Discovering Most Classificatory Patterns for Very Expressive Pattern Classes

Masayuki Takeda[1,2], Shunsuke Inenaga[1,2], Hideo Bannai[3], Ayumi Shinohara[1,2], and Setsuo Arikawa[1]

[1] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
[2] PRESTO, Japan Science and Technology Corporation (JST)
{takeda, s-ine, ayumi, arikawa}@i.kyushu-u.ac.jp
[3] Human Genome Center, University of Tokyo, Tokyo 108-8639, Japan
bannai@ims.u-tokyo.ac.jp

Abstract. The classificatory power of a pattern is measured by how well it separates two given sets of strings. This paper gives practical algorithms to find the *fixed/variable-length-don't-care pattern* (*FVLDC pattern*) and *approximate FVLDC pattern* which are most classificatory for two given string sets. We also present algorithms to discover the best *window-accumulated FVLDC pattern* and *window-accumulated approximate FVLDC pattern*. All of our new algorithms run in practical amount of time by means of suitable pruning heuristics and fast pattern matching techniques.

1 Introduction

String pattern discovery centers in the recent trend of knowledge discovery from computational datasets, since such data are usually stored as strings. Especially, the optimization problem of finding a pattern that most frequently appears in the set of *positive examples* and least frequently in the set of *negative examples*, is of great importance. To obtain useful knowledge from given datasets, we began with the possibly most basic and simple pattern class, the *substring pattern class* (as known to be the work of BONSAI [8]), for which the problem can be solved in linear time [3]. In many applications, however, it is necessary to consider a more flexible and *expressive* pattern class, for example in the field of bioinformatics, since biological functions are retained between sequences even if they are slightly different. In fact, the function may be dependent on two regions which are some distance apart in the sequence, but are close in the three dimensional structure that the sequence assumes. To this end, we considered the *subsequence pattern class* [3], and then the *variable-length-don't-care pattern class* (*VLDC pattern class*) [5]. An example of a VLDC pattern is $\star a \star abb \star$ with $a, b \in \Sigma$, where the variable length don't care symbol $\star$ matches any string. The VLDC pattern class is a generalization of the substring and subsequence pattern classes.

In this paper, we further consider mismatches on constant segments of VLDC patters. Firstly, we consider replacing a character with a *fixed* length don't care symbol $\circ$ that matches any *single* character. It yields a pattern such as $\star a \star$

$a \circ b\star$ for the running example. Such a pattern is called a *fixed/variable-length-don't-care pattern* (*FVLDC pattern*), and its class is named the *FVLDC pattern class*. Pursuing a pattern class of more expressive power, we secondly apply the *approximate matching* measure to constant segments of FVLDC patterns. An *approximate FVLDC pattern* is a pair $\langle q, k \rangle$ where q is an FVLDC pattern and k is a threshold for the number of mismatches in the constant segments of q.

The *approximate FVLDC pattern class* no doubt has a great expressive power, but in the meantime, it includes quite many patterns without a *classificatory power* in the sense of being too general and matching even most negative examples. Typically, an approximate FVLDC pattern could match almost all *long* texts over a *small* alphabet. The same problem happens to the subsequence pattern class for the first place, but its window-accumulated version called the *episode pattern class* has overcome this difficulty [4]. This paper considers the *window-accumulated FVLDC pattern class* as well as the *window-accumulated approximate FVLDC pattern class*. We address that not only do they possess a remarkable expressive power, but also include many patterns with a very good classificatory power.

The main result of this paper consists in new practical algorithms to find a best pattern that separates two given string datasets, for all of the FVLDC pattern class, approximate FVLDC pattern class, window-accumulated FVLDC pattern class, and window-accumulated approximate FVLDC pattern class. Each algorithm runs in reasonable amount of time, due to the benefit of suitable pruning heuristics and pattern matching techniques. Interested readers are guided into reading our previous work [3–5] for the overall idea of our project, and technical report [9] for more detail of this work.

2 Preliminaries

Let $\mathcal{N}$ be the set of non-negative integers. Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The *length* of a string w is the number of characters in w and denoted by $|w|$. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. Strings x, y, and z are said to be a *prefix*, *substring*, and *suffix* of string $w = xyz$, respectively. The substring of a string w that begins at position i and ends at position j is denoted by $w[i : j]$ for $1 \leq i \leq j \leq |w|$. For convenience, let $w[i : j] = \varepsilon$ for $j < i$. The *reversal* of a string w is denoted by w^R. For a set $S \subseteq \Sigma^*$ of strings, the number of strings in S is denoted by $|S|$ and the total length of strings in S is denoted by $\|S\|$.

