2003−AL−91　(3)
2003／9／19

最適 degenerate pattern 探索アルゴリズムと転写因子結合部位同定問題への適用

篠崎大輔*　阿久津達也†　丸山修‡

概要:有限アルファベット Σ 上の degenerate pattern とは Σ の部分集合からなるシーケンスである．Σ 上の正例文字列の集合Ｐと負例文字列の集合Ｑが与えられたとき，Ｐの全ての文字列にマッチし，Ｑのどの文字列にもマッチしない Σ 上の degenerate pattern の存在を判定する問題は，NP 完全であることが知られている．本稿では，多くの自然なスコア関数に対し適用可能な枝刈探索法を用いて，最適な degenerate pattern を探索するアルゴリズムを提案する．そして，このアルゴリズムを転写因子の結合部位を同定する問題に適用し，いくつかの遺伝子集合に対して計算機実験を行った結果を報告する．

Algorithm for finding optimal degenerate patterns with an application to transcription factor binding sites consensus identification.

Daisuke Shinozaki*　Tatsuya Akutsu†　Osamu Maruyama‡

abstract: A degenerate pattern over a finite alphabet Σ is a sequence of subsets of Σ. For a degenerate pattern over Σ, it is known that the problem of finding a degenerate pattern consistent with both the sets of positive strings over Σ and that of negative ones is in general NP-complete. In this work, we have proposed a heuristic algorithm for finding optimal degenerate patterns with a pruning technique, which works on most all reasonable score functions. Then we have applied this algorithm to the problem of identifying transcription factor binding sites in the upstream regions of given genes, and reported the results from the computational experiments on several gene sets.

1 Introduction

The problem of finding transcription factor binding sites in the upstream regions of given genes is algorithmically an interesting and challenging problem in computational biology. Such a site is an essential factor in the mechanism of transcriptional regulation. It is known that the binding sites of a transcription factor are often approximately conserved across the upstream regions of the co-regulated genes. Thus, the problem of finding regulatory signals can be reduced to the search problem for convincing patterns common to almost all of the given DNA sequences.

The major types of patterns modeling transcription factor binding sites can be classified into the following three groups: (i) a weight matrix, (ii) a string over nucleic acids, $\{A, C, G, T\}$, possibly with some mismatches, and (iii) a string over IUPAC nucleic acid codes. All of these patterns are often called *motifs*.

A weight matrix for a motif of length l is a $4 \times l$ matrix, whose rows and columns correspond to the 4 bases and the positions in the motif, respectively. The weight in the i-th row and j-th column means the frequency with which the i-th base is found in the j-th position of the motif. Thus a weight matrix is a very flexible expression of motifs. A weight matrix is used in a number of probabilistic approach algorithms for finding motifs, including the expectation maximization algorithm [10], MEME [1], Gibbs sampler [9] and CONSENSUS[7]. These heuristic algorithms are based on local search, which implies that in general, there is no theoretical guarantee that the optimal solutions are always found by these tools. However, local search-based motif finders have seen substantial success in practice.

The simplest case of the pattern models (ii) and (iii) is just l-mers *without* any mismatches nor mutations. [22] applied an enumerative approach for finding statistically significant l-mers. [15] considered ways of finding l-

* 九州大学大学院数理学府
Graduate School of Mathematics, Kyushu University
† 京都大学化学研究所バイオインフォマティクスセンター
Bioinformatics Center, Institute for Chemical Research, Kyoto University
‡ 九州大学大学院数理学研究院
Faculty of Mathematics, Kyushu University

mer with at most k mismatches common to the given set of DNA sequences. In their work, they proposed two novel enumerative algorithms called WINNOWER and SP-STAR, to find significant patterns over $\{A, C, G, T\}$ of length l which are allowed to have at most d mismatches, which we call (l, d)-motif pattern. Note that an (l, d)-motif pattern is *not* position-specific since at most d mutations are allowed to occur at every positions of the occurrences of the patterns. For the same problem, [4] devised a novel method which combines both random projection and local search. Various decision problems for finding (l, d)-motif patterns, known as Closest Substring Problem, Max Close String Probelm, Distinguishing String Selection Problem, etc, have been discussed in [8, 6, 5] and references therein from the viewpoint of computational complexity. Those problems are shown to be generally NP-complete.

On the other hand, the patterns over the IUPAC nucleic acid codes, which we call *degenerate patterns* over $\Sigma = \{A, C, G, T\}$ in this work, are position-specific. The problem of finding a degenerate pattern consistent with both all of the positive strings and all of the negative strings is a special case of the problem of Best Consensus Motif discussed in [21], which is shown to be NP-complete. In [19] and [18], enumerative algorithms for finding optimal degenerate patterns over the codes, A,C,G,T,R,Y,S,W and N have been presented. These patterns are evaluated by z-score and p-value, respectively. Note that R,Y,S,W mean two symbols of the basic 4 bases, and N means the 4 bases. Their patterns are in the form of the strings s over the 9 symbols whose central part is a spacer consisting of N's of length up to about 11.

