Query Driven Sequence Pattern Mining

Pedro G. Ferreira[1] **and Paulo J. Azevedo**[1]

[1]Department of Informatics
University of Minho
Campus of Gualtar, 4710-057 Braga, Portugal

{pedrogabriel,pja}@di.uminho.pt

Abstract. *The discovery of frequent patterns present in biological sequences has a large number of applications, ranging from classification, clustering and understanding sequence structure and function. This paper presents an algorithm that discovers frequent sequence patterns (motifs) present in a query sequence in respect to a database of sequences. The query is used to guide the mining process and thus only the patterns present in the query are reported. Two main types of patterns can be identified: flexible and rigid gap patterns. The user can choose to report all or only maximal patterns. Constraints and Substitution Sets are pushed directly into the mining process. Experimental evaluation shows the efficiency of the algorithm, the usefulness and the relevance of the extracted patterns.*

1. Introduction

Due to the exponential growth of newly discovered biological sequences (DNA, proteins and other types of fragments), witnessed in the last decades, the subject of sequence analysis plays a central role in bioinformatics. In this context, one of the most important tasks is the discovery of sequence patterns also called motifs. Different approaches and several methods have been proposed to tackle this problem. Earlier type of proposals consisted in a multiple alignment of the evaluated sequences [Hirosawa et al. 1995], with a subsequent report of a consensus sequence with the regions of detected similarity. Due to the inherent high computational costs this approach is only suitable for small sets of closely related sequences. Another possible solution is based on sequence pattern enumeration, where candidate patterns are obtained from combination of the different events present in the input sequences. Enumeration methods can be distinguished by the way they traverse the solution space and by the order that they enumerate the patterns. The Data Mining community has a large body of work on the related task of discovering event-set sequence patterns from transactional datasets. The proposed algorithms are best suited for datasets with many (typically millions) sequences with a relative small length (from 10 to 20), and an alphabet of thousands of events, e.g. [Srikant and Agrawal 1996, Zaki 2000, Jian Pei 2002, Ayres et al. 2002]. In these algorithms, several types of constraints are included and studied. Considering the characteristics inherent to biological sequence databases, namely protein databases (hundreds of

sequences with big average length, usually greater than 100), alternative methods were proposed by the bioinformatics community. An example of such method os the Teiresias [Rigoutsos and A.Floratos 1998] algorithm. When a particular (query) sequence is given for investigation, for instance a recently discovered protein, the detection of sequence patterns may provide valuable insights about the proteins. A possible analysis solution is to scan the query sequence against a database of signature patterns like PROSITE [Bairoch 1991] or Pfam [A. Bateman 2003], trying to identify the respective sequence family. A signature pattern is a pattern that ideally matches all the sequences in a certain family and no other sequences. These patterns can express important functional properties related to the family. Several tools like interProScan [EMBL-EBI] or eMotif [Wu et al.] can be used. These tools essentially differ in the type of patterns they search and in the way they output patterns. The signature patterns can be obtained through manual inspection of the sequences or through automatic programs like Pratt [Jonassen et al. 1995]. Pratt searches for conserved patterns in a set of related protein sequences reporting a a small number of patterns with the highest quality. Unfortunately, Pratt is not suited for finding conserved patterns in small subsets of a database.

Bearing in mind this last problem and the importance of pattern analysis in a large number of biological problems, we present a method that reports all the frequent patterns occurring in a query sequence with respect to an user defined database. The query sequence is used to drive the mining process ensuring containment of the reported patterns. The difference to the previous approaches is that patterns need not to be known in advance as is the case of PROSITE or Pfam and need not to be signature patterns as the ones outputted by Pratt. Therefore, our algorithm allows a refined analysis by enumerating patterns that eventually occur in a small subset of the database sequences. It may yield the identification of sub-families and overcome the high computational demands typical of the multiple alignment problems. Two types of patterns, with variable or fixed length spacing between events, satisfying the user restrictions and associated options can be identified.

Two possible applications of this method are sequence *classification* [Ferreira and Azevedo 2005b, Ben-Hur and Brutlag 2003] and *clustering* [Guralnik and Karypis 2001]. Sequence patterns appear as highly discriminative features to predict a protein family or to group sequences according to their similarity. The discovery of patterns can also be used in the detection of subfamilies within a larger set of sequences from one family [Brazma et al. 1996], in a task called *SubFamiliy Detection*. The proposed method can also be used to create a profile for each of the sequences, through the respective extracted patterns.

2. Preliminaries

In this work our main concern is protein databases, thus we are only considering the alphabet of amino-acids, denoted as Σ. Each symbol of the sequence is generically called an *event* and the distance between consecutive events as *gaps*. Two main types of patterns

can be distinguished:

- **Rigid Gap Patterns** only contain gaps with a fixed length. The symbol "." is used to denote a gap of size one and it matches any symbol of the alphabet. Ex: $MN..A.CA$
- **Flexible Gap Patterns** allow a variable number of gaps between events of the sequence. We will use the notation $-x(n,m)-$ to denote a variable gap with n minimum and a m maximum number of gaps. Ex: $MN-x(1,2)-A-x(0,1)-C$

