BIOINFORMATICS

Vol. 16 no. 4 2000
Pages 341–357

SPLASH: structural pattern localization analysis by sequential histograms

Andrea Califano*

IBM TJ Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA

Received on February 22, 1999; revised on September 10, 1999; accepted on October 16, 1999

Abstract

Motivation: *The discovery of sparse amino acid patterns that match repeatedly in a set of protein sequences is an important problem in computational biology. Statistically significant patterns, that is patterns that occur more frequently than expected, may identify regions that have been preserved by evolution and which may therefore play a key functional or structural role. Sparseness can be important because a handful of non-contiguous residues may play a key role, while others, in between, may be changed without significant loss of function or structure. Similar arguments may be applied to conserved DNA patterns.*

Available sparse pattern discovery algorithms are either inefficient or impose limitations on the type of patterns that can be discovered.

Results: *This paper introduces a deterministic pattern discovery algorithm, called Splash, which can find sparse amino or nucleic acid patterns matching identically or similarly in a set of protein or DNA sequences. Sparse patterns of any length, up to the size of the input sequence, can be discovered without significant loss in performances.*

Splash is extremely efficient and embarrassingly parallel by nature. Large databases, such as a complete genome or the non-redundant SWISS-PROT database can be processed in a few hours on a typical workstation. Alternatively, a protein family or superfamily, with low overall homology, can be analyzed to discover common functional or structural signatures. Some examples of biologically interesting motifs discovered by Splash are reported for the histone I and for the G-Protein Coupled Receptor families. Due to its efficiency, Splash can be used to systematically and exhaustively identify conserved regions in protein family sets. These can then be used to build accurate and sensitive PSSM or HMM models for sequence analysis.

Availability: *Splash is available to non-commercial research centers upon request, conditional on the signing of a test field agreement.*

*The results Section **The Statistical Significance of Patterns** were obtained in collaboration with Gustavo Stolovitzky. They will appear independently in a separate joint publication.

Contact: *acal@us.ibm.com, Splash main page http://www.research.ibm.com/splash*

Introduction

Whenever Nature finds a 'recipe' to accomplish a task that gives differential fitness to an organism, chances are that such recipe will be conserved throughout evolution. At the molecular level, this means that biological sequences, belonging sometimes to widely distant species, will likely share common motifs. By motif, we mean one or more highly conserved, ungapped regions of a protein or DNA sequence (Bailey and Gribskov, 1998). In many proteins, however, a handful of residues that play a key functional role can be separated by highly variable regions. An extreme example is that of serine proteases, where three individual residues, a histidine, a serine, and an aspartic acid, in widely distant regions of the protein, join together in three dimensions to form the active site.

As a result, the identification of sparse patterns in biological databases is becoming a very transited venue within the bioinformatics community. Pattern databases such as PROSITE (Bairoch, 1991) are examples of this trend. And methods for the automatic discovery of patterns of the form $(\Sigma \cup' .')^*$, which match at least twice in a sequence or sequence database, are becoming increasingly relevant in computational biology (Brazman *et al.*, 1998). In this notation, Σ is a character from a finite-size alphabet, which can match either identically or similarly in a sequence, and '.' is the 'wildcard' character which matches any sequence character.

In recent years a number of interesting pattern discovery tools have emerged. These are divided into two main categories: statistical algorithms, such as the Gibbs Sampler (Neuwald *et al.*, 1995) and MEME (Bailey and Elkan, 1994), which use heuristics to improve efficiency, and deterministic algorithms, such as Pratt (Jonassen *et al.*, 1995) and Teiresias (Rigoutsos and Floratos, 1998). Another interesting approach is that of EMOTIF which discovers interesting motifs in pre-aligned sequences (Nevill-Manning *et al.*, 1998). These algorithms are valuable in that they can discover weak motifs hidden

in the biosequences. A discussion of some of the most interesting algorithms is available in Brazman *et al.* (1998). A statistical framework has also been proposed to help determine the statistical significance of discovered patterns (Stolovitzky and Califano, 1999). However, probabilistic approaches use heuristics or *ad hoc* constraints and become inefficient when only a small subset of the proteins is homologous. Enumeration and verification algorithms must explore an exponential search space. Therefore, practical limits are imposed on the number of characters in the pattern or in its maximum number of wildcards. Both approaches work best on databases that contain no more than a few tens of sequences. Some hybrid algorithms are more efficient but they are limited to identically occurring patterns. The rest rely on a multiple alignment preprocessing step which suffers from similar limitations (Wang and Jiang, 1994).

This paper introduces an efficient, deterministic pattern discovery algorithm called Splash, which does not require enumerating all possible patterns that could occur in the input. Splash can identify all sparse patterns, of the form $(\Sigma \cup' .')^*$, which match at least twice, either identically or similarly, in one or more sequences and satisfy a predefined density constraint. The latter limits only the maximum number of wildcards over any consecutive stretch of the pattern that does not start with a wildcard. It does not limit the total number of wildcards in the patterns. Therefore, sparse patterns that extend over the entire input sequence are possible and will be discovered efficiently. Extremely low density constraints, such as 28 wildcards in any stretch of 32 characters, are routinely used by Splash, as shown in Section **Experimental Results: G-Protein-Coupled Receptors**. Even lower densities are possible. These, however, tend to be less useful because interesting patterns would then be hidden in large numbers of random patterns. This is discussed in detail in Section **Density Constraint**, and it is supported by the structure of PROSITE patterns.

Finally, Splash can also be used to discover patterns of values from a continuous range. These will be called proximal patterns and will be more rigorously defined in Section **Definitions and Notation**.

In Sections **The Algorithm** and **Performance**, we will show that Splash is both computationally and memory efficient and significantly outperforms some of the most commonly used pattern discovery algorithms. In particular, we will demonstrate that only a minimal set of patterns must be explored in order to discover those that occur in the input. The algorithm is embarrassingly parallel in nature and it scales linearly with the number of processors. It has been implemented both for SMP and distributed architectures. In Section **The Statistical Significance of Patterns** we will discuss the statistical significance of the type of patterns discovered by Splash.

The efficiency of Splash makes it possible to explore entire databases or genomes, without resorting to supercomputers. For instance, in the single threaded version, all similar patterns occurring at least three times in the current release of the non-redundant PDB (Bernstein *et al.*, 1977), which have no more than six wildcards in any window of 12 characters, are discovered in 1h on a 266 MHz Pentium II laptop. Similarly, the parallel version of Splash takes about 8 h to discover all identical patterns that are statistically significant and have no more than five wildcards in any window of 10 characters, in a non-redundant protein database with over 90 million residues. The hardware configuration used for this purpose is a 133 MHz four way SMP workstation (F50) with 1 GB of memory.

In Sections **Experimental Results: Histones** and **Experimental Results: G-Protein-Coupled Receptors**, we will discuss some experimental findings for the histone I and for the G-protein-coupled receptor families. Typical runs required to analyze large protein families, such as the GPCR or the histone, are performed in seconds to minutes on similar configurations.

Biological applications of pattern discovery algorithms

One important, and often controversial point, about sparse patterns is whether these are meaningful biological entities by themselves. Clearly, amino acids do not freely exist in space but live in the context of a continuous sequence. Also, although signatures such as the ones available in PROSITE are useful to rapidly screen unknown sequences for functional or structural annotation, non-sparse models such as Hidden Markov Models (Krogh *et al.*, 1994) and PSSM (Bailey and Gribskov, 1998) seem to perform better.

The position of this paper is that patterns are only meaningful as a local statistical measure of sequence conservation across a set of sequences. That is, patterns that occur more frequently than expected in set protein or DNA sequences are likely to correspond to regions where lower mutation rates have been favored. These patterns may then identify regions that play an important functional or structural role as well as pinpoint which residues may play a more crucial role. The latter will be the ones that are most conserved. Therefore, unusual patterns can be used to identify and align hard to find, highly specific regions in sequences. Once these have been identified, the sparse pattern model can lead directly to a contiguous representation, via HMM or PSSM, for instance, or to multiple sequence alignments using algorithms such as MUSCA (Parida *et al.*, 1999).

This is especially important if the sequence set is large or if it contains many non-homologous sequences. In this case, an efficient pattern discovery algorithm, such as

Splash, can easily provide the core for a large number of highly specific local alignments. Again, this makes it an ideal pre-processing candidate to systematically build HMM and PSSM models.

The following is a small, non-exhaustive list of applications of the Splash algorithm, which are being currently investigated in the context of protein function and structure studies. The purpose of this section is to support the importance of efficient pattern discovery techniques in a biological context. Each of these constitutes an individual piece of work whose details extend well beyond the scope of this paper. Rather, this paper is devoted to the description of the algorithm and the discussion of the algorithm's performance.

Single (super)family analysis: Pattern discovery in a family or superfamily can identify regions that have been subjected to lower rates of mutation and that may therefore play a relevant functional or structural role. HMMs or PSSM can then be derived from these local, rigid multiple sequence alignments. These can then be used to label orphan sequences or to screen large genomic databases, such as dbEST, for unknown members of the (super)family. Conducting such an analysis systematically with existing discovery algorithms may be computationally prohibitive. Splash, however, can find all highly conserved sparse patterns in each one of the more than 1300 protein families associated with at least one PROSITE motif. This process requires about 2 h on a four-way SMP workstation.

Structural similarity: Many families, such as TNF and C1Q, show no homology at the sequence level but are clearly similar from a structural perspective. Patterns that occur across both families can highlight elements that are responsible for the structural similarity. Results obtained with Splash over these families are in fundamental agreement and extend those reported in (Shapiro and Scherer, 1998). In this paper, two short patterns, `L..G` and `G..Y`, have been identified by manual structural alignment and subsequent analysis of one C1Q and one TNF structure. By analyzing all TNF and C1Q proteins in SWISS-PROT 36, Splash finds these two most statistically significant patterns in the same regions as those of (Shapiro and Scherer, 1998).

`[ILMV] [ILMFV].L...[DQEK] [RQEHK] [ILMV]` in 52 out of 68 sequences.

`[ILMFV].....G[ILMFV]Y.[ILMFV]..[RQEHK]` in 60 out of 68 sequences.

These have been used to screen dbEST and have produced six previously unknown candidates that are now under investigation.

Pairwise (super)family analysis: Pairs of functional families that are not homologous can be analyzed to discover patterns that are common across both members. Because the number of pairs is extremely high, the efficiency of the algorithm is paramount. This is useful, for instance, in the analysis of signatures common to many different transcription factors or antibodies.

