ELSEVIER

Theoretical Computer Science 210 (1999) 245–260

Theoretical Computer Science

# Alphabet indexing for approximating features of symbols[1]

Shinichi Shimozono*

*Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, 820 Japan*

**Abstract**

We consider two maximization problems to find a mapping from a large alphabet forming given two sets of strings to a set of a very few symbols specifying a symbol wise transformation of strings. First we show that the problem to find a mapping that transforms the most of the strings as to form disjoint sets cannot be approximated within a ratio $n^{1/16}$ in polynomial time, unless P = NP. Next we consider a mapping that retains the difference of the maximum number of pairs of strings over the given sets. We present a polynomial-time approximation algorithm that guarantees a ratio $k/(k-1)$ for mappings to $k$ symbols, as well as proving that the problem is hard to approximate within an arbitrary small ratio in polynomial time. Furthermore, we extend this algorithm as to deal with not only pairs but also tuples of strings and show that it achieves a constant approximation ratio. 

*Keywords:* Alphabet indexing; Polynomial-time approximation; Nonapproximability; Knowledge acquistion; Data compression

## 1. Introduction

A large number of combinatorial problems in computer science arise from computational methods in molecular biology for identifying essential amino acid sequence elements that encode functional domains of proteins. Algorithmic operations required in these methods are heavy and have to deal with a huge amount of sequences. One major technique to obtain both time efficiency and succinctness of results in practice is the use of similarities determined among amino acids by some appropriate properties [6, 8, 13]. Such a set of similarities often defines a mapping that partitions the amino acids into disjoint sets, which can be associated with features and regarded as a reduced alphabet for expressing proteins [11, 21, 22].

Assume that a set $I$ of a few symbols and two disjoint sets $P$ and $Q$ of strings over an arbitrary large alphabet $A$ are given. An alphabet indexing $f$ for $P$ and $Q$

* Tel.: +81-948-29-7642; fax: +81-948-29-7601; e-mail: sin@ai.kyutech.ac.jp.

[1] A portion of this work was presented as a preliminary version in [17].

0304-3975/99/$ – see front matter 
*PII* S0304-3975(98)00088-7

by $I$ is a mapping from $A$ to $I$ whose homomorphism (symbol-wise transformation) transforms the strings in each $P$ and $Q$ into strings forming the corresponding sets that are disjoint from each other. It was formulated in [18, 19] to represent a classification of amino acids that will be found in knowledge acquisition processes. Since finding an alphabet indexing turned out NP-hard [18], an approximate alphabet indexing allowing nonempty overlap between the transformed sets was also defined with the help of a learning algorithm in [18, 19]. The knowledge acquisition system devised in [19] has successfully found an approximate alphabet indexing which coincides with '*hydropathy scale*' [13] of amino acids.

Our purpose in this paper is firstly to formalize approximate alphabet indexings for combinatorial problems dealing with string classifications and comparisons. We consider optimization problems defined by two maximization criteria for approxi mate alphabet indexings. To clarify the computational complexity to deal with approximate alphabet indexings, we secondly present the approximability and the non-approximability of the defined problems. The obtained results give insights for the hardness of the problem considered in [18, 19] as well as for the problem associated with new formulation.

Firstly, we consider the problem MAX INDEXING FOR DISJOINT SUBSETS that gives a simple and direct formulation to deal with knowledge acquisition problems, including [18, 19]. Given two sets $P, Q$ of strings in $A^*$ and a small alphabet $I$, the problem is to find an approximate alphabet indexing $f$ that maximizes the product of the sizes of subsets $P' \subseteq P$ and $Q' \subseteq Q$ for which $f$ is an alphabet indexing. We show that this problem is APX-hard if the length of strings in given sets is at least four. This implies that there is no polynomial-time approximation scheme (PTAS) for the problem, i.e., there exists a constant ratio within which no polynomial-time algorithms can guarantee a performance ratio unless $\mathrm{P} = \mathrm{NP}$. Furthermore, we show that, if the length of strings is not limited, then there is no polynomial-time algorithm that can approximate the problem within ratio $n^{1/16}$, unless $\mathrm{P} = \mathrm{NP}$, where $n$ is the coding size of a whole input.

Secondly, we consider a new problem MAX INDEXING FOR STRING DISTINCTION. It is, given two sets $P$ and $Q$ of strings in $A^*$ and a small alphabet $I$, to find an approximate alphabet indexing that can distinguish the maximum number of pairs of strings in $P \times Q$ after the transformation. It formalizes an approximate alphabet indexing which is suitable to use with a symbol-wise lossy text compression by a fixed encoding length [20], $l$-gram (or $l$-tuple, $l$-length substring) dependent methods for approximate string matching [2, 10, 23], and the substring dissimilarity problem [1].

We show that this problem is intractable even if two sets are completely the same one, and in general APX-hard. Then we present a polynomial-time greedy algorithm and prove that it can guarantee the worst-case performance ratio $k/(k-1)$ for approximate alphabet indexings by $k$ symbols. Furthermore, we extend the problem to that deals with the maximization of distinguishing tuples of strings over more than two sets of strings. We present a polynomial-time algorithm for this problem which follows the same idea of the algorithm for MAX INDEXING, and prove that the algorithm guarantees a constant performance ratio depending on $k$ and the size of tuples.

The remainder of this paper is organized as follows. In the next section, we review some notions and definitions relating to alphabet indexing, combinatorial optimization problems and their approximation. In Section 3, we introduce MAX INDEXING FOR DISJOINT SUBSETS, and then show the nonapproximability results for this problem. Section 4 deals with MAX INDEXING FOR STRING DISTINCTION problem. After showing the nonapproximability of the problem, we present a polynomial-time greedy algorithm and prove its worst-case performance ratio. Section 5 discusses the extended problem MAX TUPLE INDEXING, and its polynomial-time algorithm that also guarantees a constant performance ratio. We conclude with some discussion in Section 6.

## 2. Basic notions and definitions

The set $\Sigma$ denotes the distinguished alphabet consisting of two symbols 0 and 1, throughout this paper. Let $A$ be an alphabet of arbitrarily many symbols. Then $A^*$ denotes the set of all strings formed from symbols in $A$. For a string $s$ in $A^*$, $|s|$ denotes the length of $s$, and $s[i]$ denotes the $i$th symbol of $s$ for any $1 \leqslant i \leqslant |s|$. Let $l$ be a positive integer. An *$l$-gram on $A$* is a string $s \in A^*$ of length $l$. The set $A^l$ denotes the set of all $l$-grams on $A$.