Our aim in this paper is to devise a heuristic algorithm to find a degenerate pattern which is *optimal* for positive and negative string sets w.r.t. a *given* score function. We then propose a branch-and-bound algorithm, called SUPERPOSITION, which works for arbitrary score functions which are *conic*, as defined in [17]. It should be noted here that most all reasonable score functions, for example, χ^2 values, entropy information gain and gini index, are conic.

To construct SUPERPOSITION, we have introduced an operation which generates a new degenerate pattern from existing two patterns $p = p_1 p_2 \cdots p_l$ and $q = q_1 q_2 \cdots q_l$, called the *superposition* of p and q, that is defined as a degenerate pattern $r = r_1 r_2 \cdots r_l$ with $r_i = p_i \cup q_i$ for $i = 1, 2, \ldots, l$. SUPERPOSITION is a sample-driven approach, that is, it first extracts all of the substrings of a specified length l from the positive strings. After this extraction phase, SUPERPOSITION goes into the superposition phase, in which a superposed pattern r, generated from existing ones, will be eliminated from this phase forever if the possible optimal score of any superpositions between r and others is less than the intermediate optimal score of all of the patterns searched so far in the process.

Finally, we will present the successful results of finding binding sites in the upstream regions of co-regulated genes of yeast, and report the performance comparison between SUPERPOSITION, YMF[19] with an auxiliary tool FindExplanators[3], MEME[1] and AlignACE[16]. In our computational experiments, several known transcription factor binding sites in a promoter database of the yeast Saccharomyces cerevisiae (SCPD) [23] have been detected by SUPERPOSITION more accurately than the others.

This paper is organized as follows. In Section 2, preliminaries are given. Section 3 shows our heuristic algorithm for the problem. In the last section, we review our computational experiments on real data of yeast.

2 Preliminaries

Let Σ be a finite alphabet, and let Σ^* be the set of all *strings* over Σ. For a non-negative integer l, by Σ^l we denote the strings of length l over Σ. A complementary map over Σ is a mapping c from Σ to Σ such that $c(c(x)) = x$ for each symbol $x \in \Sigma$. The *complement* of a string $s = s_1 s_2 \cdots s_l$ over Σ w.r.t. c, denoted by s^c, $s = c(s_l) c(s_{l-1}) \cdots c(s_1)$. In the problem of finding regulatory signals in DNA sequences, Σ and c can be set to be $\Sigma = \{A, C, G, T\}$ and $c(A) = T$, $c(T) = A$, $c(C) = G$, and $c(G) = T$.

Definition 1 A *degenerate pattern* over a finite alphabet Σ is defined as a sequence $p = p_1 p_2 \ldots p_l$ where p_i is a subset of Σ for $1 \leq i \leq l$. The length of p is defined as l.

For these degenerate patterns $p = p_1 p_2 \ldots p_l$, we can consider the following distinctive pattern matchers which are defined as binary functions. Let t be a string over Σ.

$m_{\text{basic}}(p, t)$ returns `true` if there is at least one occurrence of p in t, i.e., there exists a substring $t_i \cdots t_{i+l-1}$ of t such that $t_{i+k-1} \in p_k$ for $1 \leq k \leq l$, and `false` otherwise.

$m_c(p, t)$ returns `true` if either $m_{\text{basic}}(p, t)$ or $m_{\text{basic}}(p, t^c)$ is `true`, and `false` otherwise, for a complementary map c.

For a degenerate pattern p, a pattern matcher m, and a set S of strings, we denote by $L(p, S, m)$ the subset of S whose elements are matched by p with m, i.e., $L(p, S, m) = \{t \in S \mid m(p, t) = \texttt{true}\}$. When m

is clear from the context, we omit m and write $L(p, S)$ instead.

We introduce the concept of the ambiguity of degenerate patterns. The *degeneracy* of a degenerate pattern $p = p_1 p_2 \cdots p_l$, denoted by $degen(p)$, is defined as the value of $\prod_{i=1}^{l} |p_i|$. For example, for $p = A\{C,G\}\{A,G,T\}\{A,C\}CC$, we have $degen(p) = 12$. Note that for a degenerate pattern p of length l, $degen(p) = |L(p, \Sigma^l, m_{\text{basic}})|$.

3 Heuristic Approach

We here present a heuristic approach for finding all of the top K optimal patterns, based on a pruning technique, for an arbitrary constant K.