Full Genome/Database analysis: Regions that are unusually preserved across an entire database can be first automatically discovered and then clustered using an RMSD measure. Starting from the Brookhaven database, this methodology can be used to systematically identify patterns that are structurally, as well as sequentially, conserved. A small set of such three dimensional signatures, which were obtained by multiple sequence alignment, have been used successfully for predicting tertiary structure from sequence information (Bystroff and Bakel, 1988).

The problem

Let us start by defining the *identical pattern* discovery problem. Given a string of characters $S = s_1, \ldots, s_L$ from an alphabet $\Sigma = \{a_1, \ldots, a_n\}$, pattern discovery can be defined as the problem of identifying patterns of the form $(\Sigma \cup' .')^*$ that occur at least $J_0 > 1$ times in the input. We shall call a character from the alphabet Σ a 'full character'. A pattern character is then either a full character or a wildcard.

EXAMPLE. `A.B..A` is an identical pattern which is repeated twice in the string `ACBDDABBDCA`. Let us use the notation of (Brazman *et al.*, 1998) to represent a more general class of patterns that we shall call *similar patterns*. Let $K_1, \ldots, K_n$ be different subsets of Σ generated as follows: given a similarity metric $H(\Sigma, \Sigma')$ and a threshold h, for each character Σ_i the subset K_i contains all characters such that $H(\Sigma_i, \Sigma) \leq h$. Useful similarity metrics will be discussed in more detail in Section **Similarity Metrics**. A non-redundant set is obtained by removing all but one instance of any identical subset. Let us then define a similarity alphabet $\Psi = b_1, \ldots, b_n$, disjoint from Σ, with one character for each subset K_i. We will say that Ψ matches any character contained in the corresponding subset $K(\Psi)$. The characters in the class $K_i = \{a_{i1}, \ldots, a_{in_i}\}$ are usually denoted by the usual regular expression syntax as $[a_{i1} \ldots a_{in_i}]$.

As in the case of identical patterns, we shall call any character in Π a full character. We shall also say that a character $\bar{\Pi} \in \Pi$ matches a character $\bar{\Sigma} \in \Sigma$, or that $\bar{\Pi} \equiv \bar{\Sigma}$ if either $\bar{\Pi} = \bar{\Sigma}$, or $\bar{\Sigma} \in K(\bar{\Pi})$. A wildcard matches any character $\bar{\Sigma} \in \Sigma$. Then, one can define the similar pattern discovery problem as the problem of finding every pattern of the form $(\Pi \cup' .')^*$, which matches at least $J_0 > 1$ times in S.

EXAMPLE. Let $\Sigma = \{A, B, C, D\}$ be an alphabet and $H(\texttt{A},\texttt{B}) < h$, $H(\texttt{B},\texttt{A}) < h$, $H(\texttt{B},\texttt{C}) < h$, $H(\texttt{C},\texttt{B}) < h$, $H(\texttt{C},\texttt{D}) < h$, $H(\texttt{D},\texttt{C}) < h$ a similarity metric. Then $K_1 = [\texttt{AB}]$, $K_2 = [\texttt{ABC}]$, $K_3 = [\texttt{BCD}]$, and $K_4 = [\texttt{CD}]$. The similarity alphabet is then defined as $\Psi = \{\texttt{a},\texttt{b},\texttt{c},\texttt{d}\}$, with $\texttt{a} \equiv K_1, \ldots, \texttt{d} \equiv K_4$. In this case, a pattern `a . B . d` would match any of $\texttt{A} \times \texttt{B} \times \texttt{C}$, $\texttt{A} \times \texttt{B} \times \texttt{D}$, $\texttt{B} \times \texttt{B} \times \texttt{C}$, and $\texttt{B} \times \texttt{B} \times \texttt{D}$. Using the regular expression syntax, the previous pattern becomes `[AB] . B . [CD]`.

Many biological problems can be best modeled using similarity metrics rather than identity. Some useful metrics are obtained from the well established probability of mutation matrix (Schwartz and Dayhoff, 1978).

A variant of the similar pattern discovery problem is obtained by replacing the alphabet Σ with an interval of the real axis $\Re$. In that case, the pattern takes the form $(\rho \cup' .')^*$ and the similarity metric becomes a distance metric, $H(\rho, \rho') = |\rho - \rho'|$. Such patterns are called proximal patterns. They are useful, for instance, to describe correlation in the value of specific physiochemical properties of the amino acids, such as hydrophobicity or charge, in a string.

We shall call patterns that do not have leading and trailing wildcards *compact patterns*. Splash only reports compact patterns. Therefore, unless otherwise indicated, by *pattern* we will intend a *compact pattern*.

Similarity metrics

Since we are interested in similar patterns, let us define a similarity metric $H(x, y)$, where $x \in \Sigma$, $y \in \Sigma$, as a positive function of x and y, such that $H(x, x) = 0$ and which satisfies the triangle inequality:

$$H(x, y) \leq H(x, w) + H(w, y) \tag{1}$$

A symmetric similarity metric, i.e. $H(x, y) = H(y, x)$, is equivalent to a distance metric (Gerretsen, 1962). For instance, if the log-probabilities of amino acid mutation are used to define similarity classes,

$$H(x, y) = -\log\left[\frac{p(x \to y)}{p(y)}\right], \tag{2}$$

the resulting metric will be slightly asymmetric. $H($`Ala, Asp`$) < H($`Asp, Ala`$)$ because the probability of alanine to mutate into aspartic acid is slightly larger than that of aspartic acid to mutate into alanine. Known log-probability matrices satisfy the triangle inequality.

Note that the minimum of $H(x, y)$ is reached when $x = y$. The opposite is true for a score matrix where the score is highest along the diagonal. Given an alphabet Σ, one can use these functions to define the similarity classes K_i and the relative alphabets Ψ and Π.

Similarity metrics for protein sequences

As mentioned above, a convenient similarity metric for protein sequence analysis can be defined using the $-\log(P)$ of the amino acid mutation probabilities, such as the ones in PAM or BLOSUM matrices. For instance, using a BLOSUM50 mutation probability matrix (Henikoff and Henikoff, 1992), and a threshold $h = 2$, the following similarity classes $\{K\} = K_1, \ldots, K_{18}$ are obtained:

```
ala ≈ [Ala]                 arg ≈ [Arg, Lys]    asn ≈ Asn, Asp]
asp ≈ [Asp, Glu]            cys ≈ [Cys]         gln ≈ [Gln, Glu, Lys]
glu ≈ [Glu, Asp, Gln]       gly ≈ [Gly]         his ≈ [His, Tyr]
ile ≈ [Ile, Leu, Met, Val]  leu ≈ met           lys ≈ [Lys, Arg, Gln]
met ≈ [Met, Ile, Leu]       phe ≈ [Phe, Tyr]    pro ≈ [Pro]
ser ≈ [Ser, Thr]            thr ≈ ser           trp ≈ [Trp, Tyr]
tyr ≈ [Tyr, His, Phe, Trp]  val ≈ [Val, Ile]
```
(3)

Then, given the sequence

```
Ala Cys Gln Gln Val Trp Ala Gly Ala Phe Ile Tyr Leu His Pro
```
(4)

the pattern `Ala ... Ile Tyr`, for instance, would have locus $\{0, 6, 8\}$. Note, however that, based on the same metric, the pattern `Ala ... Val Trp`, which appears at position 0, would have a smaller locus $\{0, 6\}$ since both $f($`Val, Leu`$) > 2$ and $f($`Trp, His`$) > 2$.

Definitions and Notation

Let $S = s_1, \ldots, s_L$ be a string with L values from a fixed alphabet Σ. This could be the DNA/RNA alphabet $\Sigma_{\text{DNA}} = \{\texttt{A},\texttt{C},\texttt{G},\texttt{T}\}$, $\Sigma_{\text{RNA}} = \{\texttt{A},\texttt{C},\texttt{G},\texttt{U}\}$, or the protein alphabet $\Sigma_{\text{P}} = \{\texttt{A},\texttt{R},\texttt{N},\texttt{D},\texttt{C},\texttt{Q},\texttt{E},\texttt{G},\texttt{H},\texttt{I},\texttt{L},\texttt{K},\texttt{M},\texttt{F},\texttt{P},\texttt{S},\texttt{T},\texttt{W},\texttt{Y},\texttt{V}\}$. Σ could also be a value from a continuous interval on the real line. If Σ is discrete, given a similarity metric, one can compute the alphabets Ψ and Π, as described in the previous section. A *pattern* π is then any string of the form $(\Pi \cup' .')^*$. If Σ is continuous, then a pattern π is any string of the form $(\rho \cup .)^*$. This is also described in the previous section.

Let us call $C_\pi = \{(\Pi_i, d_i) \mid i = 1, \ldots, k\}$ the *composition* of π. This is formed by taking each full character in the pattern and its relative position, in characters, starting at zero, on the pattern. The composition of a pattern and the number of leading and trailing wildcards define it unambiguously. The length of a pattern $k = |C_\pi|$ is the total number of full characters it contains. A pattern of length k is called a k-pattern. The span of a pattern $l = d_k + 1$ is the total number of characters (full characters and wildcards) it contains. A k-pattern with a span of l characters is called a kl pattern.

For instance, `A . B . . C` has length 3, span 6, and composition $\{(\texttt{A}, 0), (\texttt{B}, 2), (\texttt{C}, 5)\}$, and `[AB] . B . . [BCD]` has length 3, span 6, and composition $\{([\texttt{AB}], 0), (\texttt{B}, 2), ([\texttt{BCD}], 5)\}$. Compact patterns start and end in a full character, therefore they have $d_1 = 0$.

By definition, a k-pattern matches (or occurs) at offset w on a string S if and only if, for each i, there exists a $\Pi_i \in \Pi$ such that

$$s_{w+d_i} \in \Pi_i \tag{5}$$

for each value of $1 \leq i \leq k$. The set of j absolute offsets where a pattern matches in S is called the *pattern locus*.

This is represented as $W_\pi = \{w_1, \ldots, w_j\}$. A *kl*-pattern that matches at $j = |W_\pi|$ offsets in S is called a *jkl*-pattern. j is called the pattern *support*.

Finally, the comb of a pattern is the string obtained by replacing each full character with a 1 and each wildcard with a zero. The term *k*-comb and *kl*-comb have similar meaning as the corresponding terms for patterns.

EXAMPLE. The comb of `A..BC..D...E` is 100110010001.

Given a string S, by $\pi_{\Psi,w}$ we shall indicate the pattern determined by the comb Ψ at offset w on S.

EXAMPLE. Let `ABCDEFGHI` be a string. Then the pattern $\pi_{1001,3}$ is `D..G`.