Let $f$ be a mapping from an alphabet $A$ to another smaller alphabet $I$ with $|I| < |A|$. Then we denote by $f^*$ both the homomorphisms that map (i) a string $s$ in $A^*$ to $f^*(s) = f(s_1) \cdots f(s_{|s|})$ in $I^*$, and (ii) a set $S \subseteq A^*$ of strings to $f^*(S) = \{f^*(s) \mid s \in S\}$. Let $P$ and $Q$ be disjoint sets of strings over $A$. An *alphabet indexing $f$ of $A$ for $P$ and $Q$ by $I$* is a mapping from $A$ to $I$ that satisfies $f^*(P) \cap f^*(Q) = \emptyset$. In the similar context, an *approximate alphabet indexing*, or *indexing* in short, refers to any mapping from $A$ to $I$. The set $I$ is called the *indexing alphabet*, and an element in $I$ is called an *index*. An indexing by $\{1, \ldots, k\}$ of $k$ indices is said to be a *$k$-indexing*.

Let $A$ be an arbitrary large alphabet, and let $I$ be a small indexing alphabet. For $a, b \in A$, we say that an indexing $f$ of $A$ by $I$ *distinguishes $a$ and $b$* if $f(a) \neq f(b)$. Also, we say that $f$ distinguishes strings $s, t \in A^*$ if $f^*(s) \neq f^*(t)$. Let $S, S' \subseteq A^*$ be sets of strings, and let $s \in A^*$ be a string not included in $S$. Then we say that an indexing $f$ of $A$ *separates $s$ from $S$* if $f^*(s)$ is not included in $f^*(S)$. Also, if $f^*(S') \cap f^*(S) = \emptyset$ holds, then we say that $f$ separates $S'$ from $S$. In the following, we denote by $S'/_f S$ the subset of $S'$ consisting of all elements that are separated from $S$ by $f$, i.e., $S'/_f S = \{s \in S' \mid f^*(s) \notin f^*(S)\}$. For the sake of simplicity, we will deal with only strings of the same length when we compare them, i.e., we consider indexings for sets of $l$-grams for some positive integer $l > 0$.

Now we review notions and definitions of combinatorial optimization problems, their approximation, and approximation preserving reductions among the problems. By assuming some standard encoding on $\Sigma$, a *maximization problem $\Pi$* is defined by the following polynomial-time computable functions $g_\Pi, h_\Pi$ and $m_\Pi$: (i) $g_\Pi$ recognizes an *instance* $x \in \Sigma^*$ of $\Pi$, (ii) $h_\Pi$ recognizes a *solution* $s \in \Sigma^*$ of $x$, and (iii) $m_\Pi$ computes

the nonnegative integer *measure* $m_\Pi(x,s)$ of $s$ with respect to $x$. Then the purpose of $\Pi$ is, given an instance of $\Pi$, to find a solution which maximizes its measure.

Let $\Pi$ be a maximization problem and $x$ an instance of $\Pi$. An *optimal solution* of $x$ is a solution whose measure is maximum. We denote by $opt(x)$ the measure of an optimal solution of $x$. The *performance ratio* $R_\Pi(x,s)$ of a solution $s$ for $x$ with respect to $\Pi$ is the ratio $opt(x)/m_\Pi(x,s) \geqslant 1$. Let $r$ be a function that maps a natural number to a fraction greater than one. An *$r(n)$-approximation algorithm for $\Pi$* is an algorithm that produces, for any instance $x$ of $\Pi$, a solution $s$ such that $R_\Pi(x,s) \leqslant r(|x|)$. If a maximization problem $\Pi$ admits a polynomial-time $r(n)$-approximation algorithm, then we say that *$\Pi$ can be approximated in polynomial time within a ratio $r(n)$*. A *polynomial-time approximation scheme* (*PTAS*) for $\Pi$ is an algorithm that approximates $\Pi$, for any supplied parameter $\varepsilon > 1$, within a ratio $\varepsilon$ and runs in polynomial time with respect to the size $|x|$ (but can depend exponentially on $1/\varepsilon$).

Next we define the reduction between maximization problems that preserves performance ratios. Among various notions [4], we will follow the one defined by Khanna et al. [12]. It is essentially the same as the $L$-reduction by Papadimitriou and Yannakakis [15], and is an instance of the PTAS-reduction defined by Ausiello et al. [4].

**Definition 1.** Let $\Pi$ and $\Theta$ be maximization problems, and let $c$ be a constant. Then, a *$c$-approximation preserving reduction from $\Pi$ to $\Theta$*[2] is a pair $(\rho,\tau)$ of logspace-computable functions $\rho : \Sigma^* \to \Sigma^*$ and $\tau : \Sigma^* \times \Sigma^* \to \Sigma^*$ that satisfies the following three conditions for any instance $x$ of $\Pi$:

(i) The function $\rho$ maps $x$ to an instance $\rho(x)$ of $\Theta$,

(ii) For any solution $s$ of $\rho(x)$, the function $\tau$ maps a pair $(x,s)$ to a solution $\tau(x,s)$ of $x$, and

(iii) For any solution $s$ of $\rho(x)$ and the solution $\tau(x,s)$ of $x$, there holds

$$R_\Pi(x,\tau(x,s)) - 1 \leqslant c(R_\Theta(\rho(x),s) - 1).$$

Also, the problem $\Pi$ is said to be *$c$-reducible to $\Theta$* if a $c$-approximation preserving reduction from $\Pi$ to $\Theta$ exists.

The approximation preserving reducibility is reflexive and transitive. If for maximization problems $\Pi$ and $\Theta$ there exist (i) an $\varepsilon$-approximation polynomial-time algorithm for $\Theta$ and (ii) a $c$-approximation preserving reduction from $\Pi$ to $\Theta$, then $\Pi$ can be approximated in polynomial time within a ratio $c(\varepsilon - 1) + 1$. We say that a problem $\Theta$ is *APX-hard* if there is an APX-complete problem $\Pi$ such that $\Pi$ is $c$-reducible to $\Theta$. It is known in [3] that for an APX-hard problem there is no PTAS, unless P = NP.

[2] In [12] Khanna et al. referred to this reduction as error-preserving reduction ($E$-reduction) and formulated it naturally by relative error $E_\Pi(x,s) = R_\Pi(x,s) - 1$. In spite of this, we use only the notions with performance ratio.