Let $\star$ be a special symbol called the *variable length don't care* matching any string in Σ^*. A string over $\Sigma \cup \{\star\}$ is a *variable-length-don't-care pattern* (*VLDC pattern* in short). For example, $\star a \star ab \star ba \star$ is a VLDC pattern with $a, b \in \Sigma$. We say a VLDC pattern q *matches* a string w if w can be obtained by replacing $\star$'s in q with some strings. In the running example, the VLDC pattern $\star a \star ab \star ba \star$ matches string *abababbbaa* with the $\star$'s replaced by *ab*, *b*, *b* and *a*, respectively. The *size* of p, denoted by *size*(p), is the length of p excluding all $\star$'s. Thus,

$size(p) = 5$ and $|p| = 9$ for $p = \star a \star ab \star ba \star$. We remark that $size(p)$ is the minimum length of the strings p matches.

A *pattern class* over Σ is a pair (Π, L) consisting of a set Π of descriptions over some finite alphabet, called *patterns*, and a function L that maps a pattern $\pi \in \Pi$ to its language $L(\pi) \subseteq \Sigma^*$. A pattern $\pi \in \Pi$ is said to *match* a string $w \in \Sigma^*$ if w belongs to the language $L(\pi)$.

Let *good* be a function from $\Pi \times 2^{\Sigma^*} \times 2^{\Sigma^*}$ to the real numbers. The problem we consider is: *Given two sets $S, T \subseteq \Sigma^*$ of strings, find a pattern $\pi \in \Pi$ that maximizes the score $good(\pi, S, T)$.* Intuitively, the score $good(\pi, S, T)$ expresses the "goodness" of π in the sense of distinguishing S from T. The definition of *good* varies with applications. For example, the χ^2 values, entropy information gain, and Gini index are often used. Essentially, these statistical measures are defined by the number of strings that satisfy the rule specified by π. Any of the above-mentioned measures can be expressed by the following form:

$$good(\pi, S, T) = f(x_\pi, y_\pi, |S|, |T|),$$
$$\text{where } x_\pi = |S \cap L(\pi)| \text{ and } y_\pi = |T \cap L(\pi)|.$$

When S and T are fixed, $x_{\max} = |S|$ and $y_{\max} = |T|$ are regarded as constants. On this assumption, we abbreviate the notation of the function to $f(x, y)$. In the sequel, we assume that f is *conic* [3–5] and can be evaluated in constant time. Let $F(x, y) = \max\{f(x, y), f(x, 0), f(0, y), f(0, 0)\}$. The following lemma derives from the conicality of function f, on which our pruning heuristics are based.

Lemma 1 ([3]). *For any $(x, y), (x', y') \in [0, x_{\max}] \times [0, y_{\max}]$, if $x \leq x'$ and $y \leq y'$, then $f(x, y) \leq F(x', y')$.*

3 Allowing Mismatches by Don't Cares

A VLDC pattern requires that all of its constant segments occur within a string in the specified order with variable-length gaps. To obtain a more expressive pattern class, we first introduce the don't care symbol $\circ$ that matches *any single* character of Σ. We have two purposes: One is to allow mismatches in the constant segments of a VLDC pattern. The other is to realize a variety of gap symbols. An s-times repetition of $\circ$ works as a fixed-length don't care that matches any string of length s over Σ. Also, a VLDC $\star$ followed by an s-times repetition of $\circ$ expresses a *lower-bounded-length gap symbol* that matches any string of length at least s. A string in $\Pi = (\Sigma \cup \{\star, \circ\})^*$ is called a *fixed/variable-length don't-care pattern* (an *FVLDC pattern*), and (Π, L) is the *FVLDC pattern class*. The length and the size of an FVLDC pattern are defined similarly to those of a VLDC pattern, regarding $\circ$ as a character of Σ.