For degenerate patterns p and q and a pattern matcher m, if $L(p, \Sigma^*, m) \subseteq L(q, \Sigma^*, m)$ then we say that p is *more specific* than q and also that q is *more general* than p. Here we define an operation on two degenerate patterns p and q, which is designed in order to generate a new degenerate pattern r.

Definition 2 For two degenerate patterns $p = p_1 p_2 \cdots p_l$ and $q = q_1 q_2 \cdots q_l$ with $p_i, q_i \subseteq \Sigma$ for $1 \leq i \leq l$, the *superposition* of p and q, denoted by $superpose(p, q)$, is a degenerate pattern $r = r_1 r_2 \cdots r_l$ such that $r_i = p_i \cup q_i$ for $1 \leq i \leq l$.

Note that vany of the degenerate patterns $r = r_1 r_2 \cdots r_l$ obtained by recursively superposing a degenerate pattern $p = p_1 p_2 \cdots p_l$ and arbitrary degenerate patterns is more general than p w.r.t. appropriate pattern matchers since $r_i \supseteq p_i$ for $i = 1, 2, \ldots, l$.

3.1 Pruning Heuristics

[17] gave the definition of a *conic* function and showed how to use it for optimal pattern-finding algorithms enumerating patterns from general to specific, for the classes of substring patterns, subsequence patterns and episode patterns.

Definition 3 ([17]) A function $f : [0, x_{max}] \times [0, y_{max}]$ to real numbers is said to be *conic* if

for any $0 \leq y \leq y_{max}$, there exists an x_0 such that

- $f(x_l, y) \geq f(x_r, y)$ for any $0 \leq x_l < x_r \leq x_0$,
- and $f(x_l, y) \leq f(x_r, y)$ for any $x_0 \leq x_l < x_r \leq x_{max}$,

and for any $0 \leq x \leq x_{max}$, there exists an y_0 such that

- $f(x, y_l) \geq f(x, y_r)$ for any $0 \leq y_l < y_r \leq y_0$,
- and $f(x, y_l) \leq f(x, y_r)$ for any $y_0 \leq y_l < y_r \leq y_{max}$.

Lemma 1 Let f be a conic function. Let P and Q be sets of strings. If S and T, sets of strings, satisfy $S \subseteq T$, we have $f(|P \cap T|, |Q \cap T|) \leq \max\{f(|P \cap S|, |Q \cap S|), f(|P|, |Q \cap S|), f(|P \cap S|, |Q|), f(|P|, |Q|)\}$.

The proof is not hard because it is almost the same as the proof of Lemma 2 in [17]. For short, we denote $superpose(\cdots superpose(superpose(p, q_1), q_2), \ldots, q_n)$ by $superpose(p, q_1, q_2, \ldots, q_n)$.

Corollary 1 Let f be a conic function, and let P and Q be sets of strings. For any degenerate patterns p and $q_1, q_2, \ldots, q_n$, let $r = superpose(p, q_1, q_2, \ldots, q_n)$. For a degenerate pattern x, we denote $\text{tp}_x = |L(x, P, m_{\text{basic}})|$ and $\text{fp}_x = |L(x, Q, m_{\text{basic}})|$. Then we have $f(\text{tp}_r, \text{fp}_r) \leq f_{\text{upper\_bound}}(\text{tp}_p, \text{fp}_p, P, Q)$ where $f_{\text{upper\_bound}}(\text{tp}_p, \text{fp}_p, P, Q) = \max\{f(\text{tp}_p, \text{fp}_p), f(|P|, \text{fp}_p), f(\text{tp}_p, |Q|), f(|P|, |Q|)\}$ (see Fig. 1).

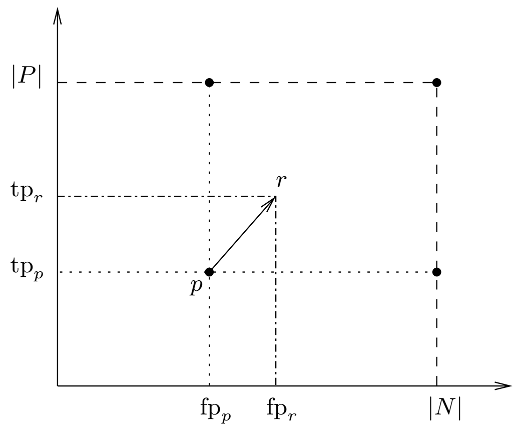

Figure 1: The four small black points correspond to the four values of $f_{\text{upper\_bound}}(\text{tp}_p, \text{fp}_p, P, Q)$, respectively. The degenerate pattern r obtained by recursively superposing p and others is always located within the square with the four points.

Note that a score function is supposed to be a maximization function. Corollary 1 is a key idea of our heuristic enumerative algorithm, called SUPERPOSITION, for finding optimal degenerate patterns for the positive and negative set w.r.t. a particular conic score function.