Basic pattern operators and relationships

The purpose of this section is to define a small number of pattern operators and relationships that will be conveniently used for a compact description of the algorithm.

Trim operator: $\pi = \mathrm{T}_t(\pi)$ is a pattern obtained by removing all leading and trailing wildcards from π.

EXAMPLE. Let `..A..B...` be a pattern π. Then $\mathrm{T}_t(\pi) =$ `A..B`.

Translation operator: $\pi = \pi + x$ is a pattern with locus $W' = W + x = \{w_1 + x, w_2 + x, \ldots, w_j + x\}$ and composition $C' = C - x = \{(\Pi_i, d_i - x)\}$.

EXAMPLE. Let `A..B` be a pattern with locus $\{5, 12\}$. Such pattern, translated by -2, becomes `..A..B` with locus $\{3, 10\}$. Patterns produced by positive translations cannot be conveniently represented as strings. However, they can still be represented by their composition and locus.

Append operator: $\pi_c = \pi_a \oplus \pi_b$ is a pattern with all the characters of π_a followed by all the characters of π_b. If the span of π_a is l, then π_c has locus $W(\pi_c) = W(\pi_a) \cap (W(\pi_b) - l)$ and composition $C_c = C_a \cup (C_b - l)$.

EXAMPLE. Let $\pi_a =$`A..B.C` and $\pi_b =$`B.C`. Then $\pi_a \oplus \pi_b =$`A..B.CB.C`.

Add operator: $\pi_c = \pi_a + \pi_b$ is a new pattern with locus $W(\pi_c) = W(\pi_a) \cap W(\pi_b)$ and composition $C_c = C_a \cup C_b$.

EXAMPLE. Let $\pi_a =$`A..B.C...D` be a pattern with locus $\{2, 13, 25, 40\}$ and $\pi_b =$`...B.CE` a pattern with locus $\{2, 7, 13, 25, 49\}$. Then, $\pi_c = \pi_a + \pi_b =$`A..B.CE..D`, with locus $\{2, 13, 25\} = \{2, 13, 25, 40\} \cap \{2, 7, 13, 25, 49\}$.

If there exists a translation x such that the intersection of the locus of two patterns $W(\pi_c) = W(\pi_a) \cap W(\pi_b) - x$ is non null, it is sometimes convenient to add π_a and $\pi_b - x$ to form a longer pattern.

Using these operators, we can define the following pattern relations.

Sub-pattern (super-pattern): π_b is a sub-pattern of π_a if there exists a translation x such that $W(\pi_a) \supseteq W(\pi_b) + x$ and $C_{\pi_b} \supset C_{\pi_a} - x$. Conversely, π_a is a super-pattern of π_b if π_b is a sub-pattern of π_a.

EXAMPLE. `A..BC.D` with locus $\{1, 31, 76\}$ is a sub-pattern of `BC.D` with locus $\{4, 12, 34, 61, 79\}$. The translation is $x = 3$.

Aligned sub-pattern (super-pattern): π_a is an aligned sub-pattern of π_b if it is a sub-pattern of π_b for $x = 0$. Then, π_b is an aligned super-pattern of π_a.

EXAMPLE. Given `A..B.C...D` with locus $\{2, 35, 60\}$ and `A..B` with locus $\{2, 10, 35, 60\}$, then `A..B.C..D` is an aligned sub-pattern of `A..B` because, for a translation value $x = 0$, $\{2, 10, 35, 60\} \supseteq \{2, 35, 60\}$.

Optimally aligned sub-pattern (super-pattern): π_b, with span l, is an optimally aligned sub-pattern of π_a if π_b has more full characters than π_a and π_b is identical to the first l characters of π_a. Then, π_a is an optimally aligned super-pattern of π_b.

By definition, an optimally aligned sub/super-pattern is also an aligned sub/super-pattern. That is, $W(\pi_b) \supseteq W(\pi_a)$ and vice versa.

For simplicity, if π_b is an optimally aligned sub-pattern of π_a, we shall call π_b a child of π_a and π_a a parent of π_b.

EXAMPLE. Both `A..BC..E` and `A..BC.D` are children of `A..BC`.

Equivalent sub-pattern (super-pattern): π_b is an equivalent sub-pattern of π_a if it is a sub-pattern of π_a and $W(\pi_a) = W(\pi_b)$. Then, π_a is an equivalent super-pattern of π_b.

EXAMPLE. `AB.D`, with locus $\{0, 4\}$, in `ABCDABQD`, is an equivalent sub-pattern of both `A..D`, with locus $\{0, 4\}$, and `B.D`, with locus $\{1, 5\}$.

Maximal patterns

A pattern π is said to be *maximal in composition* (or *c*-maximal) if there does not exist any equivalent sub-pattern of π with the same span. That is, if π cannot be extended to a longer pattern, by dereferencing one of its wildcards into a full character, without decreasing its support. A pattern is said to be *left maximal* (or l-maximal) if there does not exist any equivalent sub-pattern obtained by appending it to another pattern. That is, if it cannot be extended into a longer pattern, by appending it to another pattern, without decreasing its support. A pattern is said to be *right maximal* (or r-maximal) if there does not exist any equivalent sub-pattern obtained by appending another pattern it to. That is, if it cannot be extended into a longer pattern, by appending another pattern to it, without decreasing its support.

A *maximal pattern* is a pattern that does not have any equivalent sub-pattern. That is, it is l-, c-, and r-maximal. Maximality is an essential property of pattern discovery algorithms. It avoids reporting a combinatorial number of super-patterns of each maximal pattern in the sequence. We will require that the Splash algorithm report only maximal patterns.

An important class, as we shall see later, is that of patterns that are both left maximal and maximal in composition. These will be called lc-maximal patterns, for short.

EXAMPLE. Let the pattern `A.B..C` occur 100 times in a string and the pattern `A.B..C.EF` occur twice. That is, in two of the 100 occurrences `A.B..C` is flanked by `.EF`. Then `A.B..C` occurring 100 times is maximal, `A.B..C.EF`, occurring twice, is also maximal. However `A.B..C.E`, occuring twice, and `A....C`, occurring 100 times, are not maximal.

Density constraint

It is important to be able to specify the minimum number of full characters k_0 required in any substring of a pattern which starts with a full character and has length l_0. It is intuitive to interpret $r = k_0/l_0$ as a density of full characters per unit length. Therefore, we shall call such a constraint the *density constraint*. Given k_0 and l_0, we shall say that a comb is a $\langle k_0, l_0\rangle$-valid comb (or simply a valid comb) if either its length is smaller than k_0 or the relation $d_{i+k_0-1} - d_i \leq l_0$ is satisfied for each i such that $1 \leq i \leq k - k_0 + 1$. A pattern with a valid comb is called a valid pattern. It is useful to consider valid even patterns that have fewer than k_0 full characters, as they may be later extended to form longer patterns with more than k_0 full characters and would otherwise be immediately rejected. In any case, as we shall see later, patterns that have fewer than $K_0 \leq k_0$ full characters, with K_0 a user defined threshold, will not be reported anyway in the end.

EXAMPLE. The pattern `A....BC....D` satisfies the density constraint $k_0 = 3$, $l_0 = 7$ because all substrings (`A....BC` and `BC....D`), of length 7, starting on a full character, have at least three full characters. The fact that `..BC...` of length 7 has only two full characters is irrelevant because it does not start with a full character. The fact that substring `C....D` has two full characters is also irrelevant because the pattern contains only six positions. `A....B` is also valid pattern, for the same parameters, because it has fewer than three full characters.

If one were not to specify such constraint, j-patterns in random strings would be distributed around an average length.

$$\bar{k} = L \sum_i p(\Sigma_i) \left[\sum_j p(\Sigma_j)\alpha_{ij} \right]^{j-1} \tag{6}$$

where $[p(\Sigma_i)]^{-1}$ is the frequency of the symbol Σ_i in S and α_{ij} is 1 if $H(\Sigma_i, \Sigma_j) \leq h$ and 0 otherwise. If Σ is an interval of the real axis, the sum must be replaced by an integral.

The Splash algorithm is usually run at three different densities, from $k_0 = 4$, $l_0 = 8, 16, 32$, aimed at dense to very sparse patterns. Lower densities are not generally useful as they tend to produce large numbers of patterns that are not statistically significant. It is unlikely that biologically significant, rigid patterns contain more than 28 consecutive wildcards. For instance, the PROSITE database contains only one rigid pattern with more than 20 consecutive wildcards (PS001254). This is because, in general, long regions without matching full characters include loop regions, where insertions and deletions are likely. Not many very sparse flexible patterns are reported in the literature. For instance, PROSITE contains only eight flexible patterns with more than 20 consecutive wildcards. These are detected by Splash as two or more separate rigid patterns and can be easily fused in a simple post-processing phase. The algorithm has been implemented so that it can call itself recursively. This allows the user to find flexible patterns by discovering any rigid patterns that occur within a predefined window on the left and on the right of any other rigid pattern. The process can be repeated until no more patterns are found. This deterministic approach produces flexible patterns defined by rigid components separated by variable-length strings of wildcards.

The Algorithm

Let us define the input of the algorithm as a string $S = s_1, \ldots, s_L$ and a set of parameters k_0, l_0, J_0, and K_0. Let us also define the set Q'_s of patterns that occur at least $J_0 > 1$ times in S, and which are $\langle k_0, l_0\rangle$-valid. We shall call any such pattern a *final pattern*. The output of the algorithm is then the set $Q_s = \{\bar{\pi}\} = \mathrm{T}_t(Q'_s(k \geq K_0))$

of all final patterns which have length $k \geq K_0 \geq k_0$, with their leading and trailing wildcards removed by the trim operator. The latter are called *reported patterns*. The extension from one to multiple sequences is trivial and will be assumed in the rest of the paper.

We shall start by defining a method to construct an initial set of patterns $P_s = \{\pi\}$, which we call a Seed Set, from any string S. P_s either contains a parent of any final pattern $\bar{\pi}$ or $\bar{\pi}$ itself. We shall then define a recursive Splash operator $\mathrm{T}_s(P_s, Q_s)$, $\mathrm{T}_s^n(P_s, Q_s) = \mathrm{T}_s(\mathrm{T}_s^{n-1}(P_s, Q_s))$ to process the input set P_s and the output set Q_s. On the first iteration $Q_s = \varnothing$. At each iteration, this operator performs three tasks: (a) it eliminates patterns in the input that are not lc-maximal; (b) it systematically extends all input patterns into valid patterns that have at least one more character; (c) it inserts any pattern which is a final pattern in Q_s.