A pattern S is contained in a pattern S', if S can be obtained by dropping some events of S'. A sequence pattern S is *frequent* if it occurs in at least σ (minimum support) sequences of the database and *infrequent* otherwise. The *cover list* represents the list of all the sequence identifiers where the pattern occurs. We will follow a definition approximate to the one used in [Rigoutsos and A.Floratos 1998] and consider a pattern to be *maximal* if it is not contained in any other pattern and can not be made more specific, i.e. it is not possible to replace a wild card symbol by a concrete event and still have a frequent pattern. When extending a sequence pattern $S =< s_1\ s_2 \ldots s_n >$, with a new event s_{n+1}, S is called a *base sequence* and $S' =< s_1\ s_2\ \ldots s_n\ s_{n+1} >$ the *extended sequence*. If an event b occurs after a, it is denoted it as: $a \rightarrow b$. a is called the *predecessor (pred)* and b the *successor (succ)*. When some prior knowledge on the type of patterns that one is looking for is available, constraints appear as an efficient way to prune the search space and to focus the search on the expected patterns. The most common and generic types of constraints are:

- *Item Constraints*: restricts the set of the events (*excludedEventsSet*) that may appear in the pattern.
- *Gap Constraints*: defines the (*minGap*) minimum or the maximum distance (*maxGap*) that may occur between two adjacent events in the sequence patterns.
- *gapPenalty Constraints*: measures the density of a pattern, through the ratio between the number of concrete events and the span of the pattern.
- *Duration or Window Constraints*: defines the maximum distance (*window*) between the first and the last event of the sequence patterns.

Another useful feature in biological sequence pattern mining is the use of *Equivalent/Substitution Sets*. When used during the mining process an event can be substituted by another event belonging to the same set without lost of meaning. To report our patterns a syntax similar to the PROSITE [Bairoch 1991] syntax will be used. Since the maximal patterns may not necessarily be the most interesting ones, we designed our algorithm in order to derive both all or maximal patterns.

The problem we address in this paper can be formulated as follow: given a database of sequences D, a query sequence Q, a minimum support σ, and the optional parameters *minGap*, *maxGap*, *window*, *excludedEventsSet* and *substitutionSets*, density threshold *gapPenalty*, find all or the maximal frequent rigid or flexible gap sequence patterns present in D and contained in Q, which respect previous constraints.

Sid	Sequence
1	$< A\ B\ C\ D\ E >$
2	$< A\ C\ D >$
3	$< B\ C\ B\ C >$
4	$< B\ C\ B\ A >$
5	$< A\ B\ B\ C\ D >$

Table 1. An example database (alphabet of 5 symbols)

3. Algorithm

One way to tackle the stated problem would be to mine all frequent patterns in the database D and then select those that occur in Q. Because unnecessary computation would be done, this "*query blind*" method is clearly a naive approach. Instead we will use an adaptation of the algorithmic approach followed in [Ferreira and Azevedo 2005a] (an algorithm called *gIL*) to mine frequent sequence patterns of D present in Q, under user defined constraints. Our proposed method uses a Bottom-Up search space enumeration and a combination of frequent pairs of events to extend and find all the frequent patterns. The algorithm is divided in two phases: *scanning phase* and *sequence extension phase*. Since the frequent patterns are obtained from the set of frequent pairs, the first phase of the method consists in traversing all the sequences in the database and building two auxiliary data structures. The first structure contains the set of all pairs of events found in the database. Each pair representation points to the sequences where they appear (through a sequence identifier bitmap), see Figure 1 (a). The second data structure consists of a vertical representation of the database. It contains the positions or offsets of the events in the sequences where they occur, see Figure 1 (b). This information is required to ensure the order of the events. At the end of the scanning phase we obtain a map of all the pairs of events present in the database and a vertical format representation of the original database. In the second phase, the pairs of events are successively combined to find all the frequent patterns. These operations are fundamentally based on two properties:

Property 1 **Anti-Monotonic**, *All supersequences of an infrequent sequence are infrequent.*

Property 2 **Sequence Transitive Extension**, *Let $S =< s_1 \dots s_n >$, C_S is its cover list and O_S the list of the offset values of S for all the sequences in C_S. Let $P = (s_j \rightarrow s_m)$, C_P is it cover list and O_P the offset list of $succ(P)$ for all sequences in C_P. If $succ(S) = pred(P)$, i.e., $s_n = s_j$, then the extended sequence $E =< s_1 \dots s_n s_m >$ will occur in C_E, where $C_E = \{x \in C_S \cap C_P,\ O_P(x) > O_S(x)\}$.*

Hence, the basic idea is to successively extend all the frequent pair of events present in the query sequence Q with another frequent pair. This extension holds as long as the predecessor of the extension pair is equal to the successor of the extended sequence and the extension pair is contained and respects the order of Q. This joining step is sound provided that the above mentioned properties (1 and 2) are respected. Property 2 yields the cover list of the extended sequence (C_E) and the offsetList of $succ(E)$. The joining of pairs combined with a depth first traversal yields all the frequent patterns in the database

also contained in Q.

3.1. Scanning Phase

The first phase of the algorithm consists in the following procedure: For each sequence in D, all ordered pairs of events without repetitions are obtained. Additionally an N-bidimensional matrix (N is the size of the alphabet) is built and updated. We call this structure the *Bitmap Matrix (BM)*. Each $Cell(i, j)$ contains the information relative to the pair $i \rightarrow j$. This information consists of a bitmap that indicates the presence (1) or the absence (0) in the respective sequence (i-th bit corresponds to the sequence i in D) and an integer that contains the support count. This last value allows a fast support checking. As an example see figure 1 (a).