First, SUPERPOSITION extracts all of the substrings of a specified length l from the positive strings, and then goes into the superposition phase, in which new patterns are generated by superposing existing ones. Therefore, SUPERPOSITION is a sample-driven approach, and patterns are enumerated *from specific to* *general*. In the superposition phase, when a pattern p is used in superposing with another pattern q, if $f_{\text{upper\_bound}}(\text{tp}_p, \text{fp}_p, P, Q)$ is less than the intermediate optimal score, p is discarded and the superposition of p and q is canceled, because from corollary 1, it holds that all of the patterns obtained by recursively superposing p and arbitrary patterns mark at most the value of $f_{\text{upper\_bound}}(\text{tp}_p, \text{fp}_p, P, Q)$ as their scores. The details of SUPERPOSITION are given in Fig. 2.

Let P and Q be sets of strings, and let l be a positive integer. It would be easy to see that for a conic score function f, the algorithm $\text{SUPERPOSITION}(P, Q, l)$ returns a length-l degenerate pattern which is optimal for P and Q w.r.t. f. Notice that for a constant K, it is easy to modify the algorithm to return the best K patterns instead of returning one optimal pattern. It can be realized by using another sorted list for keeping the current best K patterns.

As mentioned before, [17] have considered the problems of finding optimal patterns for the classes of substring patterns, subsequence patterns and episode patterns. Their branch-and-bound algorithm enumerates patterns *from general to specific*, which is a different point with SUPERPOSITION. The reason why SUPERPOSITION enumerates degenerate patterns from specific to general is the practical reason that the degeneracies of many known consensus motifs are nearer to the lowest degeneracy 0 than to the highest degeneracy 4^l. It should be noted here that [2] have recently dealt with the problem called *string pattern regression*, in which, given a set of pairs of a string and a weight, the task is to find the best pattern which is conserved in a subset of the given sequences for which the distribution of weights of the subset is most different from the distribution of weights of the rest. This problem can be considered to be a natural generalization of the case where we are given both a positive and negative string sets, since the weight of a positive (negative, resp.) string could be set to 1 (-1, resp.). They have presented a branch-and-bound algorithm for the problem, based on the algorithm in [17], by devising a way of calculating the upper bound of arbitrary patterns derived from a particular pattern in the situation of string pattern regression.

3.2 Including Complements

In this subsection, we describe how to modify SUPERPOSITION which takes account of the complements of P in addition to P.

We then change the pattern matcher m_{basic} into m_{c}. Recall that the value of $m_{\text{c}}(p, t)$ is the logical sum of $m_{\text{basic}}(p, t)$ and $m_{\text{basic}}(p, t^c)$. In the case of using m_{c}, when we generate new superpositions from existing degenerate patterns $\mathit{pat1}$ and $\mathit{pat2}$, we should consider the 4 possible patterns, that is,

(i) $\mathit{superpose}(\mathit{pat1}, \mathit{pat2})$,

(ii) $\mathit{superpose}(\mathit{pat1}, \mathit{pat2}^c)$,

(iii) $\mathit{superpose}(\mathit{pat1}^c, \mathit{pat2})$,

(iv) $\mathit{superpose}(\mathit{pat1}^c, \mathit{pat2}^c)$.

However, it is clear that (i) is equivalent to (iv) w.r.t. the pattern matcher m_{c}, and that (ii) is also equivalent to (iii), because we have $m_{\text{c}}(p, t) = m_{\text{c}}(p, t^{\text{c}})$ for any degenerate pattern p and string t.

Thus, the following minor modification of SUPERPOSITION in Fig. 2 makes it possible that SUPERPOSITION can deal with P and the complements of P simultaneously.

- The *4*th line of the algorithm in Fig. 2 is replaced with "$D = \{\min\{s, s^c\} \mid s \in \Sigma^l$ is a substring of t or t^c, $t \in P\}$." Note that the function $\min$ returns the smallest one in lexicographical order from the given ones.
- The *17*th line is replaced with "**for** newPat in $\{\mathit{superpose}(\mathit{pat1}, \mathit{pat2}), \mathit{superpose}(\mathit{pat1}, \mathit{pat2}^c)\}$" and the subsequent two lines *18* and *19* are indented.

3.3 Restriction on Degeneracy

It would be reasonable to restrict the values of the degeneracy of degenerate patterns to be searched within a specified upper bound in order to reduce the running time. Because for usual DNA sequence sets in addition to random sequences, almost all of the degenerate patterns with quite a high degeneracy would be not worth searching. For example, the most general degenerate pattern of length l over Σ is Σ^l, which is meaningless to be searched. Actually, this observation is valid for motifs of binding sites in yeast, which do not have high degeneracies (for example, see [23]).