## 3. Nonapproximability of indexing for maximum disjoint subsets

In the preceding works [18, 19], the measure of an indexing $f$ for sets $P$ and $Q$ is defined as follows. Let $M$ be an algorithm that produces for $f^*(P)$ and $f^*(Q)$ a representation $\gamma$ of a language $L(\gamma) \subseteq I^*$. Then, with respect to the output of $M$, the measure of $f$ is computed as the geometric mean (square-root mean) of the ratios $|\{p \in P \mid f^*(p) \in L(\gamma)\}|/|P|$ and $|\{q \in Q \mid f^*(q) \notin L(\gamma)\}|/|Q|$. This formulation was introduced to find an indexing as well as a representation of a language that maximizes the precision of the classification for both positive examples in $P$ and negative examples in $Q$.

It turns out that this measure function played an important role to obtain the successful computational results in [19]. The system employing Quinlan's algorithm [16] to produce a decision tree as a representation of a language. Since the classification is into two classes, *information gain* involved in the Quinlan's algorithm is almost equal to the geometric mean given above. It seems that this coincidence is one of the reasons that the optimization carried out by a local search algorithm was succeeded.

To concentrate on the hardness, we simplify the measure of indexing as to maximizes the "disjointness" between two transformed sets $f^*(P)$ and $f^*(Q)$. It would be at least retained naturally by an algorithm producing a representation of a language. The problem is defined as follows.

**Definition 2** (*Maximum Indexing for Disjoint Subsets, Max Disjoint Subsets*). An instance is a quadruple $(A, P, Q, k)$, where $A$ is an arbitrary large alphabet, $P$ and $Q$ are sets of strings over $A$, $k$ is an integer strictly smaller than $|A|$. A solution for $(A, P, Q, k)$ is a $k$-indexing $f$ of $A$. The measure of a solution $f$ is the product $|P/_f Q| \cdot |Q/_f P|$ of the sizes of the subsets separated from each other.

In [18] finding an alphabet indexing that completely separates $P$ and $Q$ from each other was shown to be NP-hard. Therefore, our interest is in whether it is possible by a polynomial-time algorithm to find an approximate alphabet indexing which separates the most of strings. However, the problem is hard even for approximation algorithms.

**Lemma 3.** Max Disjoint Subsets *is APX-hard even if the length of strings is restricted to four.*

**Proof.** It is proved by a 2-approximation preserving reduction from Max 2-Satisfiability (Max 2-Sat). The reduction follows the PLS-reduction presented in [18]. We can construct for a given 2-CNF formula $\phi$ an instance $(A, P, Q, 2)$ such that (i) a 2-indexing $f$ has non-zero measure with $|Q/_f P| = 1$ if and only if $f$ distinguishes the two symbols in $A$ associated with 'true' and 'false', and (ii) a Boolean assignment defined by a 2-indexing $f$ satisfies $|P/_f Q| - 1$ clauses in $\phi$. Then, for any 2-CNF

formula $\phi$ and for any 2-indexing $f$ such that $opt(\phi) \geqslant 2$ and $opt(\phi) \geqslant m(\pi, f) > 2$,

$$\frac{opt(\phi)}{m(\phi, f)} - 1 = \frac{opt(\pi) - 1}{m(\pi, f) - 1} - 1 \leqslant 2\left(\frac{opt(\pi)}{m(\pi, f)} - 1\right).$$

(Exceptions for the conditions can be detected and handled in polynomial time.) With the function pair, it suffices for a 2-approximation preserving reduction. □

Generally, we can show a stronger result.

**Theorem 4.** MAX DISJOINT SUBSETS *cannot be approximated in polynomial time within a ratio* $n^{1/16-\varepsilon}$ *for any* $\varepsilon > 0$ *unless* $P = NP$.

**Proof.** We show an approximation preserving reduction from MAX INDEPENDENT SET (symbolized as MIS) to MAX DISJOINT SUBSETS (MDS). To show the nonapproximability thresholds depending on the size of inputs, we will use, in spite of the condition (iii) of the reducibility, a polynomial relationship between the performance ratios.

The problem MAX INDEPENDENT SET is, given a graph $G = (N, E)$, to find the largest independent set $N' \subseteq N$ on $G$, i.e., a largest subset $N'$ to which no edge in $E$ contributes its both endpoints. Let $G = (N, E)$ be a graph with a set $N = \{1, \ldots, n\}$ of nodes given as an instance of MAX INDEPENDENT SET. Let $\langle i \rangle$ be a string in $\{b_0, b_1\}^{\lceil \log n \rceil + 1}$ representing an integer $i$ by some natural binary encoding. We construct an instance $\pi_G = (A, P, Q, k)$ of MAX DISJOINT SUBSETS for $G$ as follows:

(1) $A = \{b_0, b_1, a_1, \ldots, a_n\}$ and $k = 2$ (thus $I = \{1, 2\}$).
(2) For every node $i \in N$, $P$ includes three strings $\langle i \rangle b_0 a_i$, $\langle i \rangle b_0 b_0$ and $\langle i \rangle b_0 b_1$.
(3) For every node $i \in N$ and edge $(i, j) \in E$, $Q$ includes two strings $\langle i \rangle b_1 a_i$ and $\langle i \rangle b_0 a_j$.

The strings $\langle i \rangle b_0 a_i$ in $P$ and $\langle i \rangle b_1 a_i$ in $Q$ are associated with the node $i$, and are called *node-strings*. The string $\langle i \rangle b_0 a_j$ in $Q$ is associated with the node $j$ which is adjacent to $i$. Note that all strings in $P$ and $Q$ are of length $\lceil \log n \rceil + 3$.

Let $f$ be a 2-indexing of $A$. If $f$ satisfies $f(b_0) = f(b_1)$, then the measure of $f$ is exactly zero. On the other hand, if $f$ satisfies $f(b_0) \neq f(b_1)$, then $f$ can have some non-zero measure. Therefore, without loss of generality, we can assume that any 2-indexing $f$ satisfies $f(b_0) \neq f(b_1)$.

**Claim 5.** *A 2-indexing* $f$ *separates every string of the form* $\langle i \rangle b_1 a_i \in Q$ *from* $P$, *and separates no string of the form* $\langle i \rangle b_0 a_j \in Q$ *from* $P$.

The first statement is true because $f(b_0) \neq f(b_1)$ and all the strings in $P$ has $b_0$ at before the last symbol. Also, $f(b_0) \neq f(b_1)$ implies that every string $\langle i \rangle b_0 a_j$ with $i \neq j$ is identical to either $\langle i \rangle b_0 b_0$ or $\langle i \rangle b_0 b_1$ in $P$ after the transformation by $f^*$.

The above claim guarantees that every indexing $f$ (satisfying $f(b_0) \neq f(b_1)$) separates exactly $n = |N|$ strings in $Q$ from $P$. It keeps that the measure of $f$ to be just

$n$ times the number of strings separated in $P/_fQ$. On the other hand, for strings in $P$, the following claim holds.