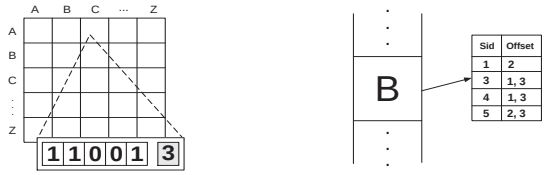

Figure 1. (a) Content of the pair $A \rightarrow C$ in the Bitmap Matrix; (b) Representation of event B in the Offset Matrix

Simultaneously, as each event in the database is being scanned, a second data structure called *Offset Matrix (OM)* is also built. Conceptually, this data structure consists of an adjacency matrix that will contain all the offset (positions) of all the events in the entire database. Each event is a key that points to a list of pairs <*Sid*,*OffsetList*>, where *OffsetList* is a list of all the positions of the event in the sequence *Sid*. The Offset Matrix is a vertical representation of the database. Figure 1 (b) shows the information stored in the Offset Matrix for the event B.One should note that the scanning phase is performed once and the two data structures are kept in main memory. Several algorithm runs with different support values do not imply additional database scanning. Only the sequence extension phase is performed during the various interactions.

3.2. Sequence Extension Phase

input : $S(BaseSequence); P(ExtensionPair); \sigma(Support)$
1 $C_S = bitmap(S)$ and $C_P = BM.bitmap(P)$
2 $C_{S'} = C_S \cap C_P$
3 **if** $support(C_{S'}) \geq \sigma$ **then**
4 return OK
5 **end**

Algorithm 1: Support Test

For implementing the extension phase we present two tests (algorithms). Conjunctively they are necessary and sufficient conditions to consider as frequent a new extended sequence. Algorithm 1 is a quick test that implements property 1. The *bitmap* function gets the correspondent bitmaps of S and P. The intersection operation is also very fast

and simple and the support function retrieves the support of the intersection bitmap. This test allows the verification of a *necessary but not sufficient* condition for the extended sequence to be frequent. A second test is necessary to ensure that the order of the events is kept along the sequences that $C_{S'}$ bitmap points to. Algorithms 1 and 2 assumes that for each frequent sequence, additional information besides the sequence event list is kept during the extension phase. Namely, the correspondent bitmap, that for the case exposed in algorithm 1 will be $C_{S'}$ if S' is determined to be frequent. Also two offset lists in the form <*Sid*, *offset*> are kept. One will contain the offset of the last event of the sequence, *offsetLastEvent*, and will be used for the "Order Test". The second, *offsetStartEvent*, contains the offset of the first event of the sequence pattern in all the Sid where it appears. This will be used when the verification of the window constraint is performed. In the Order Test given a bitmap resulted from the support test the *getSeqIdLst* function returns the list of the sequence identifiers for the bitmap. The function *offsetLst* returns a list of offset values of the event in the respective Sid. For each sequence identifier it is tested whether the extension pair has an offset greater than the offset value of the extended sequence. This implements the computation of C_E and the offsetList of $succ(E)$ as in property 2. In case the user chooses to obtain flexible gap patterns the function $getFMinMaxGap$ returns the minimum and the maximum gaps from the list of gaps (line 13). For rigid gap mode the method searches for the minimum gap value (through $getRGap$ function) that still enable the pattern to be frequent. Note that all the different gap values can be used to obtain a pattern extension with the same event. Next, all the Sid that allow a gap of this size (lines 16 to 21) are selected. At the end of the procedure (line 24) it is tested whether the order of the extended sequence pattern is respected in a sufficient number of database sequences. In the positive case the extended sequence is considered frequent.

```
   input : C_{S'}(Bitmap); S(Base Seq); P(Ext. Pair); σ(Support)
1  seqLst = getSeqIdLst(C_{S'});
2  E_v = succ(P);
3  cnt = 0;
4  foreach Sid in seqLst do
5        O_v = OM.offsetLst(Sid, E_v);
6        Y = S.offsetLastEvent(Sid);
7        W = S.offsetStartEvent(Sid);
8        if ∃X ∈ O_v, X > Y then
9              gap = X − Y; gapLst.add(gap);
10             cnt = cnt + 1;
11       end
12       if mode = FLEXIBLE then
13             (fMin, fMax) =getFMinMaxGap(gapLst);
14       else
15             rGap = getRGap(gapLst, σ);
16             foreach Sid in seqLst do
17                   Repeat Step 5 to 7;
18                   if ∃X ∈ O_v, X − Y = rGap then
19                         cnt = cnt + 1;
20                   end
21             end
22       end
23 end
24 if cnt ≥ σ then
25       return OK;
26 end
```

Algorithm 2: Order Test

Given algorithm 1 and 2, property 3 guarantees the necessary and sufficient conditions to safely extend a base sequence into a frequent one.