The next statement is clear from the definition of superposition.

Lemma 2 For degenerate patterns p and $q_1, \ldots, q_n$, let $r = \mathit{superpose}(p, q_1, \ldots, q_n)$. We have $\mathit{degen}(p) \leq \mathit{degen}(r)$.

We therefore modify SUPERPOSITION in the following way. Let B be an upper bound on the degeneracies of degenerate patterns to be searched. If the degeneracy of the degenerate pattern newPat of the *17*th line in Fig. 2 is less than or equal to B, SUPERPOSITION does the same thing, i.e., carry out the *18*th and *19*th lines. Otherwise it skips the lines, which contributes toward reducing the running time directly.

```
Input: a pattern length $l$, string sets $P$ and $Q$.
Output: the optimal pattern w.r.t. a specified conic score function $f$.
Procedure: $\mathrm{SUPERPOSITION}(P, Q, l)$
    $D = \{s \mid s$ is a substring of $t, t \in P, |s| = l\}$;
    $maxVal = -\infty$;
    $maxPat = \varepsilon$;  /* $\varepsilon$ is the empty string */
    SortedList $S$;
    Hash $G$;  /* Keep all of the enumerated patterns */
    for $s \in D$ do  $(maxVal, maxPat, S) = subroutine(s, P, Q, maxVal, maxPat, S)$;
    while ($S$ is not empty) do
        $(upperBound1, pat1) = S[1]$;  /* pop the pattern with the maximum upper bound in $S$ */
        if $upperBound < maxVal$ then break;
        SortedList $S'$;
        for $j = 2, \ldots, |S|$ do
            $(upperBound2, pat2) = S[j]$;
            if $upperBound2 < maxVal$ then $S = S[1..j-1]$; break;
            $newPat = superpose(pat1, pat2)$;
            if $newPat \notin G$ then
                $(maxVal, maxPat, S') = subroutine(newPat, P, Q, maxVal, maxPat, S')$;
        $S = S[2..|L|] + S'$;  /* sorted list concatenation */
    return $maxPat$;

Procedure: $subroutine(Pat, P, Q, maxVal, maxPat, S)$
    $\mathrm{tp} = |L(Pat, P, m_{\mathrm{basic}})|$; $\mathrm{fp} = |L(Pat, Q, m_{\mathrm{basic}})|$;
    $val = f(\mathrm{tp}, \mathrm{fp})$;
    if $val > maxVal$ then  $maxVal = val$; $maxPat = Pat$;
    $upperBound = f_{\mathrm{upper\_bound}}(\mathrm{tp}, \mathrm{fp}, P, Q)$;
    if $upperBound \geq maxVal$ then  push $(upperBound, Pat)$ into $S$;
    return $(maxVal, maxPat, S)$;
```

Figure 2: Algorithm SUPERPOSITION. **SortedList**, a type of data structure, is a list whose items are kept sorted in decreasing order. The i-th item of a **SortedList** S can be referred as $S[i]$. The consecutive items of S from the i-th item to the j-th item is referred as $S[i..j]$. The *length* of S is denoted by $|S|$.

4 Computational Experiments

In this section, we will report our preliminary computational experiments using the algorithm SUPERPOSITION on regulons of yeast, which are sets of genes co-regulated by a common transcription factor. We use the regulons reported in the database SCPD [23], in which for each regulon, the known binding sites are accumulated and compiled.

We here describe the score for measuring the performance of a motif-finder, which was proposed by [15] and also used by [20]. Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of positive DNA sequences, and let m^k and m^r be the "known" motif and the reported motif by an algorithm, respectively. The performance score Φ is defined as follows:

$$\Phi(S, m^k, m^r) = \frac{\sum_{i=1}^{n} |I_{m_i^k} \bigcap I_{m_i^r}|}{\sum_{i=1}^{n} |I_{m_i^k} \bigcup I_{m_i^r}|}$$

where for a motif m, I_{m_i} is the set of positions in S_i occupied by an occurrence of the motif m. Note that in our experiments, $I_{m_i^k}$ means the positions of sites in S_i reported as binding sites in SCPD.

We compare the performance score of SUPERPOSITION with that of quite a different type of algorithms, MEME [1] and AlignACE [16], and that of an algorithm similar to SUPERPOSITION, YMF [19]. MEME[1] and AlignACE[16] use local search techniques, based on an expectation maximization algorithm and a Gibbs sampling algorithm, respectively. The motif model that both use is a weight matrix. YMF[19] is an enumerative algorithm evaluating degenerate patterns with high z-scores. As conducted by [20], we also combine YMF with FindExplanators[3], which is a tool for selecting distinctive motifs from many motifs output by YMF. For short, we denote this combination by YMF. For the details of the differences between the three motif-finders, see Sinha and Tompa's work [20] of performance comparison of them. A feature common to the three algorithms and SUPERPOSITION is that their motif models are all *position-specific*. We wanted to include WINNOWER and SP-STAR, which find motif patterns whose mutations are not position-specific, but it does not seem to be available for downloading.