We shall demonstrate that, at each iteration n, the set $P_s^n = \mathrm{T}_s^n(P_s, Q_s)$ contains at least a parent of any final pattern that is not yet in Q_s. Therefore, when $\mathrm{T}_s^{n_\varnothing}(P_s, Q_s) = \varnothing$, Q_s will contain all the final patterns. Finally, we shall demonstrate that the algorithm converges because $n_\varnothing$ is finite. The theorems and demonstrations apply to identical, similar, and proximal patterns.

Let us start by defining the following dataset and operators.

Seed Set: Given the string S and the parameters l_0, k_0, J_0, K_0, let $Q_s = \{\bar{\pi}\}$ be the set of all the final patterns of S. Then a Seed Set $P_s = \{\pi\}$ is any set of patterns that either (a) contains at least one parent π of each pattern $\bar{\pi}$ or (b) contains $\bar{\pi}$. In other words, any final pattern is either contained in P_s or it is a child, possibly with a smaller locus, of at least one pattern in P_s.

EXAMPLE. Given the string ABCDABEFABC and the constraints $k_0 = 2$, $l_0 = 4$, $J_0 = 2$, and $K_0 = 2$, the maximal patterns are AB..AB with locus $\{0, 4\}$, ABC with locus $\{0, 8\}$, and AB with locus $\{0, 4, 8\}$. All three of them can be obtained by extending the pattern AB with locus $\{0, 4, 8\}$. Therefore, the set {AB} is a Seed Set.

Canonical Seed Set: A Seed Set P_s is a Canonical Seed Set if it does not have any subset which is also a Seed Set. That is, each valid pattern on S is either present once in P_s or it is a subpattern of one and only one pattern in P_s.

EXAMPLE. Given the string ABCDABQD and the constraints $k_0 = 2$, $l_0 = 3$, $J_0 = 2$, $K_0 = 2$, the set {AB} is a Canonical Seed Set. This is because AB is the only parent in {AB} of the only valid and maximal pattern, AB.D. The set {AB,A.C} is not canonical because the valid pattern AB.D is a sub-pattern of both AB and A.C.

A trivial Canonical Seed Set, for $J_0 = 2$, $k_0 = 1$, $K_0 = 1$, and any l_0 is $\{\pi_\varnothing\}$, where $\pi_\varnothing$ is the pattern with empty composition and locus $W(\pi_\varnothing) = \{1, 2, \ldots, L\}$, which includes all the possible offsets of S. This is because, by definition, this pattern is a super-pattern of any other pattern in s. Since this set contains only one pattern it is, by definition, canonical.

A convenient way to generate Seed Sets of a less general nature is to start from all possible

$$\binom{l_0 - 1}{k_0 - 1} \tag{7}$$

k_0-combs, $\{\psi\}_{k_0}$, with $k_0 \leq K_0$, that start with a 1 and have $k_0 - 1$ 1s in the next $l_0 - 1$ consecutive positions. For instance, for $k_0 = 2$ and $l_0 = 5$, there are four of these combs, 11000, 10100, 10010, and 10001. Given one such comb ψ, one can list all the different patterns $\pi_{w,\psi}$, identified by the comb and by an offset w on S, and evaluate their support based on the similarity metric. Finally, one can remove all the patterns with support $j < J_0$. The resulting set will be called the $\langle l_0, k_0 \rangle$-set or P_s^* for short. The following pseudocode illustrates a procedure for assembling P_s^*:

```
Create an empty Seed Set P*_s = ∅;
for each comb ψ ∈ {ψ}_k0 {
  for each offset w on S and corresponding seed pattern π_ψ,w {
    Create an initial locus W_ψ,w = {w} with just that offset;
    count = 1;
    for each other offset w' on S such that w ≠ w' {
      if (π_ψ,w matches π_ψ,w') {
        add w' to W;
        count ++;
      }
    }
    //Check that W_ψ,w is not the same as any other W_ψ,w'
      with w' < w
    NotPreviouslyFound = true;
    for each offset w' on S such that w' < w {
      if (W_ψ,w ≡ W_ψ,w') {
        //the two loci are identical
        NotPreviouslyFound = false;
        break;
      }
    }
    //If this is a new pattern with a valid support
    if (NotPreviouslyFound && (count ≥ J0)){
      add π_ψ,w to P*_s
    }
    return P*_s
}
```

THEOREM 1. *The set P_s^* is a Seed Set.*

Proofs of all theorems can be found in the Appendix. Sets of this nature can be built efficiently by means of a look-up table, such as the one described in the Flash algorithm (Califano and Rigoutsos, 1993). This procedure works also for continuous values, based on the definition of a match using a distance metric.

Maximality

As we discussed before, only maximal patterns should be reported. Otherwise a combinatorial set of super-patterns of any given pattern would be reported. Since, as we shall see, we extend patterns exclusively by appending other patterns to them, if we were to extend any pattern that is not lc-maximal we would be guaranteed to get a non-maximal pattern. On the other hand, by extending in this way an lc-maximal pattern, we may get new patterns that are maximal.

For instance, given the string ABCAABCACBCD, the pattern BC with locus $\{1, 5, 9\}$ is lc-maximal. However, BCA is not lc-maximal since it is an equivalent super-pattern of ABCA. If we were to extend BCA, therefore, we would get non-maximal patterns.

We define the operator $\mathrm{T}_m(\pi)$ to check for pattern lc-maximality:

$$\begin{cases} \mathrm{T}_m(\pi) = \{\pi\} & \text{if } \pi \text{ is } lc\texttt{-maximal} \\ \mathrm{T}_m(\pi) = \varnothing & \text{if } \pi \text{ is not } lc\texttt{-maximal} \\ \mathrm{T}_m(\{\pi_i\}) \equiv \cup \mathrm{T}_m(\pi_i) & \end{cases} \quad (8)$$

THEOREM 2. *Any parent of a final pattern is lc-maximal.*

It follows that if P_s is a Seed Set, $\mathrm{T}_m(P_s)$ is also a Seed Set.

THEOREM 3. *Any valid lc-maximal pattern π is either a final pattern or it is a parent of one final pattern with the same support.*

THEOREM 4. *If a set P_s contains only unique patterns and no pattern $\pi \in P_s$ is a child of another pattern $\pi' \in P_s$, then the set $\mathrm{T}_m(P_s)$ is a Canonical Seed Set.*

For a compact description of T_m, let us define a Boolean function $E(W, x)$ which is true if and only if there exists any character $\Pi_i \in \Pi$ such that $s(w_m + x) = \Pi_i$ for each value of $1 \le m \le j$. Then, T_m can be implemented by the following pseudo code:

```
T_m(π) {
  //First check that pattern is composition-maximal
  for each offset x corresponding to a wildcard in π {
    if (E(W_π, x)) {
      return ∅;
    }
  }
  // Then Check for left-maximality
  for each offset x such that −l_0 + d_{k_0−1} < x < 0
    if (E(W_π, x)) {
      return ∅
    }
  }
  return {π}
}
```

This operator is a key component of the efficiency of the Splash algorithm because it eliminates most potential pattern candidates very early, before they have a chance to contribute to an exponential growth of the number of hypotheses. If this operator is not implemented and patterns are checked *a posteriori* for maximality, both memory and computational efficiency of the algorithm decrease exponentially in the size of the input.

The Histogram operator

Given a jkl-pattern π, the Histogram operator T_h is used to create all the possible valid $(l+1)$-patterns $\{\dot{\pi}\}$, obtained by appending a string of the form $'\cdot' \star \Pi' \cdot' \star$ to π, which have support at least J_0.

Given an lc-maximal pattern, π, of length k, we can define the *extension interval* I_π, adjacent to π, such that π would be maximal unless it could be extended, without reducing its support, by at least one full character in I_π.

EXAMPLE. Let A.B be a lc-maximal pattern and $k_0 = 3$, $l_0 = 5$ be density parameters. Then A.B would be maximal unless one could extend it, without reducing its support, by at least one full character at either one or two positions to the right of the B. Similarly, the lc-maximal pattern A.B.CD would also be maximal for the same parameters unless one could extend it by one full character, without reducing its support, either at one, two, or three positions to the right of the C.

I_π can be defined as follows:

$$\begin{cases} I_\pi = [1, l_0 - l - (k_0 - k)] & \text{if } k < k_0 \\ I_\pi = [1, l_0 + d_{k-k_0+2} - l] & \text{if } k \ge k_0 \end{cases} \quad (9)$$

Here, l is the span of π. In the previous example,

$$\begin{aligned} I_{A.B} &= [1, 5 - 3 - (3 - 2)] = [1, 2] \\ I_{A.B.CD} &= [1, 5 + 4 - 6] = [1, 3] \end{aligned} \quad (10)$$

The Histogram operator, T_h, which, given a pattern π, produces the set $\{\dot{\pi}\}$ can be defined as follows. For each offset $x \in I_\pi$, let us define all the positions $w + l + x$, with $w \in W_\pi$, the locus of π. Our goal is to find how many of the similarity classes Π_i are supported by at least J_0 elements in each set defined by a specific value of x. Any such class and offset x can be used to extend π into a longer pattern $\dot{\pi}$ which is valid by definition. If a pattern $\dot{\pi}$ has support less than J_0 it is eliminated.

If the alphabet is discrete, this can be accomplished with an array $C[x, \Pi_i]$, with one entry for each value of the offset $x \in I_\pi$ and for each similarity class in Π. This is summarized by the following pseudo code:

```
T_h(π, Q_s) {
  create an empty result set R = ∅
  for each x in I_π {
    // Initialize the counters
    for each Π_i ∈ Π {
```

```
      set each C[x, Π_i] = ∅
    }
    for each w ∈ W_π {
      c = S(w + l + x)
      for each Π_i ∈ Π such that c ∈ Π_i {
        add w to C[x, Π_i]
      }
    }
    // Remove all redundant counters
    for each Π_i ∈ Π {
      for each Π_i' ∈ Π with i' > i {
        if C[x, Π_i] = C[x, Π_i'] {
          set C[x, Π_i'] = ∅
        }
        }
        if |C[x, Π_i]| ≥ J_0 {
          create a pattern π' with the character Π_i
            at offset
          x and wildcards at any other position I_π
          set π̇ = π ⊕ π'
          add the pattern π̇ to R,
        }
      }
    }
    if no pattern π̇ ∈ R has support W_π̇ = W_π {
      add π to Q_S
    }
    return R
}
```

If Σ is an interval of the real axis, instead, one can use a similar deterministic procedure to count all sets of values that satisfy the distance metric constraints. This can be done efficiently, for instance, (a) by sorting the values $s(w + l + x)$, for each $w \in W(\pi)$ and for a fixed value of x; (b) starting at each value, by counting how many consecutive values satisfy the distance metric constraints; (c) by removing any set that is a subset of another set.