**Claim 6.** *A 2-indexing $f$ separates $\langle i\rangle b_0a_i \in P$ from $Q$ if and only if either $\langle i\rangle b_0b_0$ or $\langle i\rangle b_0b_1$ is in $P/_fQ$.*

A node-string $\langle i\rangle b_0a_i \in P$ is in $P/_fQ$ if and only if all the symbols that correspond to the nodes adjacent to $i$ are distinguished by $f$ from $a_i$. If $f$ separates $\langle i\rangle b_0a_i \in P$ from $Q$, then either $\langle i\rangle b_0b_0$ or $\langle i\rangle b_0b_1$ must be in $P/_fQ$, and vice versa. Therefore, the size of $P/_fQ$ is twice the number of the node-strings separated from $Q$.

The above two claims are summarized as follows:

**Proposition 7.** *Let $N'_0$ and $N'_1$ be the subsets of $N$ defined by*

$$N'_0 = \{i \in N \mid f(a_i) = 1 \text{ and } \langle i\rangle b_0a_i \in P/_fQ\},$$

$$N'_1 = \{i \in N \mid f(a_i) = 2 \text{ and } \langle i\rangle b_0a_i \in P/_fQ\}.$$

*Then, both $N'_0$ and $N'_1$ are independent sets of $G$.*

Note that conversely we can construct an indexing $f$ for $\pi_G$ from an independent set $N'$ of $G$ by assigning 0 to all symbols except $b_1$ and any $a_i \in A$ corresponding to the node $i \in N'$.

Without loss of generality, we can assume that $N'_0$ is larger than $N'_1$. Then the following is immediate:

**Proposition 8.** *Let $f$ be an alphabet indexing $f$ of $\pi_G$, and let $N'_0$ be an independent set constructed from $f$. Then* (i) *the size of $P/_fQ$ is no greater than* 4 *times the size of $N'_0$, and* (ii) *the size of $P/_fQ$ is at least the size of $N'_0$.*

Let $\tilde{f}$ be an optimum indexing for $\pi_G$, and let $\tilde{N}'_0$ and $\tilde{N}'_1$ be the independent sets with respect to $\tilde{f}$. Since the sizes of both $Q/_fP$ and $Q/_{\tilde{f}}P$ are equal to $n$, the performance ratio of an indexing $f$ is simply $|P/_{\tilde{f}}Q|/|P/_fQ|$. By Proposition 8, between the performance ratios $R_{\mathrm{MIS}}(G, N'_0)$ and $R_{\mathrm{MDS}}(\pi_G, f)$, we have

$$R_{\mathrm{MDS}}(\pi_G, f) = \frac{|\tilde{N}'_0| + |\tilde{N}'_1|}{|N'_0| + |N'_1|} \geqslant \frac{opt(G)}{2 \cdot |N'_0|} = \frac{1}{2} \cdot R_{\mathrm{MIS}}(G, N'_0).$$

Therefore, if MAX DISJOINT SUBSETS can be approximated within a ratio $r(|\pi_G|)$ for some function $r$, then MAX INDEPENDENT SET can be approximated within a ratio $2 \cdot r(|\pi_G|)$. Now recall that the construction of an instance $\pi_G$ satisfies $|A| = n + 2$ and $|P \cup Q| \leqslant 3n + n^2$. Since all symbols in $A$ can be encoded in $\Sigma^{\lceil \log n \rceil + 1}$ and the length of every string is $\lceil \log n \rceil + 3$, $|\pi_G| < n^4$ is satisfied for sufficiently large $n$. It is known in [5] that MAX INDEPENDENT SET problem cannot be approximated within $n^{\frac{1}{4}-\delta}$ for any

$\delta > 0$ unless P = NP. This implies that, for any $0 < \delta \leqslant \varepsilon/4$,

$$R_{\mathrm{MDS}}(\pi_G, f) \geqslant \tfrac{1}{2} R_{\mathrm{MIS}}(G, N_0') \geqslant \tfrac{1}{2} n^{1/4-\delta} \geqslant \tfrac{1}{2} |\pi_G|^{1/16-\varepsilon}$$

holds for sufficiently large instances. □

The above nonapproximability threshold can be tightened with stronger assumptions. In [5] it is also presented that MAX INDEPENDENT SET is hard to approximate within ratio $n^{1/3}$ if coRP $\neq$ NP is supposed. Therefore,

**Corollary 9.** MAX DISJOINT SUBSETS *cannot be approximated within* $n^{1/12-\varepsilon}$ *for any* $\varepsilon > 0$ *in polynomial time, unless coRP = NP.*

## 4. Indexing for maximizing pair-wise distinction of strings

In this section, we discuss the hardness and the approximability of the maximization of an approximate alphabet indexing by the pair-wise string identification measure. We firstly introduce some notions.

Let $P$ and $Q$ be sets of strings over an alphabet $A$. We denote by $[P \times Q]$ the set of all pairs of different strings over $P$ and $Q$, i.e., $[P \times Q] = \{(p,q) | p \in P, q \in Q, p \neq q\}$. Let $p$ and $q$ be strings in $A^l$ for some $l > 0$, and let $a$ and $b$ be symbols in $A$. For a pair of strings $(p,q)$, we say that $a$ and $b$ are *facing* in $(p,q)$ if, for some $i$ with $1 \leqslant i \leqslant l$, either $(p[i] = a) \wedge (q[i] = b)$ or $(p[i] = b) \wedge (q[i] = a)$. Note that $(p,q)$ is distinguished by $f$ if $(p,q)$ has at least one facing pair of symbols $a$ and $b$ with different indices, $f(a) \neq f(b)$.

**Definition 10** (*MAX INDEXING FOR STRING DISTINCTION, MAX INDEXING*). An instance is a quadruple $(A,P,Q,k)$, where $A$ is an arbitrary large alphabet, $P$ and $Q$ are sets of strings over $A$, $k$ is an integer strictly smaller than $|A|$. A solution for $(A,P,Q,k)$ is a $k$-indexing $f$ of $A$. The measure of a solution $f$, a $k$-indexing, is the number of pairs in $P \times Q$ whose strings are transformed into different ones, i.e. $|\{(p,q) \in P \times Q \mid f^*(p) \neq f^*(q)\}|$.

MAX $k$-INDEXING is the subproblem of MAX INDEXING wherein the number of indices is always $k$ and an instance is supplied as a triple $(A,P,Q)$.