The versions of these tools are as follows: MEME is the version 3.0.4 available at ftp://ftp.sdsc.edu/pub/sdsc/biology/meme/. AlignACE is the linux version, which is the current and preferred version, at http://atlas.med.harvard.edu/download/. YMF can be downloaded from http://bio.cs.washington.edu/software.html. The current version of SUPERPOSITION, which is available at http://www.math.kyushu-u.ac.jp/~om/softwares.html, is the one without the options for optimizing lengths of degenerate patterns and the lower bounds on the number of occurrences of patterns. Note that all source code of the current version of SUPERPOSITION is written in the script language "python" (http://www.python.org), which might be an disadvantage of SUPERPOSITION in the comparison of the running times of the algorithms. On a dual Intel Xeon machine with 2G RAM in the default setting of Turbolinux workstation 8 (kernel version 2.4.18), all of the computational experiments were carried out on a single processor.

The 800 bp long upstream regions of genes in the regulon are extracted and given to each algorithm. The number of motifs finally reported by an algorithm was set to be two.

SUPERPOSITION can be parameterized with (l, d, n, s), where l is the non-spacer length, d is the upper bound on the degeneracy of the non-spacer, n is the number of negative strings, and s is the length of the spacer. The negative strings are generated using the 3rd order Markov model trained on all yeast upstream regions. To evaluate SUPERPOSITION, it is executed with various parameter values, for examples, $(l, d, n, s) = (8, 8, 800, 0)$, etc. The conic function f we use in this experiment is

$$f(\mathrm{tp}, \mathrm{fp}) = \left(\frac{\mathrm{tp}}{|P|}\right)^{\alpha} \left(1 - \frac{\mathrm{fp}}{|Q|}\right)^{\beta},$$

where P and Q are the sets of positive and negative strings, and $\mathrm{tp} = |L(p, P, m_c)|$ and $\mathrm{fp} = |L(p, Q, m_c)|$ for a degenerate pattern p. We set $\alpha = 0.5$ and $\beta = 2$ in this work.

To see how many patterns are pruned, we have introduced the *pruning ratio*, which is defined as $1 - \varepsilon$ where ε is the ratio of the number of the enumerated patterns by SUPERPOSITION to the number of degenerate patterns in the search space. Note that the enumerated patterns are the union of D defined in Fig. 2, that is, the substrings extracted from the positive strings, and the generated patterns by superposing existing ones.

The parameter settings of the other algorithms are the same as Sinha and Tompa's performance comparison work [20]. YMF is executed with three parameter sets: $(l, \lambda, \delta, t) = (6, 11, 2, 1000)$, $(7, 0, 2, 1000)$ and $(8, 0, 2, 1000)$, where l is the total length of the non-spacer parts, λ is the upper bound on the length of the spacer, δ is the maximum number of degenerate symbols, and t is the number of motifs output. MEME is run with the parameters $minw = 6$ and $maxw = 17$ which specify the range of lengths of motifs to be search. The parameter $mod = tcm$, which means multiple occurrences are allowed. AlignACE is run with $numcols = 6$ and $oversample = 2$.

The experiments were carried out for all 34 regulons in SCPD that have at least three genes. The results are presented in Table 1, in which for each regulon and each algorithm, the best performance score is written.

5 Discussion

Comparing the performance scores of the four algorithms, SUPERPOSITION outperforms any of YMF, MEME and AlignACE on 7 of the 34 regulons, as shown in the row labeled by "Wins" of Table 1. YMF, MEME and AlignACE win 9, 10 and 4, respectively. From this result, YMF and MEME look better than the others.

On the other hand, the last three rows of Table 1 show the number of the performance scores greater than or equal to thresholds 0.25, 0.5, and 0.75, respectively. From these data, it seems that SUPERPOSITION and YMF have a potential for identification of highly accurate motifs.

A drawback of SUPERPOSITION would be the running time. Compared with the time of the other tools, the time of SUPERPOSITION is definitely long. However, they all seem to be still *practical* and there is a guarantee that the found degenerate patterns are optimal in the search space w.r.t. the specified score function.

As an additional information on the performance of SUPERPOSITION, the pruning ratio for each regulon is also provided in Table 1. We can see that about 90% of the search space is pruned in most of the executions of SUPERPOSITION.