Property 3 **Frequent Extended Sequence**, *Given a minimum support σ, a frequent base sequence $S =< E_1 \dots E_n >$, where $|S| \geq 2$ and a pair $P = E_k \rightarrow E_w$. If $E_n = E_k$, then $S' =< E_1 \dots E_n \; g_{n,k} \; E_k >$, where $g_{n,k} = -x(mingap, maxgap)-$ if in flexible mode or $x(mingap)$ if in rigid mode, is frequent if algorithm 1 and 2 return OK.*

3.3. Space Search Traversal

Guided by the query sequence Q, the search space can be traversed using a *depth first* traversal mode. During the traversal, the set of the frequent sequences starts as the set of the frequent pairs present in Q (named as *seed pairs* - in gray at Figure 2) . The traversal begins with a sequence of size 2 that is successively expanded until it can not be further extended. If we view the search space as a tree this means that for a given branch of the tree (which represents a sequence), we step down as much as possible in the tree. Then we backtrack and re-start search using another path. An advantage of this type of traversal is that when looking for maximal sequences all the subsequences of the longest sequence found when traversing a branch of the space tree can be rejected. Consequently, it reduces the number of potential maximal sequences to consider. In the example of Figure 2, we do a level-by-level search of the frequent patterns present in Q. Each level corresponds to a position event in Q that is the prefix of all the seed pairs at that level. For instance, level 1 starts with the event at position 1 in Q and contains $A \rightarrow B$, $A \rightarrow C$ and $A \rightarrow D$. If an event is repeated in Q then the correspondent level it is not tested since all patterns obtained from the repeated seed pair will be contained in the frequent patterns found before. When extending a frequent pattern and obtaining an infrequent one, the suffix event is neglected and the posterior event in Q is then tested (ex: the pattern $AB - x(0,1) - C$ becomes infrequent when extended with B. Thus B is rejected and a new extension is tested with D, since it follows B in Q. During the mining process a list of the maximal patterns is kept. Only the longest sequence patterns (marked within a box) found are tested. The test consists in checking whether the pattern is *contained* or *contains* other sequences from this list. The maximal patterns for this example are: $AB - x(0,1) - CD$ and BCB. We should note that the procedure, described in Figure 2, is identical for rigid and flexible gap patterns. Also, in the rigid patterns extraction, the gap size of the sequences in D match the respective gaps in Q.

4. Constraints

The introduction in our method of constraints like *min/max gap, window size, items exclusion* is a straightforward process and typically translates into considerable performance gains. The introduction of *substitution sets* is also very easy to achieve. Implementing the event exclusion constraint and substitution sets requires only simple changes in the Bitmap Matrix (used to verify if patterns are frequent) and in the Offset Matrix (discriminates the positions of the events in every sequence where they occur). These features are applied between the scanning phase and the sequence extension phase, before algorithm 2 is performed.

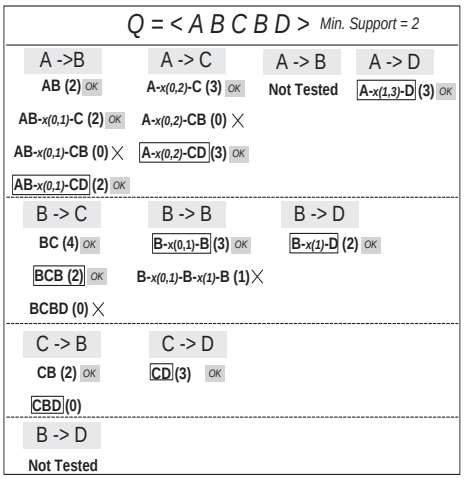

Figure 2. Example of the frequent flexible pattern finding in respect to Q and D (Table 1) for a minimum support of 2. Patterns labelled with OK are frequent. The support is also provided for each pattern.

4.1. Events Exclusion and Substitution Sets

The event exclusion constraint is applied traversing the rows and columns of the Bitmap Matrix where the excluded events occurs. At those cells, set to zero the support [1] count variable. When substitution sets are activated we have one or more sets of equivalent events. For each set of equivalent events one has to make the union of the rows (horizontal union) and columns (vertical union) in Bitmap Matrix, where those events occur. The vertical union is similar to the horizontal union. Moreover, for all the equivalent events, one needs to pairwisely intersect the sequences (through the bitmaps) where they occur and then perform the union of the offsetLists for the intersected sequences. This results in the new offsetLists of the equivalent events.

4.2. Min / Max Gap and Window Size

These constraints are trivially introduced in the "Order Test". In algorithm 2, the test in line 8 and 15 is extended with three additional tests: $(X - Y) < maxGap\ AND\ (X - Y) > minGap\ AND\ (X - W) < windowSize$.

5. Experimental Evaluation

To test our algorithm we developed a prototype written in the C++ language and compiled with g++, for linux and windows XP. All the experiments were performed on 1.5GHz Intel Centrino machine with 512MB of main memory, running windows XP Professional. Performance evaluation was done through the use of three different protein datasets which exhibit different features. Two of the datasets were obtained from PFAM (see Table 2 (a)).

[1] Future interactions on this data set still have the Bitmap Matrix intact since the bitmaps remain unchanged

The A2M consists of a family of the A-macroglobulin receptors (PF07677). The CRYSTALLIN is composed by a set of proteins that occur in high concentration in the cytoplasm of eye lens fiber cells (PF00525). The Yeast (*saccharomyces cerevisiae*) dataset is available at [GenBank]. In this section we focus our attention in the performance issues of the method. We applied a "*leave-one-out*" methodology for computational performance evaluation. The presented values represent average results. The time to perform the scanning phase is almost negligible. It takes less than 0.1 seconds for the above two datasets and 0.6 seconds for a synthetic dataset with 8000 sequences with an average length of 60.