The weighted version of this problem can be naturally defined, by instances providing the weight functions $w : P \to \mathbf{Z}^+$ and $w' : Q \to \mathbf{Z}^+$, where $\mathbf{Z}^+$ is the set of nonnegative integers, and the measure

$$\sum_{(p,q) \in P \times Q \wedge f^*(p) \neq f^*(q)} w(p) \cdot w'(q)$$

of an indexing $f$. In the following, we only consider the unweighted version of MAX INDEXING. Note that, however, the results obtained for the unweighted problem in this paper also hold for the weighted version.

From the definition, instances of MAX DISJOINT SUBSETS whose sets of strings are completely the same are negligible. However, MAX INDEXING remains still intractable even if the two sets of strings are identical.

**Lemma 11.** *Even for instances of* MAX INDEXING *with two same sets of strings, deciding whether there exists or not a k-indexing that distinguishes all pairs of different strings is* NP*-complete.*

**Proof.** We show a sketch of a reduction from 3-SAT. Given an instance $(C,X)$ of 3-SAT, a set $C$ of 3-literal clauses over variables $X$, we construct an instance $(A,Q,Q)$ of MAX 2-INDEXING as follows. Let $A$ be $X \cup \{\mathtt{t},\mathtt{f}\}$, where $\mathtt{t}$ and $\mathtt{f}$ are symbols associated with Boolean values and not included in $X$. For each clause $c_i$ with literals $l_1, l_2$ and $l_3$ in $C$, the set $Q$ includes two strings $\langle l_1\rangle\langle l_2\rangle\langle l_3\rangle\langle i\rangle$ and $\mathtt{tftftf}\langle i\rangle$, where $\langle l_j\rangle$ is either $\mathtt{t}x_k$ or $x_k\mathtt{f}$ for some $x_k \in X$, depending on whether $l_j$ is positive $x_k$ or negative $\bar{x}_k$, and $\langle i\rangle \in \{\mathtt{t},\mathtt{f}\}^{\lceil \log |C| \rceil}$ represents $i$ in some natural binary coding. Then an indexing $f$ distinguishes all originally different pairs in $Q \times Q$ if and only if $f(\mathtt{t}) \neq f(\mathtt{f})$ and $f^*(\langle l_1\rangle\langle l_2\rangle\langle l_3\rangle) \neq f^*(\mathtt{tftftf})$, and this implies that there is an assignment $f : X \to \{\mathtt{t},\mathtt{f}\}$ satisfying all clauses in $C$. □

Furthermore, in general case, the next lemma holds.

**Lemma 12.** MAX $k$-INDEXING *is APX-hard.*

**Proof.** We show a 1-approximation preserving reduction (an APX-reduction) to MAX 2-INDEXING from MAX CUT-$B$, which is known APX-complete [15]. Given a graph $G=(V,E)$ with a constant degree bound $B>0$, MAX CUT-$B$ is the problem to find a partition of the vertices $S$ and $\bar{S} = V - S$ that maximizes the number of edges going from $S$ to $\bar{S}$.

We assume that the set $V$ of nodes is given as $V = \{v_1, \ldots, v_n\}$ for some positive integer $n$. Then, the transformation from an instance $G$ of MAX CUT to an instance $(A,P,Q)$ of MAX 2-INDEXING is defined as follows. The alphabet $A$ is equal to $V$, and $P$ consists of only one string

$$v_1^B \cdot v_2^B \cdots v_{n-1}^B \cdot v_n^B.$$

For every edge $(v_i, v_j) \in E$ from $v_i \in V$ to $v_j$ which is $b(i,j)$th smallest adjacent node (thus $1 \leqslant b(i,j) \leqslant B$), $Q$ includes

$$v_1^B \cdots v_i^{b(i,j)-1} v_j v_i^{B-b(i,j)} \cdots v_n^B.$$

It is immediate that $|[P \times Q]| = |E|$ and the facing pair associated with an edge $(v_i, v_j)$ occupies the position that is unique to it.

For these instances $G$ and $(A,P,Q)$, the partition corresponding to a 2-indexing $f$ is defined by $S_f = \{v_i \in V \mid f(v_i) = 1\}$. Then, $f$ distinguishes a pair of $v_1^B \cdots v_i^B \cdots v_n^B$ and $v_1^B \cdots v_i^{b(i,j)-1} v_j v_i^{B-b(i,j)} \cdots v_n^B$ in $P \times Q$ if and only if the partition $(S_f, V - S_f)$

separates the edge $(v_i, v_j) \in E$. Therefore, there is an approximation preserving reduction with a factor $c = 1$. □

This concludes that MAX 2-INDEXING is not approximable within 1.012 from [5]. Thus our next goal is to devise a polynomial-time algorithm which can achieve a constant performance ratio as small as possible. Let us consider applying algorithm $Greedy_k$ explained below.

**Algorithm 1.** $Greedy_k$(input : $A, P, Q$)
1. *Let* $R := [P \times Q]$ *and* $C := \emptyset$.
2. *For each* $a \in A$ *do*
   (a) *For each* $i \in \{1, \ldots, k\}$, *compute* $S(i) := \{(p,q) \in R \mid \exists b \in C$ *such that* $b$ *faces* $a$ *in* $(p,q)$ *and* $f(b) \neq i\}$.
   (b) *Find* $j$ *that maximizes the size* $|S(j)|$.
   (c) $R := R - S(j)$, $f(a) := j$ *and* $C := C \cup \{a\}$.
3. *Output* $f$.

The set $R$ initially consists of all possible pairs of different strings in $P \times Q$. After all the iterations in Step 2, $R$ contains only pairs for which all chances to distinguish two strings have been failed. The set $C$ holds symbols whose indices have been fixed.

The algorithm takes $O(k \cdot |A| \cdot |P| \cdot |Q|)$ time, and thus runs in $O(n^3)$ with respect to the coding length of $P$ and $Q$. It is still polynomial even if $k$ becomes a part of input. For this algorithm, we have the following theorem.

**Theorem 13.** $Greedy_k$ *approximates* MAX $k$-INDEXING *within ratio* $k/(k-1)$.

**Proof.** Suppose that the algorithm has been applied to an instance $\pi = (A, P, Q)$, and is now computing Step 2(a) at the $l$th iteration. Then, $R$ is the set of pairs that are still not distinguished after the $(l-1)$th iteration. Let $a_l \in A$ be the symbol whose index is going to be fixed at the $l$th iteration. For all $1 \leqslant i \leqslant k$, we define

$$d_i = \{(p,q) \in R \mid \exists b \in C \text{ such that } b \text{ faces } a_l \text{ in } (p,q) \text{ and } f(b) = i\}.$$

The pairs in $d_i$ are included in $S(j)$ for all $i \neq j$. Also, if $f(a_l) \neq i$, the pairs in $d_i$ will be removed from $R$ in Step 2(c). We divide the union $\bigcup_{i=1}^{k} d_i$ into $k+1$ disjoint partitions $D_0, D_1, \ldots, D_k$ as follows.