If none of the patterns $\dot{\pi}$ is an equivalent sub-pattern of π, i.e. it has the same support of π, then π is maximal. In that case, if it has at least $K_0 \geq k_0$ full characters, it is a reported pattern and it is added to Q_S. As a result, π is never contained in $\{\dot{\pi}\}$.

EXAMPLE. Let S = ABCDEABCCEABCDA be a string and AB be a pattern. Given the constraints $k_0 = 3$, $l_0 = 5$, the extension interval is $I_{\mathtt{AB}} = [1, 3]$. If the only similarity class is [CD], then the only array entries with a positive count are $C[1, \mathtt{C}] = \{2, 7, 12\}$, $C[2, \mathtt{D}] = \{2, 12\}$, $C[2, [\mathtt{CD}]] = \{2, 7, 12\}$, and $C[3, \mathtt{E}] = \{2, 7\}$. As a consequence, the following patterns can be generated:

$x = 1$ $\dot{\pi}_0 = \mathtt{ABC..}$; $W_{\pi_0} = \{0, 5, 10\}$
$x = 2$ $\dot{\pi}_1 = \mathtt{AB.D.}$; $W_{\pi_1} = \{0, 10\}$ and
$\dot{\pi}_2 = \mathtt{AB.[CD].}$; $W_{\pi_2} = \{0, 5, 10\}$
$x = 3$ $\dot{\pi}_3 = \mathtt{AB..E}$; $W_{\pi_3} = \{0, 5\}$

Since both $\dot{\pi}_0$ and $\dot{\pi}_2$ are equivalent sub-patterns of AB, the latter is not a maximal pattern.

Enumerate operator

T_h has been used to list all possible valid, single full character extensions $\{\dot{\pi}\}$ of π. Then, the purpose of the enumerate operator $\mathrm{T}_e(\{\dot{\pi}\})$ is to create all the possible valid patterns which extend π by one, two, or more full characters in the extension interval I_π. These can be obtained by adding together any possible subset of $\{\dot{\pi}\}$. For instance, if π = AB and $\{\dot{\pi}\}$ contains the patterns AB.C.. and AB..D., then the pattern AB.CD = AB.C..+AB..D. also extends AB. Any such pattern is valid by definition because it is formed by adding valid patterns and can therefore only be denser. Resulting patterns that have support less than J_0 are eliminated.

Let us define $\{\tilde{\pi}\} = \mathrm{T}_e(\{\dot{\pi}\}) = \mathrm{T}_e(\mathrm{T}_h(\pi, Q_s))$ as the set of all possible patterns with support $j \geq J_0$, obtained by adding any possible subset of $\{\dot{\pi}\}$.

THEOREM 5. *The set $\{\tilde{\pi}\}$ is a Seed Set for any final pattern that is a child of π.*

Note that, since independent subsets of $\{\dot{\pi}\}$ are unique, the prerequisites of Theorem 4 are also satisfied. That is, no two patterns of $\{\dot{\pi}\}$ are either identical or have a parent–child relation. Therefore, from Theorems 5 and 4, $\mathrm{T}_m(\tilde{\pi})$ is a Canonical Seed Set.

The Enumerate operator $\mathrm{T}_e(\{\dot{\pi}\})$ can be defined recursively and efficiently by the following pseudocode:

```
T_e({π̇}) {
  if ({π̇} = ∅ {
    return ∅
  }else {
    define a results set R_S = {π̇};
    for each pattern π̇_i in {π̇} {
      initialize a temporary empty set T_S = ∅;
      for each pattern π̇_i' in {π̇} such that i' > i {
        compute π̃ = π̇_i + π̇_i'
        if π̃ has support greater or equal than J_0 {
            insert π̃ in T_S;
          }
        }
        R_S = R_S ∪ T_e{T_S}
      }
    }
  }
  return R_S;
}
```

Splash

Given the Seed Set P_s^* the Splash algorithm can be compactly represented by the following recursive notation:

$$\begin{cases} \mathrm{T}_s^n(P_s^*, Q_s) = \mathrm{T}_s(\mathrm{T}_s^{n-1}(P_s^*, Q_s)) \\ \mathrm{T}_s(P_s^*, Q_s) = \mathrm{T}_{eh}(\mathrm{T}_m(P_s^*), Q_s) \end{cases} \tag{11}$$

where

$$\mathrm{T}_{eh}(\{\pi\}, Q_s) = \cup_\pi \mathrm{T}_e(\mathrm{T}_h(\pi, Q_s)) \tag{12}$$

Pattern discovery terminates at the nth iteration, if $T_s^n(P_s^*, Q_s) = \varnothing$. The following six properties of the algorithm have been demonstrated.

1. An initial Seed Set can be constructed for any set of constraints. From Theorem 1, P_s^* is a Seed Set for any choice of l_0, k_0, J_0, K_0.
2. At any iteration n, $T_s^n(P_s^*, Q_s)$ is a Seed Set for any final pattern not contained in the result set Q_s. This follows from Theorems 2, 3 and 5.
3. As a consequence, when $T_s^{n\varnothing}(P_s^*, Q_s) = \varnothing$, all final patterns are contained in Q_s.
4. The algorithm converges. At each iteration n, patterns in $T_s^{n+1}(P_s^*, Q_s)$ contain at least one more full character than patterns in $T_s^n(P_s^*, Q_s)$. Therefore, for $n_\varnothing \leq L$, $T_s^{n\varnothing}(P_s^*, Q_s) = \varnothing$.
5. Patterns in Q_s are maximal. This follows by the definition of the histogram operator.
6. The algorithm is efficient. Theorem 4 proves that the set $T_m(T_s^n(P_s^*, Q_s))$ is a Canonical Seed Set for any final pattern not in Q_s. This is, by definition, the smallest set of patterns that all remaining final patterns can be derived from.

Parallelism

As shown in previous sections, each Seed Pattern π can be processed independently of any other Seed Pattern, as there are no global dependencies in any of the operators. Even the lc-maximal operator, $T_m(\pi)$, which must verify that π is not a sub-pattern of any other maximal pattern, can be implemented without any knowledge of other discovered patterns. It follows that the algorithm is embarrassingly parallel in the number of patterns in the Seed Set. The only globally required information is the input string S. Also, final results must be collected and reported.

A typical approach to parallelize this kind of algorithm is to use a function such as $f(\pi) = [g(\pi) \bmod N_{\text{pr}}]$, where $g(\pi)$ is a hash function that returns a positive integer based on the pattern π and N_{pr} is the number of available processors. Then, each node operates on the partial Seed Set $P_s^*(N_{\text{pr}} = f(P_s^*)$ and returns a partial results set $Q_s(N_{\text{pr}})$ such that $Q_s = \cup Q_s(N_{\text{pr}})$. More sophisticated load-balancing functions can be used as well.

The Statistical Significance of Patterns

In Stolovitzky and Califano (1999), it is shown how the average number of maximal jkl-patterns in a random database s of length L, with a $\langle l_0, k_0 \rangle$ density constraint, is given by

$$\langle n_{jkl} \rangle \approx N_0(k,l) \binom{L-l+2}{j} \langle \wp \rangle^{k(j-1)} [1 - \langle \wp \rangle^k]^{L-l-2-j} \langle p_{\text{in}} \rangle \langle p_{\text{out}} \rangle \quad (13)$$

In the above expression, $N_0(k, l)$ is the number of $\langle l_0, k_0 \rangle$-valid combs that have a span l, and length k. $\langle \wp \rangle$ is given by the following expression,

$$\langle \wp \rangle = \sum_{\Sigma} p(\Sigma)[\wp(\Sigma)], \quad (14)$$

where $p(\Sigma)$ is the probability of the value Σ to occur in the sequence and

$$\wp(\Sigma) = \sum_{\Sigma'} p(\Sigma') H(\Sigma, \Sigma') \quad (15)$$

is the probability of the value Σ to randomly match any other character in S. Also, p_{in} and p_{out} are respectively the probability that a given jkl-pattern is maximal in composition and length. From this analysis, it is possible to estimate the probability that any discovered pattern would have occurred in a random database of similar size and composition. This probability is close to a binomial distribution and its first and second moment, its mean and variance, are well defined. Therefore, it is useful to compute a z-score as:

$$z = \frac{n_{jkl} - \langle n_{jkl} \rangle}{\sigma_{n_{jkl}}} \quad (16)$$

where n_{jkl} is the number of discovered jkl-patterns. Full details of this analysis, which is in excellent agreement with experimental data, are available in Stolovitzky and Califano (1999).

Performance

To test the scaleability of the algorithm we have run a range of comparison tests against Pratt and MEME. The tests show that Splash significantly outperforms both algorithms in terms of raw performance, limitations on discovered patterns, and scaleability. Because MEME is geared towards the discovery of the most conserved regions in protein families rather than of the exhaustive set of conserved patterns, it cannot be compared directly. This will be discussed in Section 6.2.

Pratt

The performance of Splash and Pratt as a function of an increasing database size is shown in Figure 1. The database is produced as follows. First an appropriate sequence sample set is selected from the Brookhaven PDB database to obtain the desired size. Then the position of the amino acids are randomized. This results in a random database of an appropriate length, with the same amino acid frequency as the corresponding PDB sample. Patterns that occur in more that 20% of the sequences in the database are reported. Total size of the databases is 8192×2^i, with $0 \leq i \leq 6$. Databases for values of i ranging from 0 to 6 are processed. The largest random

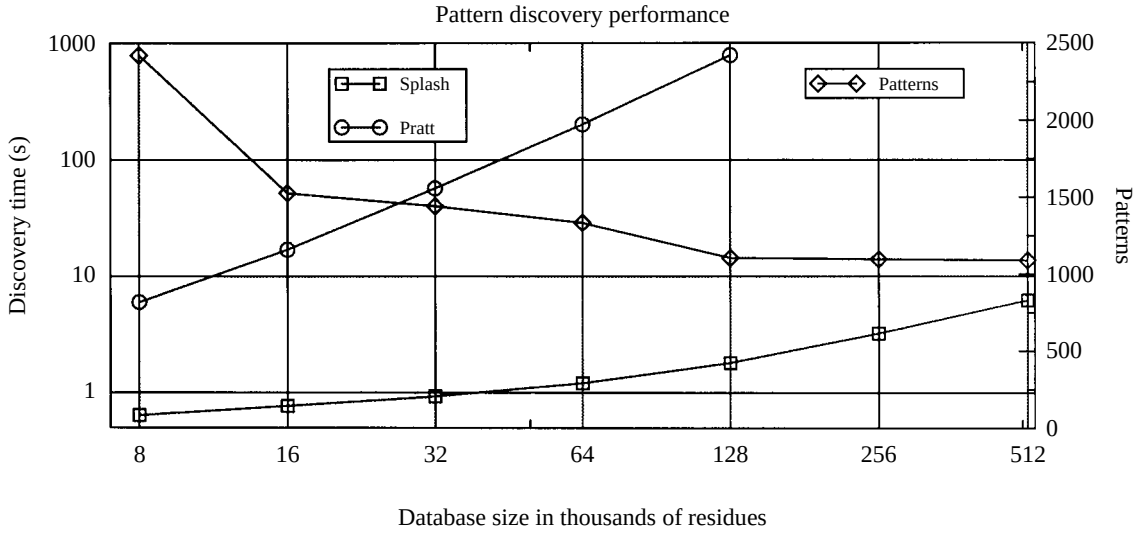

Fig. 1. Splash and Pratt: Time versus Database Size. Pattern discovery time is reported versus database size. Identical patterns are reported by the two algorithms. Discovery parameters are $k_0 = 2$, $l_0 = 5$, support is 20% of sequences in databases.