Dataset	Protein	Intra Similarity	Avg. length	Stdev length	Nº instances
A2M	A-macroglobulin	32 %	88.4	3.03	54
CRYSTALLYN	Crystallin	67 %	53.6	1.70	13
Yeast	Yeast	43%	255	256	393

Relative σ	Absolute σ	Max.	All	Span Max.	Num. Events	Time Max (secs)
4%	2	128	253804	25.6	4.9	50.7
5%	3	119	26752	22.7	4.3	2.9
6%	4	105	8592	21.0	3.8	0.2

Table 2. (a) Properties of the Protein Datasets; (b) Results for the A2M dataset in RIGID gap mode.

Relative σ	Absolute σ	Max	All	Avg. Events	Time (secs) Max	Time (secs) All
80%	42	27777	797930	6.35	745.0	24.07
85%	45	16519	366502	5.80	95.0	11.2
90%	48	3848	102074	5.35	7.06	3.20
95%	50	1411	33810	5.03	1.41	1.07

gapPenaltyThreshold	Max	All	Avg. Events	Span Max.	Time (secs) Max
10	50	2199	3.53	6.2	0.18
15	55	4966	4.61	10.3	0.40
20	62	5202	5.40	13.6	1.09
25	71	6342	5.49	16.9	1.39
30	83	7966	5.97	21.2	2.94

Table 3. (a) Results for the A2M dataset in FLEXIBLE gap mode; (b) Results for the gap penalty threshold in the A2M dataset for a support of 8% in the rigid gap mode and mining maximal patterns.

In table 2 (b) we have, for each value of relative support, the number of All and Maximal rigid gap patterns found. As we can see, Maximal patterns represent only a small fraction of All patterns. Constraints are checked but are set to neutral values in order to not interfere in the mining process. We also show the span length and the average number of events of the maximal sequences. Results show that for lower support values more frequent patterns are found with greater average span. Table 3 shows the results for the FLEXIBLE mode applied to the A2M dataset. This mode is more expensive since the solution space is larger than in the rigid mode. The more demanding operation, largely due to the ratio All/Max patterns is the maximal test. If we ignore this test we can achieve significant time improvements. For instance in table 3(a) we can see that for a support of 80% in A2M dataset it took 745 seconds to find the maximal flexible patterns. However, when ignoring the maximal test, computation reduces to 24.07 seconds representing an improvement around 97%. It is also presented the average number of events present in a flexible pattern. Again, we can see that for lower support values the average of the pattern length and span increases. Figure 3 (a) shows runtime for the A2M dataset in Flexible mode, for a support of 80% and maximal patterns, with relation to two constraints. For a value less than 50 both constraints have a very restrictive impact in the pattern extraction process. For values greater than 70 the impact of the constraints becomes neutral. Figure 3 (b) shows the runtimes for different support values in the presence of the maxGap and window constraints. Due to its relative high intra similarity, the Crystallin dataset has a high density, in contrast to A2M dataset which is

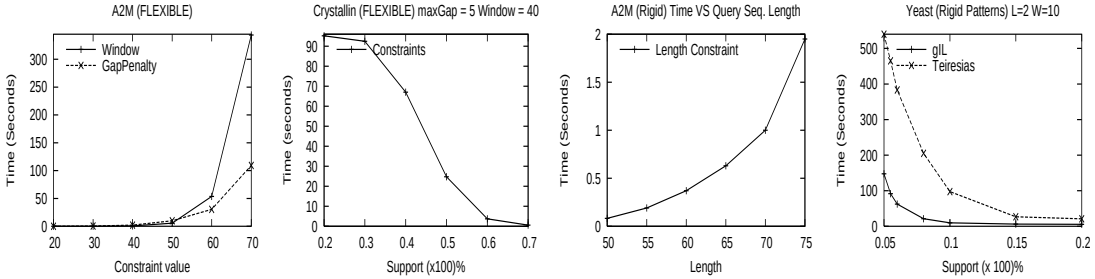

Figure 3. (a) Flexible patterns for A2M dataset in the presence of the Window and the gapPenalty Constraints; (b) Simultaneous application of the maxGap and Window constraint to find Flexible patterns in the Crystallin dataset; (c) Runtime for RIGID mode for different sizes of the query sequence A2m dataset; $\sigma = 5\%$ (d) gIL and Teiresias[L=2 and W(maxGap)= 10] algorithms, support variation for the Yeast dataset.

sparser. Additionally, the two datasets have different sizes. This explains the different range of minimum support values used. Clearly the introduction of the constraints represents a significant runtime reduction. Figure 3 (c) shows the impact of the sequence length in the algorithm runtime. From table 3 (b), we can see that the *gapPenaltyThreshold* is also a constraint with a direct impact in the number of reported patterns. Lower values for this threshold restricts the span of the patterns and consequently the average number of events present in a pattern is smaller. For a comparison of the performance of the proposed approach with two other algorithms: SPAM [Ayres et al. 2002] (for flexible gap patterns) and Teiresias [Rigoutsos and A.Floratos 1998] (for rigid gap patterns) see [Ferreira and Azevedo 2005a]. In this work we show that the gIL algorithm, in which this method is based, outperforms the methods mentioned above.