Definition 1 (Finding best FVLDC pattern according to f).
Input: *Two sets $S, T \subseteq \Sigma^*$ of strings.*
Output: *An FVLDC pattern $p \in \Pi$ that maximizes the score $f(x_p, y_p)$, where $x_p = |S \cap L(p)|$ and $y_p = |T \cap L(p)|$.*

Note that any FVLDC pattern p of size greater than ℓ matches no strings in $S \cup T$, and thus we have $x_p = 0$ and $y_p = 0$. The possible maximum length of p is $2size(p) + 1$ since we ignore the patterns with two or more consecutive $\star$'s, and therefore we can restrict by $2\ell+1$ the length of the patterns to be examined against S and T.

Lemma 2 (Search space for best FVLDC pattern). *The best FVLDC pattern for S and T can be found in $\Pi^{(\ell)} = \{p \in \Pi \mid |p| \leq 2\ell + 1\}$, where ℓ is the maximum length of the strings in $S \cup T$.*

We can prune the search space according to the following lemma.

Lemma 3 (Pruning lemma for FVLDC pattern). *Let p be any FVLDC pattern in Π. Then, $f(x_{pq}, y_{pq}) \leq F(x_{p\star}, y_{p\star})$ for every FVLDC pattern $q \in \Pi$.*

The following is a sub-problem of Definition 1, which should be solved quickly.

Definition 2 (Counting matched FVLDC patterns).
Input: *A set $S \subseteq \Sigma^*$ of strings and an FVLDC pattern $p \in \Pi$.*
Output: *The cardinality of the set $S \cap L(p)$.*

The minimum DFA that accepts $L(p)$ for an FVLDC pattern p has an exponential number of states. However, there is a nondeterministic finite automaton (NFA) with only $m + 1$ states that accepts the same language. As a practical solution, we adopt the bit-parallel simulation to this NFA. We use $(m + 1)$-bit integers to simulate in constant time the state transitions in parallel for each character of the input strings. When $m + 1$ is not greater than the computer word length, say 32 or 64, the algorithm runs in $O(\|S\|)$ time after $O(|p||\Sigma|)$-time preprocessing for p.

4 Finding Best Approximate FVLDC Patterns

Let (Π, L) be the FVLDC pattern class, and let $\delta : \Sigma^* \times \Sigma^* \to \mathcal{N} \cup \{\infty\}$ be the well-known Hamming distance measure [2]. A pattern $p \in \Pi$ is said to *approximately match a string w within a distance k* if there is a string $w' \in L(p)$ such that $\delta(w, w') \leq k$. For any $\langle p, k\rangle \in \Pi \times \mathcal{N}$, let

$$L_\delta(\langle p, k\rangle) = \{w \in \Sigma^* \mid \exists w' \in L(p) \text{ such that } \delta(w, w') \leq k\}.$$

Then, the pair $(\Pi \times \mathcal{N}, L_\delta)$ is a new pattern class derived from (Π, L) with δ. Let us call the elements of $\Pi \times \mathcal{N}$ the *approximate FVLDC patterns*. For a fixed $k \in \mathcal{N}$, an ordered pair $\langle p, k\rangle$ with $p \in \Pi$ is called a *k-approximate FVLDC pattern*.

Definition 3 (Finding best approximate FVLDC pattern according to f).
Input: *Two sets $S, T \subseteq \Sigma^*$ of strings, and a non-negative integer $k_{\max}$.*
Output: *An approximate FVLDC pattern $\pi = \langle q, k\rangle \in \Pi \times [0, k_{\max}]$ that maximizes the score $f(x_\pi, y_\pi)$, where $x_\pi = |S \cap L_\delta(\pi)|$ and $y_\pi = |T \cap L_\delta(\pi)|$.*

We here have to find the best combination of a pattern q and an error level k.

Lemma 4 (Search space for best approximate FVLDC pattern). *The best approximate FVLDC pattern for $S, T \subseteq \Sigma^*$ and $k_{\max}$ can be found in $\Pi^{(\ell)} \times [0, k_{\max}]$, where $\Pi^{(\ell)}$ is the same as in Lemma 2.*

We have two pruning techniques for the problem of Definition 3. The first one is as follows.