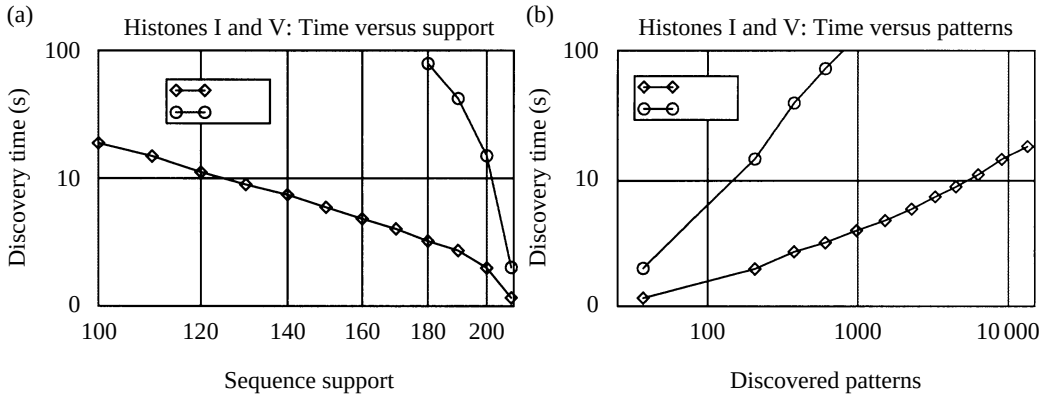

Fig. 2. Splash versus Pratt: (a) versus pattern support j, (b) versus the number of discovered patterns.

database is approximately 512 000 residues. On the left Y axis, we show the time, in seconds, required by Splash and Pratt to process the random databases in log scale. On the X axis, we show the database size, also in log scale. On the right Y axis, we show the number of discovered patterns in linear scale. This is shown by the curve with the diamond symbol. The density constraints are $k_0 = 2$, $l_0 = 5$. The maximum memory footprint of the program is 12 Mb.

The patterns discovered by Pratt are identical to those discovered by Splash. However, Splash is increasingly faster than Pratt as the database size increases. At the smallest database size, 8 KRes, it is about six times faster. At size 128 KRes., it is about three orders of magnitude faster. Also, while patterns reported by Pratt are limited to a maximum span, 50 characters in this case, Splash would discover any pattern that satisfies the density and support constraints, no matter how long.

We report similar performance measurement against a histone I database (Makalowska *et al.*, 1999), with 209 proteins, at increasingly higher values of the support. This is an interesting case because this database is pattern-rich, generating in excess of 10 000 patterns for $k_0 = 2$, $l_0 = 5$, and $J_0 = 100$.

The discovery performance for Splash and Pratt is shown in Figure 2. Pratt crashes on our machine for support $J_0 < 180$. In Figure 2(a), time is shown as a function of an increasing value for J_0, the minimum number of sequences containing the pattern. In Figure 2(b) time is shown as a function of the number of discovered patterns. For $J_0 = 180$, Splash is almost two orders of magnitude faster. The

maximum memory footprint of the program is 8 Mb for $J_0 = 100$.

Meme

MEME, which is based on a PSSM model, is geared towards the discovery of the most conserved contiguous regions in a protein set. Also, the density constraint does not make sense in the context of PSSM. Therefore, a straightforward comparison is difficult and probably inappropriate. To compare the performance of the two algorithms, therefore, we only tested Splash's ability to discover the most conserved regions in a protein set. We run against the lipocalin file which is included with the MEME distribution. MEME takes 8.9 s to discover the two top patterns in the five sequences. Splash discovered two patterns that extend over the same regions in 1.16 s. These were the two most statistically significant patterns among those that occur in all sequences, for $k_0 = 4$, and $l_0 = 8$. As discussed, this is the standard initial density constraint used with Splash to discover PROSITE-like patterns or motifs.

To test scaleability, we also performed the analysis of the histone I family described in Section **Pratt**. Splash takes 8.7 s to discover the motif `G . S . . . [ILMV] . . . [ILMV]` which occurs in every sequence. This motif is discussed in detail in Section **Experimental Results: Histones**. On the same CPU, MEME takes 3771.38 s to discover a PSSM spanning the first six characters of the pattern discovered by Splash.

Density constraints

Finally, the impact of the density constraints on performance is given in Figure 3. This is an important parameter, as it determines the ability of the algorithm to discover either dense or sparse patterns. Here we show the performance of pattern discovery, for a substantial variety of density constraints, against a set of 124 proteins, about 150 KRes, that contain the aspartic acid and asparagine hydroxylation site. For $k_0 = 2$ we use $l_0 = 2, 4, 8, 16, 32$. For $k_0 = 4$, we use $l_0 = 4, 8, 16, 32$. Larger values of l_0 result in patterns that are too sparse and not statistically significant.

This protein set has been obtained from the PROSITE entry PS00010. The goal is to automatically discover the PROSITE signature `C-x-[DN]-x(4)-[FY]-x-C-x-C`. The support constraint, then, is set at $J_0 = 124$, or 100% of the proteins in the set. BLOSUM50, with a threshold $h_0 = 2$, is used to produce the similarity metric. That is, pairs of amino acids that score better or equal to 2 in BLOSUM50 are considered similar. Other values for the support parameters are possible as well, for instance, to discover PROSITE signatures that do not occur in all proteins of a functional family. A full scale analysis of the PROSITE database, however, is beyond the scope of this paper and has been conducted separately. Results will be reported in a follow-up paper.

A pattern virtually identical to the PROSITE signatures (`C-x-[DN]-x(4)-[FYW]-x-C-x-C`) is correctly discovered as the most statistically significant pattern for $k_0 = 2$, $l_0 = 16, 32$ and for $k_0 = 4$, $l_0 = 16, 32$. Figure 3(b) gives the number of discovered patterns for the various parameter choices. As shown here, there are no patterns discovered for $k_0 = 4$, $l_0 < 16$ and just a small number of patterns, with two amino acids only and low statistical significance (e.g., `G . . . . . D`), for $k_0 = 2$, $l_0 < 16$. Figure 3(a) shows the time, in seconds, as a function of the density constraint parameters. As can be seen, the efficiency of the algorithm allows one to explore a considerable range of density and support parameters in minimal time.

Sensitivity of identity versus similarity metrics

To analyze the increase in sensitivity resulting from similar pattern discovery, the following analysis has been performed. First, 10 identical copies of a random sequence of length 40 have been generated with composition identical to the histone I family (Makalowska *et al.*, 1999). Then, each sequence has been padded left and right to a predefined length L with random amino acids using the same composition. From this set, subsequent mutated populations of identical length and with the same number of sequences have been generated using the procedure described in (Dayhoff *et al.*, 1978), starting at an evolutionary period of 5 PAMs and increasing that in steps of 2 PAMs. Finally pattern analysis is performed using either an identity or a similarity metric. The number of statistically significant patterns (z-score $>$ 3) are computed. Results for identical pattern discovery, for increasing values of L, are shown in Figure 4(a). Results for similar pattern discovery are shown in Figure 4(b).

The goal of this exercise is to model the likelihood of detecting an active site (chosen to be 40 residue in length) which has been preferentially mutated to preserve amino acids of similar properties, in sequences of different length. The statistical significance of pattern after x PAMs is a measure of the correlation of the original sequences. This is an oversimplified model and does not accurately model evolution. However, as discussed in Dayhoff *et al.* (1978), it is indicative of the amount of conservation one can expect in a fairly small peptide region.

The similarity metric of Section **Similarity Metrics for Protein Sequences**, has been used. This is based on BLOSUM50 with a threshold of 2. Populations have been produced by repeatedly applying PAM1 to the original sequences. We have deliberately mixed PAM and BLOSUM matrices, during simulated evolution and analysis, to show that even with different models, similar patterns

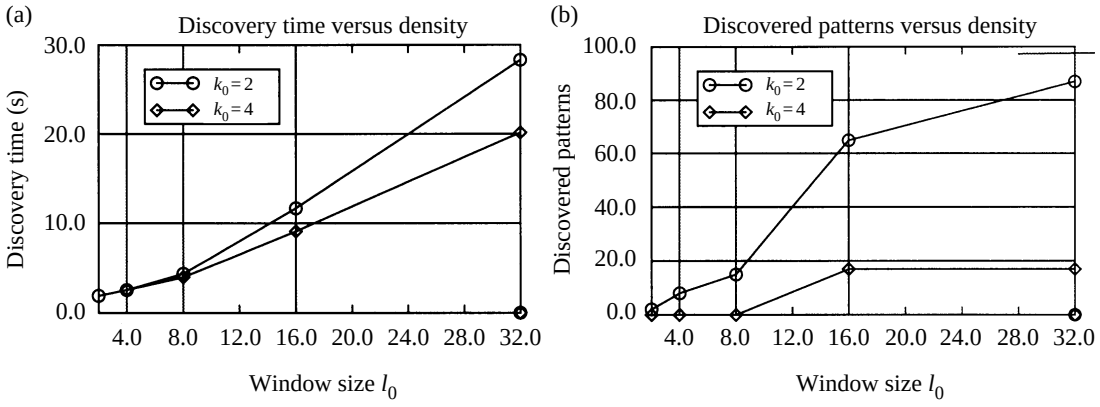

Fig. 3. Pattern discovery performance as a function of the density constraint. Runs on the PS00010 PROSITE family with $k_0 = 2$, $l_0 = 2, 4, 8, 16, 32$ and $k_0 = 4$, $l_0 = 4, 8, 16, 32$.

are better at pinpointing distant relationships. The number of statistically significant patterns discovered with the identity metric drops to zero quickly. With the exception of the shortest sequence length $L = 100$, which generates the smallest amount of noise (i.e. patterns that are not statistically significant), no pattern is reported after more than 11 PAMs. As shown, the similarity metric practically doubles the time horizon.