A future work is to devise a pruning algorithm for finding composite patterns, that is a combination of single motif patterns with additional attributes, for example, a constraint on the gap between the running occurrences of the single patterns and an order condition of occurrences of the patterns, etc. We can expect such an algorithm to find more accurate motifs since a composite motif is more discriminative in classifying the given strings (see for example, [14, 12, 11, 13]).

Acknowledgments

The authors would like to thank Ayumi Shinohara and Hideo Bannai for fruitful discussions. The authors would also like to thank anonymous referees for valuable comments. This work was supported in part by Research for the Future Program of JSPS and Grant-in-Aid for Scientific Research on Priority Areas (C) of MEXT.

References

[1] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21:51–80, 1995.

Table 1: Performance score and time. The column “size” shows the number of genes in each regulon. The columns labeled Φ_s, Φ_y, Φ_m and Φ_a show the performance scores of SUPERPOSITION, YMF, MEME and AlignACE, respectively. For each regulon, the highest performance score is underlined if it is above 0.10. The columns “time” give the running time of one execution of the algorithms, in seconds. The columns (l, d, n, s), “ratio” and “optimal” are the parameter values of SUPERPOSITION, the pruning ratios and the optimal pattern, respectively.

Regulon		SUPERPOSITION					YMF		MEME		AlignACE	
	size	(l, d, n, s)	ratio	optimal	Φ_s	time	Φ_y	time	Φ_m	time	Φ_a	time
ABF1	19	6,2,400,6	0.764	TCAN^{6}ACG	$\underline{0.31}$	2459	$\underline{0.31}$	19	0.25	968	0.04	189
BAS1	6	6,4,400,0	0.906	GGTACC	0.00	1355	$\underline{0.17}$	18	0.04	113	0.02	12
CAR1	12	6,2,400,0	0.856	CCGCSG	0.23	1444	0.29	6	$\underline{0.46}$	476	0.01	54
CPF1	3	7,8,800,0	0.999	CACGTGM	$\underline{0.91}$	259	0.62	16	0.62	37	0.00	6
CSRE	4	6,8,400,0	0.987	CGGN^{6}GGA	0.28	719	0.28	19	$\underline{0.32}$	68	0.24	9
GAL4	6	6,8,400,11	0.992	CGSN^{11}SCG	$\underline{0.71}$	473	0.61	24	0.66	105	0.64	32
GATA	4	6,8,400,0	0.956	CGCTTA	0.18	1262	$\underline{0.40}$	20	0.13	90	0.39	13
GCN4	9	6,4,400,0	0.883	GAGTCA	0.31	2332	$\underline{0.33}$	15	0.02	291	0.28	33
GCR1	6	6,4,400,0	0.998	CGGGAY	0.02	941	0.04	22	$\underline{0.29}$	113	0.02	29
GLN3	3	6,8,400,0	0.997	CCGACG	0.00	92	0.00	8	0.00	38	0.00	10
HAP1	5	6,4,800,6	0.901	CCGN^{6}CYC	$\underline{0.31}$	487	0.15	17	0.03	91	0.03	10
HAP2	4	6,4,400,0	0.961	ATGGCC	0.00	527	0.00	20	0.07	66	0.03	10
HSE	6	6,4,400,2	0.937	ACCNNGCS	0.00	1373	0.26	10	0.18	88	$\underline{0.27}$	13
MATA1	3	8,4,400,0	0.998	AATTAGGA	0.17	1453	0.19	21	$\underline{0.20}$	34	0.11	9
MATA2	7	6,4,400,0	0.907	CCATGT	0.08	1336	0.32	23	$\underline{0.36}$	164	0.03	22
MCB	6	6,4,400,0	0,974	ACGCGT	$\underline{0.67}$	216	$\underline{0.67}$	24	0.09	23	0.48	19
MCM1	23	6,2,400,6	0.778	RCCN^{6}GGA	0.26	4290	0.26	23	$\underline{0.43}$	1044	0.42	156
MIG1	9	6,2,800,0	0.892	CCGN^{6}GSG	0.03	951	$\underline{0.28}$	28	0.00	300	0.02	114
PDR3	7	8,2,400,0	0.993	TCCGYGGA	$\underline{0.78}$	1821	$\underline{0.78}$	31	0.41	208	0.49	16
PHO2	3	6,8,800,0	0.989	CCGGAG	0.06	844	0.00	9	0.00	45	0.00	5
PHO4	5	7,4,400,0	0.997	CACGTGS	0.20	73	$\underline{0.26}$	22	0.16	70	0.15	14
RAP1	16	6,2,400,0	0.784	CCCSCC	0.01	2063	0.17	13	$\underline{0.23}$	661	0.19	87
REB1	14	6,4,400,0	0.841	CGGGTA	0.31	3231	$\underline{0.32}$	12	0.26	520	0.00	66
ROX1	3	8,2,400,0	0.997	ATCGKCCG	0.00	746	0.07	20	0.03	47	$\underline{0.16}$	8
RPA	3	6,8,400,6	0.998	CGGN^{6}GCC	$\underline{0.20}$	67	0.12	23	0.00	47	0.00	8
SCB	3	7,8,400,0	0.998	TCGCGAA	0.08	124	0.35	9	0.28	45	$\underline{0.44}$	5
SFF	3	6,8,400,0	0.994	GCCCGK	0.04	158	0.06	27	0.04	45	0.09	13
STE12	4	6,4,400,0	0.959	CCGAGA	0.00	332	0.33	9	0.00	82	$\underline{0.37}$	12
TBP	17	6,2,400,0	0.773	CCGGRG	0.00	3743	0.00	11	0.00	750	0.00	106
UASCAR	3	8,4,400,0	0.999	CGCCGSTM	0.00	386	0.02	10	$\underline{0.13}$	46	0.04	12
UASH	18	6,4,400,0	0.908	GCCGCC	0.00	1605	0.01	15	0.00	860	0.00	111
UASPHR	17	6,2,400,0	0.782	GGCAAC	0.01	2399	0.04	13	0.02	836	0.05	108
UIS	3	6,8,400,0	0.995	CACCGC	0.08	190	0.24	23	$\underline{0.35}$	47	0.20	8
URS1H	13	7,4,400,0	0.981	TAGCCGC	0.59	1828	0.55	27	$\underline{0.67}$	496	0.40	67
Wins					7		9		10		4	
$\Phi \geq 0.25$					11		18		13		10	
$\Phi \geq 0.50$					5		5		3		1	
$\Phi \geq 0.75$					2		1		0		0	