Considering all experiments, we conclude that the algorithm's runtime is essentially affected by the number of reported frequent patterns. The query sequence length and the intra similarity of the sequences in the database are the two variables that have the major impact in the number of outputted patterns. In more computationally demanding situations or when user wants to specify the type of patterns, constraints have proved to be effective and very efficient.

The efficiency of our query driven approach can be assessed through a comparison with the query blind approach. Consider a query blind application where for the A2M dataset (see section 5) and a support of 15%. Mining (all and rigid gap patterns) the entire database takes 91.5 seconds. Comparing this time (plus the time to extract only the patterns present in the query sequence) with the time that our method takes for the same support, 0.15 seconds, the first approach takes at least 610 times more time. For the Yeast dataset our method takes 0.7 and 1.2 seconds for a support of respectively 10% and 5%. The gIL algorithm [Ferreira and Azevedo 2005a] (see Figure 3 (d), for a comparison with Teiresias where L and W are respectively the number of non-wild cards events in a pattern

and the maximum spanning between two consecutive events) takes 9.6 and 147 seconds for the same support values, which is 16 and 122 times more expensive. This difference becomes more significant for lower support values. This seems a sound evidence that query driven mining appears as a promising approach.

5.1. Pattern Quality

In the past, several sequence pattern analysis algorithms have used the PROSITE [Bairoch 1991] database to evaluate their work. PROSITE contains high quality patterns, represented by an enhanced regular expression. Regular expression patterns consist in a very convenient tool for fast sequence analysis but have a limited descriptive power and do not take into account the overall diversity of the input sequences. Also they do not report deviant but closely related patterns. Enumerating all the frequent patterns that occurs in the input set provides a more complete overview of the sequence similarities.

5.1.1. Test Cases

We demonstrate the application of our algorithm and the relevance of the extracted patterns in real life data. We used two examples of well know protein sequences families, one from PROSITE (release 19.3) and one from eMotif [Wu et al.]. For each family we randomly selected one query sequence and submitted it to the sequence miner.

Zinc Finger (C2H2) {**prosite id: ps00028**}

'Zinc finger' domains are protein structures first identified in the Xenopus transcription factor TFIIIA and since than have been found in numerous nucleic acid-binding proteins. It consists in two cysteines (C) and two histidines (H) residues at both extremities of the domain. Prosite reports a dataset with 5547 sequences in Swiss-Prot containing the consensus pattern. The prosite pattern for the C2H2 domains is:

```
C - x(2,4) - C - x(3) - [LIVMFYWC] - x(8) - H - x(3,5)-H
```

Analyzing the query sequence from figure 4 (a), with a support of 40% (2219) against this dataset we obtained a total of 57 patterns, which three of them are maximal. It took 0.219 seconds to mine.

Tubulin Dataset {**emotif**}

This dataset is used as an example in the emotif database [Wu et al.]. It contains a total of 159 sequences with an average length of 40. Emotif generates patterns on a set of aligned sequences, depending on the specificity or the sentitivity required by the user. An example emotif pattern is:

```
m[fy].[kr].af[ilv]h.[fy]..egm[de]e.[de]f[ast][de]a..[dn]...l..[de][fy]..[filvy]
```

This motif matches 134 out of the 158 sequences supplied. Scanning the query sequence in figure 4(b) against the tubulin dataset with a support of 90% (143) 123 frequent patterns are found in 0.016 seconds. Two of these patterns are maximal and are aligned with the query sequence. For a support of 60% (95) 1435 sequences were found in 0.23 seconds. Seven of these sequences are maximal.

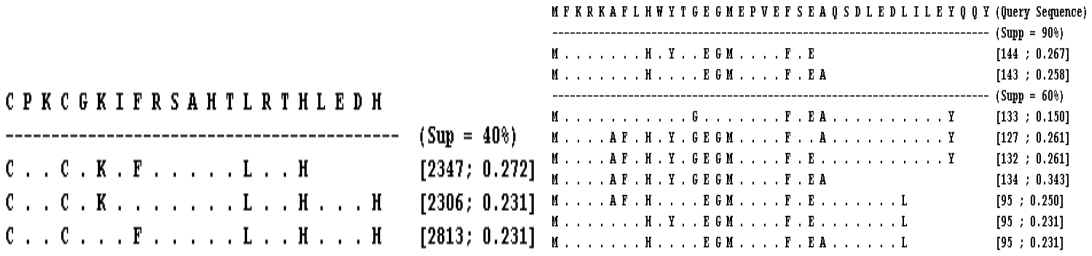

Figure 4. Extracted maximal patterns, with respective support and density: (a) Alignment of a query sequence from the C2H2 family; (b) Alignment of a query sequence from the Tubulin family.