Experimental Results: Histones

An important measure of a pattern discovery algorithm's performance is its ability to discover hidden motifs in large protein databases. A typical approach is to start with a minimum support equal to a high percentage of the total number of proteins in the set, say 95%, and a fairly high density constraint. Typical values are $k_0 = 4$, $l_0 = 8$. BLOSUM50 or BLOSUM62 can be used with a threshold of 1 or 2. If no interesting motif is discovered, first the density constraint is decreased ($l_0 = 12, 16, \ldots$), until patterns that are not statistically significant start appearing (typically $l_0 \leq 40$ for a few hundred proteins). If statistically significant patterns still fail to occur, the minimum support is progressively dropped to increasingly lower values and the cycle is restarted.

In this case, we have run Splash against a histone I database with 209 proteins (Makalowska *et al.*, 1999). If a similarity metric defined by the BLOSUM50 matrix with a threshold of 2 is used and the density constraint has $k_0 = 4$ and $l_0 = 12$ (this is the second attempt in the above procedure), Splash takes 8.7 s to detects the following motif:

$$\pi_0 = \texttt{G.S...[ILMV]...[ILMV]},$$
$$(j = 209, Z_s = 3.013 \cdot 10^5)$$

Pratt and Meme take respectively 66 s and 3771, to find basically the same protein region. The z-score Z_s is computed from the analysis described in Section **The Statistical Significance of Patterns**. It is defined as:

$$\mathbf{Z}_s[\pi_{jkl}] = \frac{n(\pi_{jkl}) - \bar{n}(\pi_{jkl})}{\sigma(\pi_{jkl})}, \qquad (17)$$

where $\bar{n}(\pi_{jkl})$ and $\sigma(\pi_{jkl})$ are, respectively, the mean and the variance of the number of jkl-patterns that would occur in a random database of that size and composition. Here $n(\pi_{jkl}) = 1$ because we are considering each pattern independently. That is, the jkl-pattern occurred at least once.

Given the very large value of the Z_s, it is highly unlikely that a pattern like this would arise spontaneously from a database with a histone-like amino acid frequency of the same size. The analysis, accounts for the fact that two of the four residues match exactly and two match similarly.

This is confirmed empirically by the analysis of the three dimensional structures of the globular domain of the only two histone proteins where the motif occurs in PDB. These are respectively the globular domain of the H1 (Cerf *et al.*, 1993) and H5 (Ramakrishnan *et al.*, 1993) histone from chicken. Note that although H1 and H5 are homologous, H5 proteins were not in the database used to discover the motif. As can be seen from Figure 5(a) and (b) where the two motifs are highlighted, there is significant structural similarity. This occurs in the region between residues 22 and 32 on H1 and between residues 44 and 54 on H5.

The glycine and serine are exposed on the surface of the protein in a loop at the end of an alpha-helix. The two leucines in H1, mutated into isolucines in H5, contribute to the stability of the hydrophobic core on the alpha-helix. Although the structural properties of this area of the histone globular domain have not been fully understood, it could play an important structural role

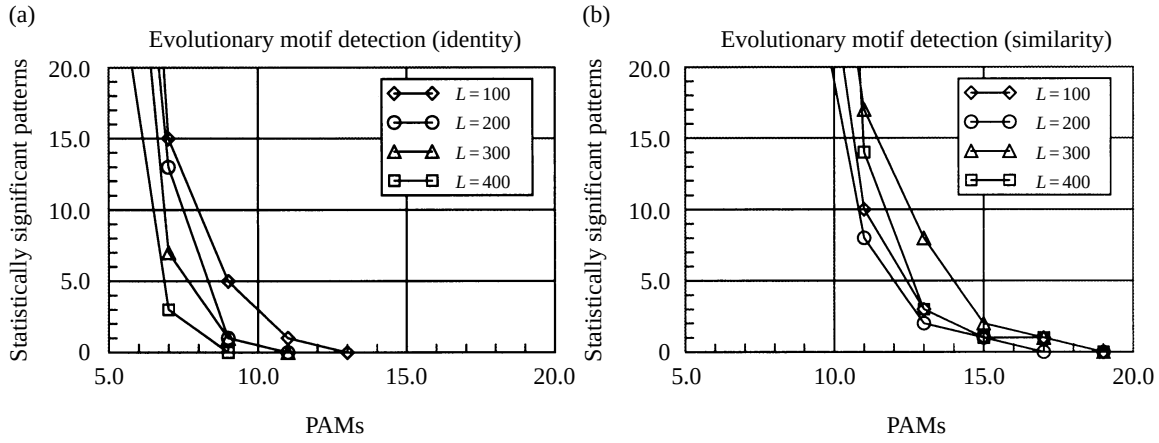

Fig. 4. Statistically significant patterns as a function of simulated evolution. Different curves model the detection of a 40 amino acid conserved region within a longer uncorrelated sequence.

because it harbors the most conserved motif in the whole histone I family.

It should be noted that, in this case, the ability to discover patterns with a similarity metric plays an important role. In particular, there are no three full character sub-patterns of π_0 that appear in all the sequences. The sub-pattern π_3 = G . S . . . . . K, which occurs in 201 proteins, has $Z_s = -0.64$ which makes it virtually indistinguishable from background noise. In fact, there are about 170 patterns of this kind spread all over the input sequences. From a statistical point of view, patterns containing two full characters are not significant in this database. Therefore, without the use of a similarity metric, directly in the discovery process, pattern π_0 would not readily emerge.

Experimental Results: G-Protein-Coupled Receptors

Another result is shown from a superfamily of 918 known and hypothetical G-Protein-Coupled Receptors. Here, Splash has been used to discover statistically significant motifs that occur in the largest number of protein sequences in a set, with a procedure identical to that of Section **Experimental Results: Histones**.

If an identity metric is used, even with a very low density constraint defined by $k_0 = 4$ and $l_0 = 30$, a single four amino acid pattern is found that occurs in more than half of the proteins:

$$\pi_0 = \texttt{N} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \texttt{DRY},$$
$$(j = 474,\ Z_s = -11.13)$$

This pattern contains the previously known and well preserved DRY tripeptide for family A. Based on the Z_s, however, this pattern would be difficult to separate from the noise.

Under identical conditions, if the similarity metric defined by the BLOSUM50 matrix with a threshold of 2 is used with the same density constraint, almost 500 patterns are discovered. These have at least two identical and two similar residues and occur in more than half of the proteins. The most frequent pattern with a $Z_s \geq 3$ is

$$\pi_1 = \texttt{C} \ldots\ldots\ldots\ldots\ldots \texttt{ILMV]} \ldots$$
$$\texttt{[ILMV]} \ldots \texttt{[NDE]R} \ldots \texttt{[ILMV]},$$
$$j = 619;\ Z_s = 2.26 \times 10^{34}.$$

This highly statistically significant pattern has two identical and three similar residue matches. This motif, used to search against SWISS-PROT Rel. 36, returns 874 GPCR proteins and only 30 potential false positives. About half of these are hypothetical proteins or proteins with a putative annotation.

Even a pattern with a significantly lower chance of appearing in a large database than π_0, such as

$$\pi_2 = \texttt{C} \ldots\ldots\ldots\ldots\ldots \texttt{L} \ldots \texttt{[ILMV]} \ldots$$
$$\texttt{[DEB][RQK][HFWY]} \ldots \texttt{[ILMV]},$$
$$j = 483;\ Z_s = 2.46 \times 10^{34}$$

with two identical and five similar amino acid matches, still occurs in more proteins than the identical pattern π_0, and it has a much higher Z_s, 2.46×10^{34} versus -11.13. Note that the z-score is not a measure of how likely a motif is to occur in a database. Rather it is a measure of how likely the motif would be to occur j times in a random database of the same size and composition.

This motif is extremely selective. If it is used to search against SWISS-PROT Rel. 36, it returns 578 GPCR proteins and only four possible false positives. These are:

1. C555_METCA Cytochrome C-555,

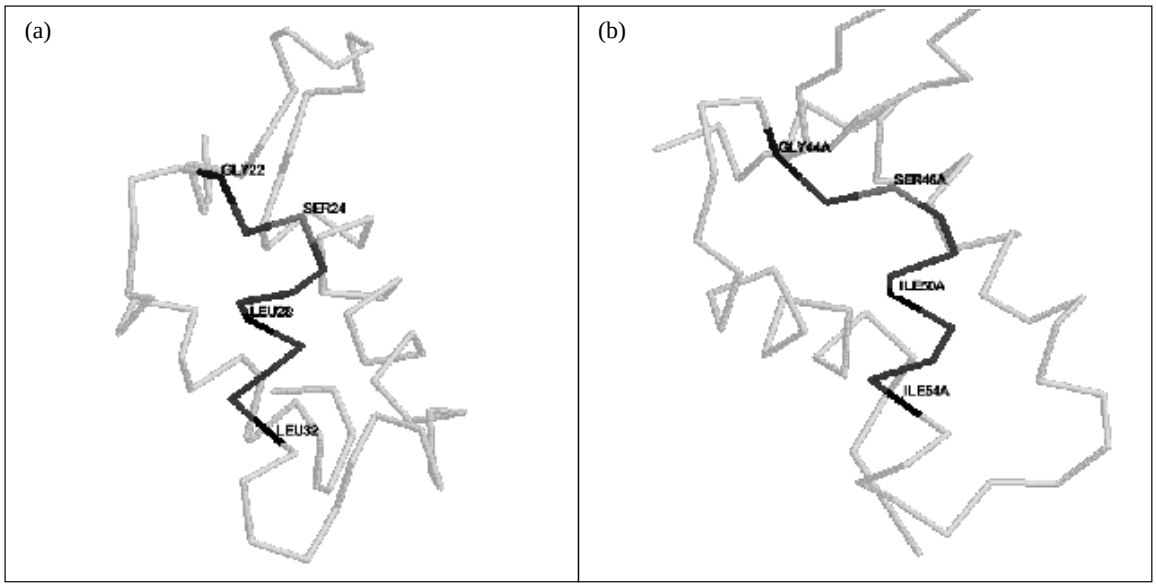

Fig. 5. Globular domain of histone I and V proteins. (a) Chicken histone H1, (b) chicken histone H5.