[2] H. Bannai, S. Inenaga, A. Shinohara, M. Takeda, and S. Miyano. A string pattern regression algorithm and its application to pattern discovery in long introns. In *Genome Informatics 2002 (GIW2002)*, pages 3–11, 2002.

[3] M. Blanchette and S. Sinha. Separating real motifs from their artifacts. *Bioinformatics*, 17:S30–S38, 2001.

[4] J. Buhler and M. Tompa. Finding motifs using random projections. *Journal of computational biology*, 9:225–242, 2002.

[5] M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of closest substring and related problems. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, volume 2285 of *Lecture Notes in Computer Science*, pages 262–273, 2002.

[6] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for Closest String and related problems. In *Proceedings of the 12th Annual International Symposium on Algorithms and Computation (ISAAC 2001)*, volume 2223 of *Lecture Notes in Computer Science*, pages 441–452, 2001.

[7] G. Z. Hertz and G. D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15:563–577, 1999.

[8] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 633–642, 1999.

[9] C. Lawrence, S. Altschul, M. Boguski, J. Liu, F. Neuwald, and J. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

[10] C. Lawrence and A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7:41–51, 1990.

[11] X. Liu, D. Brutlag, and J. Liu. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In *Pacific Symposium on Biocomputing*, pages 127–138, 2001.

[12] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *J. Computational Biology*, 7:345–362, 2000.

[13] O. Maruyama, H. Bannai, Y. Tamada, S. Kuhara, and S. Miyano. Fast algorithm for extracting multiple unordered short motifs using bit operations. *Information Sciences*, 146:115–126, 2002.

[14] P. Pavlidis, T. Furey, M. Liberto, D. Haussler, and W. Grundy. Promoter region-based classification of genes. In *Pacific Symposium on Biocomputing*, pages 151–163, 2001.

[15] P. A. Pevzner and S.-H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 269–278, 2000.

[16] F. Roth, J. Hughes, P. Estep, and G. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, 16:939–945, 1998.

[17] A. Shinohara, M. Takeda, S. Arikawa, M. Hirano, H. Hoshino, and S. Inenaga. Finding best patterns practically. In S. Arikawa and A. Shinohara, editors, *Progress in Discovery Science 2001*, pages 307–317. Springer-Verlag Berlin Heidelberg, 2002.

[18] S. Sinha. Discriminative motifs. In *Proceedings of the 6th Annual International Conference on Computational Biology*, pages 291–298, 2002.

[19] S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 344–354, 2000.

[20] S. Sinha and M. Tompa. Performance comparison of algorithms for finding transcription factor binding sites. In *Proceedings of the 3rd IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2003)*, pages 214–220, 2003.

[21] E. Tateishi, O. Maruyama, and S. Miyano. Extracting best consensus motifs from positive and negative examples. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 219–230, 1996.

[22] J. van Helden, B. André, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281:827–842, 1998.

[23] J. Zhu and M. Q. Zhang. SCPD: a promoter database of the yeast Saccharomyces cerevisiae. *Bioinformatics*, 15(7/8):607–611, 1999.