5.2. Pattern Selection

In order to assess the quality and the usefulness of the reported patterns we apply the query driven miner to three different situations. First we define a set of statistics to be obtained for each pattern. These statistics enable the scoring selection of the most interesting patterns. We will use the term *Sensitivity* (Sn) [Brazma et al. 1995] to measure the proportion of sequences of the target family covered by the pattern. The term *Specificity* (Sp) is used to measure the proportion of sequences outside the target family that are matched by the pattern. The two measures are defined as: $Sensitivity = TP/(TP + FN); Specificity = TN/(TN + FP)$, where TP, TN and FN correspond respectively to the number of True Positives, True Negatives and False Positive matches of a pattern. Additionally, we make use of an adaptation of another measure called *Discrimination Power* (Dp) [Ben-Hur and Brutlag 2003] that we define as: $Dp = \frac{TP}{|I_F|} - \frac{FP}{|O_F|}$. In this case $|I_F|$ and $|O_F|$ correspond respectively to the number of sequences in the target family and outside the target family. Note that the range of values for Dp is between -1 and 1, whereas for the Sensitivity and Specificity is in between 0 and 1. This measure is particularly useful as a filter since the greater the Dp value, the more selective the pattern is.

5.2.1. Experiment One

In this experiment we made use of a set proteins related by a group of PROSITE (release 19.6) entries. This group is called Inhibitors and is composed by 13 entries (families). For each entry we retrieve the sequences in Swiss-Prot matching the respective pattern. We considered a unique dataset with all the 3865 sequences and for each family we randomly choose 10 sequences to mine. For each evaluated sequence we determined a base support which corresponds to the maximum support where the query sequence contains at least one frequent pattern. To obtain the frequent patterns we apply a support of 80% of the base support. Only patterns with a size greater than 3 amino-acids and a $Dp > 0$ are considered. Table 4 summarizes the obtained results for the 13 sets of proteins. The families ps00280, ps00282 and ps00286 present the set of motifs with the highest family correlation, ie. the highest Dp. Figure 5(a) shows a graph of the distribution of the

Family	Num Seq	Supp(%)	$Dp > 0$ Patterns	Total Patterns	Sn	Sp	Dp
ps00280	427	0.015	4.5	7.2	0.477	0.968	0.591
ps00281	132	0.027	10.4	20.2	0.145	0.950	0.212
ps00282	288	0.038	6.0	12.2	0.280	0.970	0.386
ps00283	604	0.026	5.0	21.6	0.107	0.990	0.183
ps00284	1230	0.030	7.2	15.0	0.050	0.982	0.086
ps00285	74	0.030	4.0	16.0	0.054	0.930	0.063
ps00286	35	0.195	6.2	7.6	0.351	0.955	0.404
ps00287	312	0.020	6.9	28.9	0.025	0.951	0.024
ps00288	64	0.035	5.7	7.9	0.155	0.926	0.221
ps00426	73	0.025	5.5	12.4	0.127	0.931	0.165
ps00477	178	0.022	7.1	28.1	0.026	0.955	0.027
ps00553	14	0.034	4.7	6.4	0.261	0.920	0.342
ps50279	434	0.015	11.0	41.3	0.040	0.974	0.060

Table 4. Results of the evaluation of the sequences from the group of Inhibitors. The last three rows shows the average results of Sn, Sp and Dp for each family.

patterns with respect to the Sensitivity versus Specificity for three of the families.

5.2.2. Experiment Two

Although our sequence miner algorithm finds interesting sequence patterns in a set of unrelated proteins, like for example in the dataset of the previous experiment where proteins from 13 families are considered together, it is best suited for sets of related proteins. In this experiment we analyze the query sequence with relation to a unique protein family. This allows us to directly control the sensitivity of the patterns since the support defines a minimum bound for this measure. In this case the specificity is defined like in [Wu et al.] as the expected number of false positives a pattern may match. So, let $S = s_1\ s_2\ \ldots\ s_n$, then $P(S) = \prod_{i=1}^{n} p(s_i)$, where $p(s_i)$ is the frequency of s_i in the Swiss-Prot database and for $s_i =' .', p(s_i) = 1$. The number of expected false positives, EFP(S)=P(S) $\times$ NResidues, where NResidues is the number of possible match positions (number of residues in Swiss-Prot). It can be verified by empirical observation that this calculation gives a reasonably estimate of the number of occurrences of a sequence pattern. We choose the family Zinc finger PHD-type (ps00516) also called C4HC3. This family contains 2654 sequences, where the sequences contain an average length of 597.8 amino-acids. In this experiment we evaluated 100 randomly chosen sequences. Only patterns with a length greater than 3 were extracted. Constraints were set to: maxgap = 20; window = 25. The average time to mine each sequence was 1.5 seconds, with an average number of 8 frequent patterns reported. For the 100 sequences a total of 1151 patterns were reported. Figure 5 (b) displays the support variation versus the expected number of false positives that the reported patterns will match in the Swiss-Prot database. We can observe that due to the large size of the family, only for small support values a significant number of patterns is found.

5.2.3. Experiment Three

In this last experiment we developed a simple classifier. We select the set of proteins that match the patterns in the group of Receptors from the PROSITE database. This group contains 27 entries matching a total of 13458 protein sequences. To classify a sequence S, we defined the Match Expectation Score (MES) with respect to a family of proteins C_i as, $MES(S|C_i) = \sum_{j=1}^{n} \{motifSpan \times support\}_j$, where n corresponds to the number of motifs for S w.r.t C_i. Next, we performed a "sequence-against-family" analysis for all the 27 families. All the sequences of each family were evaluated and the average of the