Definition 4 (Computing best error level according to f).
Input: *Two sets $S, T \subseteq \Sigma^*$ of strings, an FVLDC pattern $q \in \Pi$, and a non-negative integer $k_{\max}$.*
Output: *An integer $k \in [0, k_{\max}]$ that maximizes the score $f(x_\pi, y_\pi)$ for $\pi = \langle q, k \rangle$, where $x_\pi = |S \cap L_\delta(\pi)|$ and $y_\pi = |T \cap L_\delta(\pi)|$.*

For an FVLDC pattern $q \in \Pi$ and string $u \in \Sigma^*$, we define the *distance* between q and u by $Dist_\delta(q, u) = \min\{\delta(w, u) \mid w \in L(q)\}$. If there is no such w, let $Dist(q, u) = \infty$. For $q \in \Pi$ and $S \subseteq \Sigma^*$, let $\Delta(q, S) = \{Dist_\delta(q, u) \mid u \in S\}$. Then, the best error level of a pattern $q \in \Pi$ for given $S, T \subseteq \Sigma^*$ can be found in the set $\Delta(q, S \cup T) \cap [0, k_{\max}]$.

Lemma 5. *For an FVLDC pattern $q \in \Pi$ and string $w \in \Sigma^*$, $Dist_\delta(q, w)$ can be computed in $O(|q||w|)$ time.*

Proof. Directly from the results of Myers and Miller [6], in which regular expressions are treated instead of FVLDC patterns. □

Lemma 6 (Pruning lemma 1 for approximate FVLDC pattern). *Let p be any FVLDC pattern in Π and $\pi = \langle p, k_{\max} \rangle$. Then, $f(x_\tau, y_\tau) \leq F(x_\pi, y_\pi)$ for every approximate FVLDC pattern $\tau = \langle pq, k \rangle$ such that $q \in \Pi$ and $k \in [0, k_{\max}]$.*

The second approach for pruning the search space is quite simple. We repeatedly execute a procedure that finds the best k-approximate FVLDC pattern $\langle p, k \rangle$ for S and T, in increasing order of $k = 0, 1, \ldots, k_{\max}$. It is possible to prune the search space $\Pi^{(\ell)} \times \{k\}$ by:

Lemma 7 (Pruning lemma 2 for approximate FVLDC pattern). *Let $k \in [0, k_{\max}]$, p be any FVLDC pattern in Π, and $\pi = \langle p, k \rangle$. Then, $f(x_\tau, y_\tau) \leq F(x_\pi, y_\pi)$ for every k-approximate FVLDC pattern $\tau = \langle pq, k \rangle$ with $q \in \Pi$.*

The following is a sub-problem of Definition 3, which should be solved quickly.

Definition 5 (Counting matched approximate FVLDC patterns).
Input: *A set $S \subseteq \Sigma^*$ of strings and an approximate FVLDC pattern $\langle p, k \rangle$.*
Output: *The cardinality of the set $S \cap L_\delta(\langle p, k \rangle)$.*

We developed an efficient algorithm to solve the above sub-problem. Although we omit the detail due to lack of space, it performs the diagonal-wise bit-parallel simulation (see [7]) of an NFA that recognizes the language $L_\delta(\langle p, k \rangle)$ of an approximate FVLDC pattern $\langle p, k \rangle$. The NFA being simulated has $(m+1)(k+1)$

states ($m = size(p)$), but $(m-k+1)(k+1)$-bits are enough. If the $(m-k+1)(k+1)$-bit representation fits in a single computer word, it runs in linear time in $\|S\|$ after $O(|q||\Sigma|)$-time preprocessing of $\langle p, k\rangle$. The algorithm was inspired by the work of Baeza-Yates and Navarro [1], which aims an approximate substring pattern matching where the Levenshtein distance is used as a distance measure δ, not the Hamming distance. Although their algorithm could be easily extended to the approximate FVLDC pattern matching if the Levenshtein distance measure was used, a new development is actually necessary to cope with the Hamming distance.

5 Extension to Window-Accumulated Patterns

For any pattern class $\langle \Pi, L\rangle$, we introduce a *window* whose size (width) limits the length of a pattern occurrence within a string. A pattern $p \in \Pi$ is said to *occur in a string w within a window of size h* if w has a substring of length at most h the pattern p matches. For any pair $\langle p, h\rangle$ in $\Pi \times \mathcal{N}$, let

$$\hat{L}(\langle p, h\rangle) = \{w \in \Sigma^* \mid p \text{ occurs in } w \text{ within a window of size } h\},$$

and let $\hat{L}(\langle p, \infty\rangle) = L(p)$ for convenience. The pair $(\Pi \times \mathcal{N}, \hat{L})$ is a new pattern class derived from (Π, L). We call the elements of $\Pi \times \mathcal{N}$ the *window-accumulated patterns* for (Π, L).

