 10 to 300 sequences. The performance is relatively stable as the fluctuation of ± 17,000 out of a potential of 25 million pairwise distance computations is negligible. We advocate a block size of maximum 10% of the size of the first groups of sequences and preferably 5%.

Testing on real datasets is very important for validation of effectiveness. Initial tests on plant protein data used to identify extracellular and intracellular proteins [24] show promising results but remain to be confirmed by biologists. Significance of the discovered emerging sequences can only be assessed by domain experts. Another validation we are experimenting is the inclusion of the discovered emerging sequences in a classification model. If the accuracy of the classifier is improved using the emerging sequences then these discovered emerging sequences have a critical discriminant power and are hence relevant discoveries.

As for the motivating e-learning project mentioned previously in which we intend to compare successful and less successful teams in a collaborative software development undertaking, the hindering issue is the validation of a good distance function between event sequences. Indeed the vocabulary describing these sequences is very rich, not just in terms of different possible events but also in terms of successive repetitions of events and event authors, creating a complex semantics [12]. Since the distance function is central in our algorithms, it is critical to initially confirm a semantically correct and effective distance function in our application before contemplating the interpretation of emerging sequences discovered. Nevertheless, preliminary results are very promising in terms of effectiveness. We will use domain theories to help inform design of the distance function, for example theories of group operation [18].

6. Conclusion

We introduced the problem of finding emerging sequences when contrasting groups of sequences. An emerging sequence captures what makes a group of sequences different from another. In essence, given two sequence sets A and B, an emerging sequence contrasting A from B is a sequence from A that is the most different from all sequences in B and would be considered an outlier if in B.

We presented and studied two efficient algorithms to discover emerging sequences. SNL uses a simple trick to filter out doomed candidates and provide the possibility to exploit domain specific heuristics to sort the sequence groups for better performance. BNL reads blocks of sequences from the outer loop group and prunes on both sequence groups to reduce pairwise distance computation. We showed the efficiency of our algorithms in finding emerging sequences.

BNL performs best when the groups of sequences are randomized. It is worthwhile investigating the possibility to add a pre-processing phase in which the sequence groups are randomized to better take advantage of BNL's pruning.

Another valuable study is to compare the top n emerging sequences discovered with our methods with the support-based distinguishing subsequences. Distinguishing subsequences appear more frequently in one group than another. Our emerging sequences do not depend on frequencies. However, an interesting question is to investigate the relationship between the two patterns.

As mentioned above, emerging sequences could have good discriminating power between groups. When discovered between classes in a learning phase of a classification problem for sequence data, a more accurate classifier could be modeled. Further research in this direction could be constructive.

References

[1] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 429–435, Edmonton, Canada, July 2002.

[2] J. Bailey, T. Manoukian, and K. Ramamohanarao. Fast algorithms for mining emerging patterns. In *European Conference of Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 39–50, Helsinki, Finland, August 2002.

[3] S. D. Bay and M. J. Pazzani. Detecting change in categorical data: Mining contrast sets. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 302–306, San Diego, USA, August 1999.

[4] R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM-SIGMOD International Conference on Management of Data*, pages 85–93, Seatle, WA, USA, June 1998.

[5] Y. Cai, N. Cercone, and J. Han. *Knowledge Discovery in Databases*, chapter Attribute-oriented induction in relational databases, pages 213–228. AAAI/MIT Press, Cambridge, MA, USA, 1991.

[6] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 43–52, San Diego, USA, August 1999.

[7] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

[8] J. Han, Y. Fu, K. Koperski, W. Wang, and O. R. Zaïane. Dmql: A data mining query language for relational databases. In *SIGMOD Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, pages 27–33, Montreal, Canada, June 1996.

[9] D. J. Hand. *Kernel Discriminant Analysis*. Research Studies Press/Wiley, 1982.

[10] R. Hilderman and T. Peckham. A statistically sound alternative approach to mining contrast sets. In *Australian Data Mining Conference*, pages 157–172, Sydney, Australia, December 2005.

[11] X. Ji, J. Bailey, and G. Dong. Mining minimal distinguishing subsequence patterns with gap constraints. In *IEEE International Conference on Data Mining (ICDM)*, pages 194–201. Houston, TX, USA, November 2005.

[12] J. Kay, N. Maisonneuve, K. Yacef, and O. R. Zaïane. Mining patterns of events in students' teamwork data. In *Educational Data Mining Workshop, held in conjunction with Intelligent Tutoring Systems (ITS)*, Taiwan, June 2006.

[13] J. Li, G. Dong, and K. Ramamohanarao. Instance-based classification by emerging patterns. In *European Conference of Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 191–200, Lyon, France, September 2000.

[14] J. Li, G. Dong, and K. Ramamohanarao. Making use of the most expressive jumping emerging patterns for classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 220–232, Kyoto, Japan, April 2000.

[15] B. Minaei-Bidgoli, P.-N. Tan, and W. F. Punch. Mining interesting contrast rules for a web-based educational system. In *International Conference on Machine Learning and Applications (ICMLA)*. Louisville, KY, USA, December 2004.

[16] O. Nasraoui, O. R. Zaïane, M. Spiliopoulou, B. Mobasher, P. Yu, and B. Masand, editors. *ACM SIGKDD Workshop on Knowledge Discovery in the Web (WebKDD): Taming Evolving, Expanding and Multi-faceted Web Clickstreams*, Chicago, IL, USA, August 2005.

[17] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *IEEE International Conference on Data Engineering (ICDE)*, pages 215–224, Heidelberg, Germany, April 2001.

[18] E. Salas, D. Sims, and C. Burke. Is there a "big five" in teamwork? *Small Group Research*, 36(5):555–599, 2005.

[19] A. Soller, J. Wiebe, and A. Lesgold. A machine learning approach to assessing knowledge sharing during collaborative learning activities. In *Computer Support for Collaborative Learning*, pages 128–137, Boulder, Colorado, USA, January 2002.

[20] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology (EDBT)*, pages 3–17, Avignon, France, Mars 1996.

[21] R. M. H. Ting and J. Bailey. Mining minimal contrast subgraph patterns. In *SIAM Data Mining Conference*, pages 638–642, Bethesda, MD, USA, April 2006.

[22] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. A. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *ACM SIGMOD International Conference on Management of Data*, pages 115–125, Minneapolis, USA, June 1994.

[23] G. I. Webb, S. M. Butler, and D. A. Newlands. On detecting differences between groups. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 256–265, Washington, DC, USA, August 2003.

[24] O. R. Zaïane, Y. Wang, R. Goebel, and G. Taylor. Frequent subsequence-based protein localization. In *Workshop on Data Mining for Biomedical Applications (BioDM'06), in Conjunction with PAKDD'06*, Lecture Notes in Bioinformatics Volume 3916, Springer Verlag, pages 35–47, Singapore, April 2006.

[25] M. J. Zaki. Efficient enumeration of frequent sequences. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 68–75, Bethesda, Maryland, USA, November 1998.

[26] X. Zhang, G. Dong, and K. Ramamohanarao. Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 310–314, Boston, USA, August 2000.