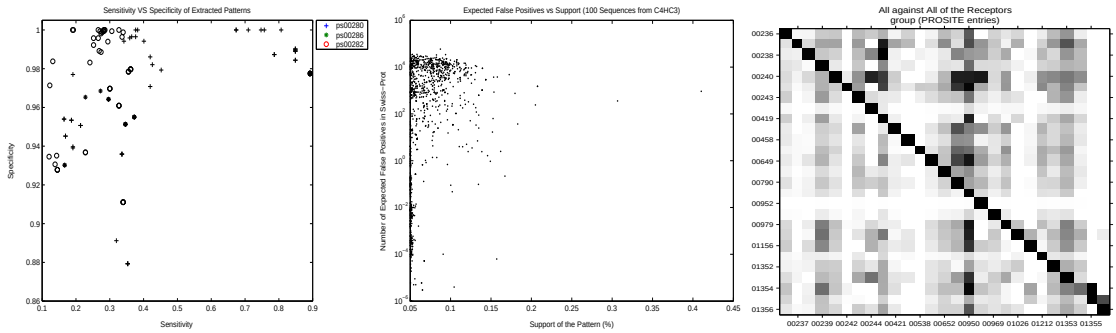

Figure 5. (a) Distribution of Sensitivity versus Specificity for the patterns from the families ps00280, ps00282 and ps00286; (b) Distribution of the expected false positive in the Swiss-Prot database for different values of support for the patterns of the sequences in C4HC3 dataset; (c) Similarity Matrix for the classification performed on the 27 families on the group of Receptors from the Prosite database. Rows and columns indicate the respective family identifier.

MES values calculated. Finally a similarity matrix for all the 27 families was obtained. This matrix is represented in figure 5 (c). The black areas indicate higher similarity and white areas a low similarity. Despite the aim of the experiment was not to build a robust classifier, the quality of the results obtained suggest the use of this approach in complex problems like protein sequence classification.

6. Conclusions

In this paper we propose a method that given a sequence to be analyzed (query sequence) and a database of sequences, it finds all the frequent patterns present in the query sequence with respect to the database. In this way, only the frequent patterns present in the query sequence are reported. It provides a closer analysis of the query sequence in relation to all the sequences in the database. The use of the method results into a considerable efficiency gain when compared to a query blind approach. It is, as far as we know, the first method to tackle the problem of sequence patterns analysis from this perspective. The method has a high adaptability making possible the extraction of two main types of patterns, *rigid* and *flexible gap patterns*, under the same algorithmic framework. The data organization turns the introduction of *constraints*, *substitution sets* and *scoring measures* a straightforward process.

7. Acknowledgments

The authors acknowledge the support of the ”Fundação para a Ciência e Tecnologia”. Pedro Ferreira is supported by a PhD scholarship (SFRH/BD/13462/2003). Authors would also like to thank to the anonymous reviewers for their valuable suggestions.

References

A. Bateman, e. a. (2003). The pfam protein families database. *Nucleic Acids Research*, vol 32, Database issue.

Ayres, J., Flannick, J., Gehrke, J., and Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *Proc. of 8th ACM SIGKDD - 2002*.

Bairoch, A. (1991). Prosite: a dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 25(19):2241–2245.

Ben-Hur, A. and Brutlag, D. (2003). Sequence motifs: highly predictive features of protein function. In *Proc. of Workshop on feature selection, Neural Information Processing Systems*.

Brazma, A., Joanassen, I., Eidhammer, I., and Gilbert, D. (1995). Approaches to the automatic discovery of patterns in biosequences. Technical Report 113, University of Bergen, Norway.

Brazma, A., Jonassen, I., Ukkonen, E., and Vilo, J. (1996). Discovering patterns and subfamilies in biosequences. In *Proc. of 4th ISMB - 1996*.

EMBL-EBI. Interproscan sequence search. http://www.ebi.ac.uk/InterProScan/.

Ferreira, P. and Azevedo, P. (2005a). Protein sequence pattern mining with constraints. In *Proc. of 9th PKDD - 2005*.

Ferreira, P. G. and Azevedo, P. (2005b). Protein sequence classification through relevant sequence mining and bayes classifiers. In *Proc. of 12th EPIA - 2005*.

GenBank. yeast (saccharomyces cerevisiae). www.maths.uq.edu.au/ fc/datasets/yeast_SC_gb117/.

Guralnik, V. and Karypis, G. (2001). A scalable algorithm for clustering protein sequences. In *Proc. of BIOKDD workshop, 7th ACM SIGKDD - 2001*.

Hirosawa, M., Totoki, Y., Hoshida, M., and Ishikawa, M. (1995). Comprehensive study on iterative algorithms of multiple sequence alignment. *Computer Applications in the Biosciences*.

Jian Pei, Jiawei Han, W. W. (2002). Mining sequential patterns with constraints in large databases. *Proc. of 11th CIKM - 2002*.

Jonassen, I., Collins, J., and Higgins, D. (1995). Finding flexible patterns in unaligned protein sequences. *Protein Science*, pages 1587–1595.

Rigoutsos, I. and A.Floratos (1998). Combinatorial pattern discovery in biological sequences: the teiresias algorithm. *Bioinformatics*, 1(14).

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proc. of 5th EDBT - 1996*.

Wu, T., Nevill-Manning, C., and Brutlag, D. emotif search. http://motif.stanford.edu/emotif/emotif-search.html.

Zaki, M. J. (2000). Sequence mining in categorical domains: Incorporating constraints. In *Proc. of 9th CIKM - 2000*.