Definition 6 (Finding the best window-accumulated pattern in (Π, L) according to f).
Input: *Two sets $S, T \subseteq \Sigma^*$ of strings.*
Output: *A window-accumulated pattern $\pi = \langle q, h\rangle \in \Pi \times \mathcal{N}$ that maximizes the score $f(x_\pi, y_\pi)$, where $x_\pi = |S \cap \hat{L}(\pi)|$ and $y_\pi = |T \cap \hat{L}(\pi)|$.*

We stress h is *not* given beforehand, and hence we have to find the best combination of a pattern q and window width h. The search space is thus $\Pi \times \mathcal{N}$, not Π.

The following is a sub-problem of Definition 6, which should be solved quickly.

Definition 7 (Computing best window size for (Π, L) according to f).
Input: *Two sets $S, T \subseteq \Sigma^*$ of strings and a pattern $q \in \Pi$.*
Output: *An integer $h \in \mathcal{N}$ that maximizes the score $f(x_{\langle q,h\rangle}, y_{\langle q,h\rangle})$, where $x_{\langle q,h\rangle} = |S \cap \hat{L}(\langle q, h\rangle)|$ and $y_{\langle q,h\rangle} = |T \cap \hat{L}(\langle q, h\rangle)|$.*

For a pattern $q \in \Pi$ and for a string $u \in \Sigma^*$, we define the *minimum window size* θ of q for u by $\theta_{q,u} = \min\{h \in \mathcal{N} \mid u \in \hat{L}(\langle q, h\rangle)\}$. If there is no such value, let $\theta_{q,u} = \infty$. For any $q \in \Pi$ and any $S \subseteq \Sigma^*$, let $\Theta(q, S) = \{\theta_{q,u} \mid u \in S\}$. The best window size of $q \in \Pi$ for $S, T \subseteq \Sigma^*$ can be found in $\Theta(q, S \cup T)$.

Lemma 8 (Search space for best window-accumulated pattern). *The best window-accumulated pattern in (Π, L) for S and T can be found in $\{\langle q, h\rangle \mid q \in \Pi$ and $h \in \Theta(q, S \cup T)\}$.*

We emphasize that the above discussion holds for the window-accumulated version of *any* pattern class (Π, L). However, the complexity of computing the minimum window size depends on (Π, L).

5.1 Window-Accumulated FVLDC Patterns

This section is devoted to finding the best window-accumulated FVLDC pattern from two given sets S, T of strings.

Lemma 9 (Search space for best window-accumulated FVLDC pattern). *The best window-accumulated FVLDC pattern for S and T can be found in $\{\langle q, h\rangle \mid q \in \Pi^{(\ell)}$ and $h \in \Theta(q, S \cup T)\}$, where $\Pi^{(\ell)}$ is the same as in Lemma 2.*

Lemma 10 (Pruning lemma for window-accumulated FVLDC pattern). *Let p be an FVLDC pattern and $\pi = \langle p, \infty\rangle$. Then, $f(x_\tau, y_\tau) \leq F(x_\pi, y_\pi)$ for every window-accumulated FVLDC pattern $\tau = \langle pq, h\rangle$ such that $q \in \Pi$ and $h \in \mathcal{N}$.*

Lemma 11. *The minimum window size $\theta_{q,w}$ of an FVLDC pattern q for a string $w \in \Sigma^*$ can be computed in $O(|q||w|)$ time.*