(1) $D_0 = \bigcup_{i \neq j}(d_i \cap d_j)$, the set of pairs that are distinguished by any assignment of index $f(a_l)$ to $a_l$.
(2) $D_i = d_i - D_0$, the set of pairs that are distinguished if and only if the index for $a_l$ is not $i$, $f(a_l) \neq i$.

Additionally, we define $Ex_i \subseteq D_i$ as the set of the pairs that will be lose all chances to be distinguished if $i$ is assigned to $f(a_l)$. Note that $D_i \cap D_j = \emptyset$ for all $1 \leqslant i \neq j \leqslant k$, and if $i$ is assigned to $f(a_l)$, then all strings in $Ex_i$ will remain in $R$ forever. To choose

$j \in \{1,\ldots,k\}$ for $f(a_l)$, algorithm $Greedy_k$ maximizes the size of $S(j) = D_0 \cup (\bigcup_{i \neq j} D_i)$ and thus $j$ must be a number that minimizes $|D_j|$. For the assignment $j$ to $f(a_l)$, we lose the pairs in $Ex_j$, so we have

$$|S(j)| = |D_0| + \left|\bigcup_{i \neq j} D_i\right| \geqslant |D_0| + (k-1) \cdot |D_j| \geqslant (k-1) \cdot |Ex_j|.$$

This inequality holds at any iteration. When the algorithm stops, $R$ is the union of all $Ex$'s of all iterations. Conversely, $[P \times Q] - R$ is the union of all sets $S(j)$ for $f(a_l) = j$ with $1 \leqslant l \leqslant |A|$. Therefore, with the inequality given above, we have $|[P \times Q] - R| \geqslant (k-1) \cdot |R|$. Since $|[P \times Q] - R|$ is the measure of the produced indexing $f$, and $opt(\pi) \leqslant |[P \times Q]|$, we have the inequality

$$opt(\pi) \leqslant |[P \times Q]| = |[P \times Q] - R| + |R| \leqslant \frac{k}{k-1} \cdot m(\pi, f).$$

Therefore, the algorithm $Greedy_k$ guarantees a performance ratio $k/(k-1)$ for every instances.

Now we show that there actually exist instances for which our algorithm produces indexings whose performance ratio approach to $k/(k-1)$. Let $m$ be a sufficiently large positive integer. Let us consider the following instance $\pi'$ with strings of length two:

$$A = \{a_1,\ldots,a_k, b_1, b_2, c_1,\ldots,c_m\},$$

$$P = \{a_i b_1 \mid 1 \leqslant i \leqslant k\}, Q = \{a_i b_2 \mid 1 \leqslant i \leqslant k\} \cup \{c_i b_1 \mid 1 \leqslant i \leqslant m\}.$$

In $P \times Q$, we have $(a_i b_1, a_j b_2)$ for $1 \leqslant i, j \leqslant k$ and $(a_i b_1, c_j b_1)$ for $1 \leqslant i \leqslant k, 1 \leqslant j \leqslant m$. Clearly, an optimal indexing is, for example

$$f(a) = \begin{cases} 1 & \text{if } a = a_i \text{ with } 1 \leqslant i \leqslant k, \\ i & \text{if } a = b_i, \\ 2 & \text{if } a = c_i \text{ with } 1 \leqslant i \leqslant m. \end{cases}$$

This distinguishes all $k^2 + mk$ pairs of the forms $(a_i b_1, a_j b_2)$ and $(a_i b_1, c_j b_1)$, by $f(b_1) \neq f(b_2)$ and $f(a_i) \neq f(c_j)$. However, the algorithm may assign $k$ different symbols to $a_1,\ldots,a_k$ firstly with distinguishing $2(l-1)$ pairs at each $l$th decision of index. Then, in succession, the algorithm will assign different two symbols to $b_1$ and $b_2$. As a result, all pairs of the form $(a_i b_1, a_j b_2)$ can be distinguished, but at least one pair of the form $(a_i b_1, c_j b_1)$ cannot be distinguished for each $c_j$. This results that $f$ produced by the algorithm distinguishes all pairs of the form $(a_i b_1, a_j b_2)$, but for any succeeding indexing to each $c_j$, $f$ can save only $k-1$ pairs from $(a_i b_1, c_j b_1)$'s. With $m \to \infty$, the performance ratio goes to

$$\frac{k(k-1) + k + m(k-1)}{k^2 + mk} \to \frac{k-1}{k}.$$

This realizes the worst-case performance ratio $k/(k-1)$. □

## 5. Extending distinction in pair to tuples of strings

In this section, we consider a problem MAX $m$-TUPLE $k$-INDEXING, which is an extension of MAX $k$-INDEXING to deal with tuples of strings more than two.

Let $t=(q_1,\ldots,q_m)$ be a tuple of $m$ strings $q_1,\ldots,q_m\in A^*$, and let $f$ be a $k$-indexing of $A$. We say that an indexing $f$ *totally distinguishes* $t$ if $f^*(q_i)\neq f^*(q_j)$ for all $q_i\neq q_j$ in $t$. Let $f$ and $f'$ be $k$-indexings, and let $C$ be a subset of $A$. We say that $f$ *derives* $f'$ *on* $C$ if, for any $a\in C$, $f(a)=i$ implies $f'(a)=i$.

**Definition 14** (*MAX m-TUPLE k-INDEXING*). An instance is an $(m+2)$-tuple $(A,k,Q_1,\ldots,Q_m)$ of an alphabet $A$, a positive integer $k$ and $m$ sets of strings $Q_1,\ldots,Q_m$ over $A$. A solution is a $k$-indexing $f$, and the measure of a solution $f$ is the number of tuples in $Q_1\times\cdots\times Q_m$ that are totally distinguished by $f$.

For MAX $m$-TUPLE $k$-INDEXING, we consider a greedy algorithm $Greedy_k^m$ defined as follows.