2. KEMK_MOUSE: Putative serine/threonine-Protein Kinase EMK,
3. SRG8_CAEEL: SRG-8 Protein,
4. UL29_HCMVA: Hypothetical Protein UL29.

Given the extremely low false positive ratio (0.68%), it is quite possible that either two or four may have been mislabeled and should really be considered members of the GPCR family.

We have been unable to replicate these results using either MEME or Pratt. MEME was still running after several days. PRATT crashed on our hardware configuration. We attributed this to a combination of the database size, its low overall homology, and its high pairwise homology.

In this paper we limit ourselves to single pattern analysis. However, given the large number of statistically significant yet different patterns that can be discovered, even better classification results can be obtained by using multiple statistically significant patterns simultaneously. This will be the subject of a follow-up paper.

Conclusions

This paper introduces Splash, a new deterministic algorithm for identical, similar, and proximal pattern discovery. The most significant contribution of Splash is its efficiency and scaleability without any sacrifice in terms of the completeness of the results. Very large biological sets, such as the full Swissprot or PDB database or entire genomes, can be processed by Splash in hours on conventional workstation class hardware. These reduced computational requirements allow a significant portion of the search parameter space to be explored and studied. As a result, motifs that would normally elude discovery can be quickly and systematically identified.

The resulting motifs can be used for a variety of biologically significant purposes, from automatic sequence clustering, to the definition of HMM and PSSM models for accurate and sensitive sequence screening, to multiple sequence alignment.

The paper also shows that, as expected from pairwise sequence comparison, the use of a similarity metric significantly increases the chances of detecting statistically significant motifs in distantly related families. Finally, by coupling the algorithmic part with the analysis of the statistical significance of patterns, we have shown that interesting motifs can, in some cases, be separated from the large background of patterns that are not statistically significant discovered in large data sets.

Splash is currently being used for a wide variety of analysis tasks that range from the automated functional and structural annotation of orphan sequences, to the systematic, automatic discovery of PROSITE motifs.

Acknowledgments

We would like to thank Gustavo Stolovitzky, Ajay Royyuru, and Laxmi Parida for their valuable help with this paper and for many useful suggestions. We would also like to thank Barry Robson, Aris Floratos, Yuan Gao, and

Mike Pitman for the many useful discussions on the topic of pattern discovery.

Appendix

THEOREM 1, PROOF. Given any final pattern $\bar{\pi}$ on S, let us construct π_{k_0}, a parent of $\bar{\pi}$, by selecting its k_0 leftmost full characters. If $\bar{\pi}$ is a final pattern, it must have at least k_0 full characters and the first k_0 ones must span no more than l_0 positions. Given that $\bar{\pi}$ must also satisfy the support requirements $j \geq J_0$, then also π_{k_0} satisfies it because, as a super pattern, its locus includes that of $\bar{\pi}$. Then, π_{k_0} is contained in the set P_s^* because this set contains all patterns that have exactly k_0 full characters over at most l_0 positions and satisfy the support requirements.

THEOREM 2, PROOF. Let us define π^{-i}, a parent of a final pattern $\bar{\pi}$ of length k, obtained by removing the last i full characters of $\bar{\pi}$, with $i < k$. By definition, the support of π^{-i} is greater or equal to that of $\bar{\pi}$ because $W(\pi^{-i}) \supseteq W(\bar{\pi})$. Therefore, if π^{-i} were not lc-maximal then, any child of π^{-i}, including $\bar{\pi}$, would also not be lc-maximal. Since any parent of $\bar{\pi}$ is of the form π^{-i}, then any parent of a final pattern must be lc-maximal.

THEOREM 3, PROOF. Consider the set of all the valid, equivalent sub-patterns of a pattern π. If the set is empty, then π is a final pattern by definition. Otherwise, if the string S is finite, there must be one such sub-pattern π' that does not have any other valid, equivalent sub-pattern. By definition that pattern is a final pattern. π is a parent of π' because, by definition, an lc-maximal pattern does not have any equivalent sub-pattern that is not a child.

THEOREM 4, PROOF. From Theorem 3, it follows that $\mathrm{T}_m(\{\pi\})$ can only contain final patterns or parents of a final pattern $\bar{\pi}$. Suppose there are two independent parents of $\bar{\pi}$ in $\mathrm{T}_m(\{\pi\})$. Then, say that the first one, π_k, contains the k leftmost full characters of $\bar{\pi}$ and that the second one, $\pi_{k'}$, contains the $k' \leq k$ leftmost full characters of $\bar{\pi}$. If $k = k'$, then $\pi_k \equiv \pi_{k'}$ which violates the first assumptions. If $k' < k$, π_k is a child of $\pi_{k'}$, which violates the second assumption.

THEOREM 5, PROOF. If π is maximal, it has no valid sub-patterns and $\{\tilde{\pi}\} = \{\dot{\pi}\} = \varnothing$ satisfies our requirements. Otherwise, let $\bar{\pi}$ be a final pattern which is a child of π. There must be a non-empty set of full characters $\{\Pi_i(x_i)\}$ in $\bar{\pi}$ that occur at an offset $x_i \in I_\pi$ after π. Otherwise, the density constraint would be violated. Each such full character must occur at least J_0 times in S at the offset x_i after π. Then, each one of these full characters is detected by the histogram operator and it is used to extend π, by a single full character, into a pattern $\dot{\pi}_i \in \{\dot{\pi}\}$. The latter can be written as $\dot{\pi}_i = \pi \oplus \pi_i$, where π_i consists of $|I_\pi|$ wildcards and the character Π_i at the offset x_i. The set of all the patterns $\{\dot{\pi}_i\}$, formed from $\{\Pi_i(x_i)\}$, is therefore a subset of $\{\dot{\pi}\}$. By construction, the pattern obtained by adding all the $\dot{\pi}_i$ together is contained in the set $\{\tilde{\pi}\}$. This pattern is also a parent of $\bar{\pi}$ since its k full characters are the first k full characters of $\bar{\pi}$.

References

Bailey,T.L. and Elkan,C. (1994) Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second ISMB Conference*. AAAI Press, Menlo Park, pp. 28–36.

Bailey,T.L. and Gribskov,M. (1998) Methods and statistics for combining motif match scores. *J. Comp. Biol.*, **5**, 211–221.

Bairoch,A. (1991) PROSITE: a dictionary of sites and patterns in proteins. *Nucl. Acids Res.*, **19**, 2241–2245.

Bernstein,F.C., Koetzle,T.F., Williams,G.J. B., Meyer, Jr.,E.F., Brice,M.D., Rodgers,J.R., Kennard,O., Shimanouchi,T. and Tasumi,M. (1977) The protein data bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol.*, **112**, 535–542.

Brazman,A.*et al.* (1998) Approaches to the automatic discovery of patterns in biosequences. *J. Comp. Biol.*, **5**, 279–305.

Bystroff,C. and Bakel,D. (1988) Prediction of local structure in proteins using a library of sequence-structure motifs. *J. Mol. Biol.*, **281**, 565–577.

Califano,A. and Rigoutsos,I. (1993) FLASH: A fast look-up algorithm for string homology. In *Proceedings of 1993 ISMB*. Bethesda, MD, pp. 56–64.

Cerf,C., Lippens,G., Mulyldermans,S., Segers,A., Ramakrishnan,V., Wodak,S.J., Hallenga,K. and Wyns,L. (1993) Homo- and heteronuclear two-dimensional NMR studies of the globular domain of Histone H1: Sequential assignment and secondary structure. *Biochemistry*, **32**, 11345.

Dayhoff,M.O., Schwartz,R.M. and Orcutt,B.C. (1978) A model of evolutionary change in proteins. In Dayhoff,M.O. (ed.), *Atlas of Protein Sequence and Structure*, pp. 345–352.

Gerretsen,J.C. H. (1962) *Lectures on Tensor Calculus and Differential Geometry*. Noorthoff,P. (ed.), N. V. Groningem.

Henikoff,S. and Henikoff,J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA*, **89**, 10915–10919.

Jonassen,I., Collins,J.F. and Higgins,D.G. (1995) Finding flexible patterns in unaligned protein sequences. *Prot. Sci.*, **4**, 1587–1595.

Krogh,A., Brown,M., Mian,I.S., Sjoelander,K. and Haussler,D. (1994) Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.*, **235**, 1501–1531.

Makalowska,I., Ferlanti,E.S., Baxevanis,A.D. and Landsman,D. (1999) Histone sequence Database: sequences, structures, post-translational modifications and genetic loci. *Nucl. Acids Res.*, **27**, 323–324.

Neuwald,, Liu, and Lawrence, (1995) Gibbs motif sampling: detection of bacterial outer membrane protein repeats. *Prot. Sci.*, **4**, 1618–1632.

Nevill-Manning,C.G., Wu,T.D. and Brutlag,D.L. (1998) Highly specific protein sequence motifs for genome analysis. *Proc. Natl Acad. Sci. USA*, **95**, 5865–5871.

Parida,L., Floratos,A. and Rigoutsos,I. (1999) An approximation al-

gorithm for alignment of multiple sequences using motif discovery. *J. Comb. Opt.*, **3**, 247–275.

Ramakrishnan,V., Finch,J.T., Graziano,V., Lee,P.L. and Sweet,R.M. (1993) Crystal structure of globular domain of histone H5 and its implications for nucleosome binding. *Nature*, **362**, 219.

Rigoutsos,I. and Floratos,A. (1998) Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, **14**, 56–67.

Schwartz,R.M. and Dayhoff,M.O. (1978) Matrices for detecting distant relationships. In Dayhoff,M.O. (ed.), *Atlas of Protein Sequence and Structure*, pp. 353–358.

Shapiro,L. and Scherer,P.E. (1998) The crystal structure of a complement-lq family protein suggests an evolutionary link to tumor necrosis factor. *Curr. Biol.*, **8**, 335–338.

Stolovitzky,G. and Califano,A. (1999) Pattern statistics in biological datasets. *J. Comp. Biol.* available at www.research.ibm.com/topics/popus/deep/math/html/statistics.pdf.

Wang,L. and Jiang,T. (1994) On the complexity of multiple sequence alignment. *J. Comp. Biol.*, **1**, 337–338.

Waterman,M.S. (1995) *Introduction to Computational Biology*. Chapman & Hall, London.