Proof. By a standard dynamic programming approach. □

The dynamic programming method is, however, relatively slow in practice. An alternative way to solve the problem is to build from a given FVLDC pattern q two NFAs accepting $L(q)$ and $L(q^R)$, which we call the *forward* and *backward* NFAs, respectively. An occurrence of a pattern q that starts at position i and ends at position j of a string w, denoted by (i, j), is said to be *minimal* if no proper substring of $w[i : j]$ contains an occurrence of q. Let $(i_1, j_1), \cdots, (i_r, j_r)$ be the sequence of the minimal occurrences of q in w satisfying $i_1 < \cdots < i_r$. We run the forward NFA over w starting at the first character to determine the value j_1. When j_1 is found, we use the backward NFA going backward starting at the j_1-th character in order to determine the value i_1. After i_1 is determined, we again use the forward NFA going forward starting at the $(i_1 + 1)$-th character for finding the value j_2. Continuing in this fashion, we can determine all of the minimal occurrences of q in w. The minimal window size is obtained as the minimum among the widths of the minimal occurrences. We simulate the two NFAs over a given string basing on the bit-parallelism mentioned in Section 3 when $size(q)+1$ does not exceed the computer word length. Although the running time of this method is $O(r|w|) = O(|w|^2)$ in the worst case, it shows a good performance compared with the above-mentioned dynamic programming based method, since the number r of minimal occurrences of q is not so large in reality.

5.2 Window-Accumulated Approximate FVLDC Patterns

The search space for the best window-accumulated approximate FVLDC pattern is $\Pi \times [0, k_{\max}] \times \mathcal{N}$. A reasonable approach would be to compute the best pattern in $\Pi \times \{k\} \times \mathcal{N}$ for each $k = 0, 1, \ldots, k_{\max}$, and then choose the best one among them. We have only to consider finding the best window-accumulated k-approximate FVLDC pattern for a fixed k.

Lemma 12 (Search space for best window-accumulated k-approximate FVLDC pattern). *Let k be a fixed non-negative integer. The best window-accumulated k-approximate FVLDC pattern for S, T can be found in $\{\langle\langle q, k\rangle, h\rangle \mid q \in \Pi^{(\ell)}$ and $h \in \Theta(\langle q, k\rangle, S \cup T)\}$, where $\Pi^{(\ell)}$ is the same as in Lemma 2.*

Lemma 13 (Pruning lemma for window-accumulated k-approximate FVLDC pattern). *Let k be a fixed non-negative integer. Let p be an FVLDC pattern, and let $\pi = \langle\langle p, k\rangle, \infty\rangle$. Then, $f(x_\tau, y_\tau) \leq F(x_\pi, y_\pi)$ for every window-accumulated k-approximate FVLDC pattern $\tau = \langle\langle pq, k\rangle, h\rangle$ such that $q \in \Pi$ and $h \in \mathcal{N}$.*

Lemma 14. *Let k be a fixed non-negative integer. The minimum window size $\theta_{q,w}$ of a k-approximate FVLDC pattern $\langle q, k\rangle$ for a string $w \in \Sigma^*$ can be computed in $O(k|q||w|)$ time.*

Proof. A straightforward extension of the dynamic programming method for Lemma 11. □

In practice, we again adopt the two NFAs based approach. It is possible to simulate the NFA for an approximate FVLDC pattern $\langle q, k\rangle$ over a string w using $(m - k + 1)(k + 1)$-bit integers in linear time, where $m = size(q)$. The running time is therefore $O(r|w|)$ if $(m - k + 1)(k + 1)$ does not exceed the computer word length.

References

1. R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2):127–158, 1999.
2. D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York, 1997.
3. M. Hirao, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. A practical algorithm to find the best subsequence patterns. In *Proc. Discovery Science 2000*, volume 1967 of *LNAI*, pages 141–154. Springer-Verlag, 2000.
4. M. Hirao, S. Inenaga, A. Shinohara, M. Takeda, and S. Arikawa. A practical algorithm to find the best episode patterns. In *Proc. Discovery Science 2001*, volume 2226 of *LNAI*, pages 435–440. Springer-Verlag, 2001.
5. S. Inenaga, H. Bannai, A. Shinohara, M. Takeda, and S. Arikawa. Discovering best variable-length-don't-care patterns. In *Proc. Discovery Science 2002*, volume 2534 of *LNCS*, pages 86–97. Springer-Verlag, 2002.
6. E. W. Myers and W. Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.
7. G. Navarro and M. Raffinot. *Flexible pattern matching in strings: Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, Cambridge, 2002.
8. S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa. Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Trans. of Information Processing Society of Japan*, 35(10):2009–2018, 1994.
9. M. Takeda, S. Inenaga, H. Bannai, A. Shinohara, and S. Arikawa. Discovering most classificatory patterns for very expressive pattern classes. Technical Report DOI-TR-CS-219, Department of Informatics, Kyushu University, 2003.