**Algorithm 2.** $Greedy_k^m$(input: $A,Q_1,\ldots,Q_m$)

1. *Let* $T:=Q_1\times\cdots\times Q_m$, $C:=\emptyset$ *and* $w(t):=r^{-\mathrm{diff}(t)}$ *for all* $t\in T$, *where* $r\geqslant 1$ *is an arbitrarily fixed "preserving constant," and* $\mathrm{diff}(t)$ *is the number of originally different pairs in* $t$.
2. *For each* $a\in A$ *do*
   (a) *For* $i\in\{1,\ldots,k\}$ *and* $t\in T$, *let* $f(a):=i$, *and compute the following*:
      (i) $p(i,t)=$ *the number of pairs of strings that are in* $t$ *and are distinguished by the assignment* $f(a):=i$, *i.e., the number of pairs of strings in* $t$ *in which some symbol* $b\in C$ *with* $f(b)\neq i$ *faces to* $a$.
      (ii) $D(i)=$ *the subset of* $T$ *whose all tuples can be totally distinguished by some indexing* $f'$ *derived from* $f$ *on* $C\cup\{a\}$.
   (b) *Choose index* $j$ *that maximizes*

$$\sum_{t\in D(j)\&p(j,t)\geqslant 1} w(t)\cdot r^{p(j,t)}.$$

   (c) $T:=D(j)$, $w(t):=w(t)\cdot r^{p(j,t)}$ *for all* $t\in T$, $f(a):=j$ *and* $C:=C\cup\{a\}$.
3. *Output* $f$.

Notice that, for $m=2$, the algorithm $Greedy_k^m$ is equivalent to $Greedy_k$ since for all $t\in T$ the initial value of $w(t)$ is fixed to $1/r$ by $\mathrm{diff}(t)=1$, $p(j,t)$ can have non-zero value 1 at most only once, and $D(j)$ is equivalent to the set $R$ of remained pairs at every iteration.

**Claim 15.** $Greedy_k^m$ *runs in polynomial time.*

**Proof.** The time needed for computing $D(i)$'s in Step 2(a) dominates the running time of the algorithm. Let $n$ be the coding length of $Q_1,\ldots,Q_m$, and let $l$ be the length

of the longest string in these sets. In a computation of $D(i)$, we must decide whether or not, for each tuple $t$, a $k$-indexing implied by $f$ over $C \cup \{a\}$ that distinguishes $t$ exists. This can be done by finding a set of pairs of symbols, namely a "witness set", that satisfies the following conditions:

(i) In every pair $(p,q)$ of strings in $t$, at least one pair in a witness set must appear as a facing pair of symbols in $(p,q)$, and

(ii) At least one $k$-indexing $f'$ which can be implied by $f$ over $C \cup \{a\}$ must distinguish all pairs of symbols.

Since there are at most $l$ facing pairs for each pair of strings in a tuple, the number of candidates for a witness set is at most $l^{m(m-1)/2}$. In each set, we have at most $m(m-1)$ different symbols, and we have to verify at most $k^{m(m-1)}$ $k$-indexings by fixing $|A-C|-1$ indices. Therefore, the computation of $D(i)$ for each $1 \leqslant i \leqslant k$ can be done in $O(n \cdot l^{m(m-1)/2} \cdot k^{m(m-1)}) = O(n^{m(m-1)/2+1})$, and the whole algorithm runs in $O(n^{m(m-1)/2+2})$ time. □

If $m$ is unbounded, then deciding even for one tuple $t$ whether a $k$-indexing that totally distinguishes $t$ exists is intractable. It is equivalent to the problem regarded in Lemma 11.

For the above algorithm, we have the following result.

**Theorem 16.** *If we choose $r=k$, algorithm $Greedy_k^m$ produces a $k$-indexing that totally distinguishes at least $1/r^{m(m-1)/2}$ of all the tuples. If $k$ is sufficiently large to satisfy $k > m(m-1)/2$, then $r$ can be reduced to $r = k/k - m(m-1)/2$.*

The above statement gives an "absolute performance ratio" of the algorithm. We can conclude this as follows.

**Corollary 17.** *$Greedy_k^m$ approximates* MAX $m$-TUPLE $k$-INDEXING *within $k^{m(m-1)/2}$. If $k > m(m-1)/2$ is satisfied, then the performance ratio can be reduced to $(k/k - m(m-1)/2)^{m(m-1)/2}$.*

Notice that, for $m=2$, the guaranteed performance ratio is also the same with that of $Greedy_k$.

**Proof.** At first, let us note the following two facts.

(1) At initialization of the algorithm in Step 1, the number $\mathrm{diff}(t)$ of originally different pairs of strings in tuple $t$ is at most $m(m-1)/2$. Thus the sum of the weights $\sum_{t \in T} w(t)$ is at least $|T| \cdot r^{-m(m-1)/2}$.

(2) While the iterations at Step 2, the weight $w(t)$ assigned to a tuple $t$ does not exceed 1, since any tuple $t$ with $w(t)=1$ has no pairs to be distinguished.

Suppose that the algorithm is deciding an index for a symbol in Step 2. Let $D'(i)$ be the subset of $D(i)$ defined by $D'(i) = \{t \in D(i) \mid p(i,t) \geqslant 1\}$ for $1 \leqslant i \leqslant k$, and let $T' \subseteq T$ be the union of all $D'(i)$ for $1 \leqslant i \leqslant k$. In Step 2(b), an index $j$ is selected to maximize

the sum of the updated weights of tuples in $D(j)$. If we choose $r=k$, the inequality

$$\sum_{t\in D'(j)} w(t)\cdot r^{p(j,t)} \geqslant r \sum_{t\in D'(j)} w(t) \geqslant \sum_{t\in T'} w(t)$$

holds since $T'=\bigcup_{i=1}^{k} D'(i)$. The weight $w(t)$ of any tuple $t\in T$ for which $p(j,t)$ equals zero is unchanged in Step 2(b). Therefore, we have

$$\sum_{t\in D(j)} w(t)\cdot r^{p(j,t)} \geqslant \sum_{t\in T} w(t),$$

i.e., the choice of an index $j$ does not decrease the sum of the weights of tuples in new $T=D(j)$ in Step 2(c) compared with that of the previous iteration. This holds at every iteration in Step 2, and thus the algorithm retains the initial weight sum and finally produces a $k$-indexing that totally distinguishes at least $r^{-m(m-1)/2}$ of all the tuples for $r=k$.

Now we look into a special case, where the preserving constant $r$ can be chosen smaller than $k$. Let $1\leqslant r_0\leqslant m$ be a positive integer. If all totally distinguishable tuples belong at least $r_0$ sets among all $D$'s, the sum of the weights of all $D$'s satisfies

$$\sum_{i=1}^{k} \sum_{t\in D(i)} w(t) r^{p(i,t)} \geqslant r_0 \sum_{t\in T} w(t).$$

Since each selected index $j$ maximizes the sum of the weights of survived tuples, the weight sum in $D(j)$ is larger than the average of those of all $D$'s. Thus $\sum_{t\in D(j)} w(t)$ is at least $r_0/k \sum_{t\in T} w(t)$.

Let us estimate $r_0$ greater than 1 for a case $m$ is relatively small. To totally distinguish the tuples, we have to distinguish pairs of facing symbols at most $m(m-1)/2$. Obviously, a tuple will remain distinguishable if symbols in every facing pair receive different indices. Therefore, if $k>m(m-1)/2$ holds, then $r_0=k-m(m-1)/2$. □

Note that worst-case instances for $Greedy_k^m$ can be constructed in the similar way to that shown in the proof of Theorem 13.

## 6. Conclusion

We have introduced two measures for approximate alphabet indexings which are designated to cope with substring comparative methods. According to these measures, we have defined two maximization problems, MAX DISJOINT SUBSETS and MAX INDEXING.

For MAX DISJOINT SUBSETS, we have proved APX-hardness when strings dealt with are no shorter than four. Also, we have shown that the problem for strings of unlimited length has no polynomial-time algorithm that can always find an approximate alphabet indexing whose measure is more than $1/n^{1/16}$ times the optimum unless P = NP.

These results indicate that even the approximation is hard in polynomial time. However, in computational experiments, a local search algorithm employed in [19]

produced with high probability a good solution which seems to be near-optimal in a reasonable amount of time. Specifying sub-problems which can be solved or approximated efficiently and are of interests from real applications is an open issue for this problem.

For the another maximization problem, MAX INDEXING, we have shown that the problem is intractable even if two sets of strings are completely the same one, and in general APX-hard. Then we have proposed a polynomial-time algorithm $Greedy_k$ that finds a $k$-indexing, and proved that it achieves the worst-case performance ratio $k/(k-1)$.

The performance ratio of $Greedy_k$ seems far from good, especially when $k$ is small. In recent results [7, 24], remarkable improvements of approximation algorithms for MAX SAT and MAX CUT have been presented. Although both ideas of $Greedy_k$ and the algorithm for MAX SAT in [24] comes from the same greedy algorithm by Johnson [9], the techniques introduced in [24] cannot be directly applied to improve our algorithm. Improvements of performance ratios of algorithms for MAX INDEXING must be considered as forthcoming issues.

Additionally, we have extended $Greedy_k$ to deal with problem MAX $m$-TUPLE $k$-INDEXING, and showed that it guarantees a constant performance ratio. Theorem 16 states that the guaranteed performance ratio rises with the size of the indexing alphabet. This seems curious, since clearly using more indices makes distinguishing tuples easier. In some sense, this result only states that if more and more indices are needed, then distinguishing tuples is more and more difficult. If we can find other subproblems that permit a larger factor $r_0$, we can fill the gap between the statement and this observation.

## Acknowledgements

I would like to thank Magnús M. Halldórsson and Hiroki Arimura for their helpful suggestions and comments. I also thank anonymous referees for their comments which improved this paper.

## References

[1] S. Abbasi, A. Sengupta, An $O(n \log n)$ algorithm for finding dissimilar strings, Inform Process. Lett. 62 (1997) 135–139.

[2] S.E. Altschul, W. Gish, W. Miller, E.W. Mayers, D.J. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (1990) 403–410.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and hardness of approximation problems, in: Proc. 33rd Annual Symp. on Foundations of Computer Science, IEEE, 1992, pp. 14–23.

[4] G. Ausiello, P. Crescenzi, M. Protasi, Approximate solution of NP optimization problems, Theoret. Comput. Sci. 150 (1995) 1–55.

[5] M. Bellare, O. Goldreich, M. Sudan, Free bits, PCPs and non-approximability – towards tight results, in: Proc. 36th Annual Symp. on Foundations of Computer Science, IEEE, New York, 1995, pp. 422–431.

[6] M.O. Dayhoff, R.M. Schwartz, B.C. Orcutt, A model of evolutionary change in proteins, in: Atlas of Protein Sequence and Structure, Ch. 22, National Biomedical Research Foundation, Washington, 1978, pp. 345–358.

[7] M.X. Goemans, D.P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problem using semidefinite programming, J. Assoc. Comput. Mach. 42 (1995) 1115–1145.
[8] S. Henikoff, J.G. Henikoff, Amino acid substitution matrices from protein blocks, Proc. Natl. Acad. Sci. USA 89 (1992) 10915–10919.
[9] D.S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Systems Sci. 9 (1974) 256–278.
[10] P. Jokinen, E. Ukkonen, Two algorithms for approximate string matching in static texts, in: Proc. Mathematical Foundations in Computer Science, Springer, Berlin, 1991, pp. 240–248.
[11] K. Karplus, Regularizers for estimating distributions of amino acids from small samples, Technical Report UCSC-CRL-95-11, University of California, Santa Cruz, 1995.
[12] S. Khanna, R. Motwani, M. Sudan, U. Vazirani, On syntactic versus computational views of approximability, in: Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE, New York, 1994, pp. 819–830.
[13] J. Kyte, R.F. Doolittle, A simple method for displaying the hydropathic character of protein, J. Mol. Biol. 157 (1982) 105–132.
[14] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1993.
[15] C.H. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes, J. Comput. Systems Sci. 43 (1991) 425–440.
[16] J. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81–106.
[17] S. Shimozono, An approximation algorithm for alphabet indexing problem, in: Proc. 6th Annual Internat. Symp. on Algorithms and Computation, Lecture Notes in Computer Science, vol. 1004, Springer, Berlin, 1995, pp. 2–11.
[18] S. Shimozono, S. Miyano, Complexity of finding alphabet indexing, IEICE Trans. Inform Systems E78-D (1995) 13–18.
[19] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, S. Arikawa, Knowledge acquisition from amino acid sequences by machine learning system BONSAI, Trans. Inform Proc. Soc. Japan 35 (1994) 2009–2018.
[20] T. Shinohara, Private communication, 1995.
[21] R.F. Smith, T.F. Smith, Automatic generation of primary sequence patterns from sets of related protein sequences, Proc. Natl. Acad. Sci. USA 87 (1990) 118–122.
[22] W.R. Taylor, The classification of amino acid conservation, J. Theoret. Biol. 119 (1986) 205–218.
[23] W.J. Wilbur, D.J. Lipman, Rapid similarity searches of nucleic acid and protein data banks, Proc. Natl. Acad. Sci. USA 80 (1983) 726–730.
[24] M. Yannakakis, On the approximation of maximum satisfiability, J. Algorithms 17 (1994) 475–502.