数理解析研究所講究録
第876巻 1994年 1-15

Complexity of Finding Alphabet Indexing

Shinichi Shimozono
下薗 真一
Department of Control Engineering and Science,
Kyushu Institute of Technology,
Iizuka 820, Japan

Satoru Miyano
宮野 悟
Research Institute of Fundamental Information Science, 結合場工学研究所
Kyushu University 33,
Fukuoka 812, Japan

Abstract

An alphabet indexing is a mapping from an alphabet Σ to a smaller size alphabet, such that no essential information is lost from original data strings over Σ by these replacements of symbols. We define an indexing and analyze the hardness of finding an indexing for coping with strings over an alphabet of a large size, such as sequences of amino acid residues representing proteins. We show the P-completeness of a greedy algorithm that reduces the size of indexing alphabet and the PLS-completeness of finding an indexing by a local search algorithm applied for a bioinformatical knowledge acquisition.

Key words: Algorithm and Computational Complexity

1 Introduction

Given a pair of sets of strings, an alphabet indexing is a mapping from the alphabet forming the strings to another alphabet for reducing the number of symbols without confusing any string in one set with those in the other set. Developing efficient algorithms for finding this mapping is important to knowledge acquisitions from large amounts of data consisting of the large number of symbols, such as genome sequences.

This work is partly supported by Grant-in-Aid for Scientific Research on Priority Areas, "Genome Informatics" from the Ministry of Education, Science and Culture, Japan.

Arikawa et al.[1] developed an efficient machine learning method to find a bioinformatical hypothesis for identifying "transmembrane domains" of proteins[2, 4, 8]. Their algorithm finds a decision tree over regular patterns for a given pair of positive examples and negative examples, each of which consist of sequences chosen from transmembrane domain regions of proteins and those from the other parts. Moreover, according to the hydropathy index of Kyte and Doolittle[4], they classified 20 symbols of amino acid residues into three categories, and then transformed the positive and the negative examples to those consisting of three symbols. The hydropathy index of an amino acid residue is a number varying from -4.5 to 4.5, and this is known to play an essential role in transmembrane domain identification[4]. They succeeded to discover a hypothesis recognizing transmembrane domain sequences with high accuracy by using the sequences in PIR database[5].

Inspired by this success, we consider the problem for finding an *alphabet indexing*. Let P and Q be the disjoint sets of strings over an alphabet Σ, and let Γ be another alphabet with $|\Gamma| < |\Sigma|$. An alphabet indexing ψ for P, Q by Γ is a mapping from Σ to Γ for transforming the strings in P and Q by replacing any symbol $\sigma \in \Sigma$ by $\psi(\sigma)$, such that the transformation does not produce no strings in both two sets. While Arikawa et al. have used the hydropathy index of Kyte and Doolittle, we can apply some algorithm to find an alphabet indexing for the transmembrane domain identification problem. Even for unknown characteristics of other protein identification problems, we can expect that the algorithm may discover an efficient alphabet indexing.

From this point of view, we define an alphabet indexing and show that the problem of finding a consistent indexing by an alphabet of fixed size is computationally hard. Next we consider another problem for minimizing the size of the indexing alphabet with retaining transformed sets having no same strings. For this problem we propose a greedy algorithm and show its P-completeness. Coping with the hardness of these problems, we consider an approximation scheme for finding an alphabet indexing in

section 4. First, we define a pseudo-indexing, which allows some overlaps between the transformed sets of strings, and a combinatorial optimization problem to find a good "pesudo-indexing" for data and a learning algorithm as follows:

(1) Any pseudo-indexing is a feasible solution.

(2) The performance measure of an indexing is given by the accuracy of a hypothesis produced by the learning algorithm for the transformed inputs.

Then we show a local search strategy for searching pseudo-indexing, which has been developed and succeeded for the bioinformatical experiments[7]. We analyze its computational complexity according to the formulation of the class PLS defined by Johnson et al.[3] and show the PLS-completeness of a local search version of this problem. We end with some conclusion.

2 Indexing is intractable

First, we define an alphabet indexing, and show that the problem of deciding whether the disjoint sets of strings have a consistent indexing or not is computationally hard, even for by two symbols.

Let Σ and Γ be finite alphabets, and let ψ be a mapping from Σ to Γ. The homomorphism $\tilde{\psi}(s)$ for $s \in \Sigma^*$ and $\tilde{\psi}(S)$ for $S \subseteq \Sigma^*$ are defined by $\tilde{\psi}(s_1 \cdots s_n) = \psi(s_1) \cdots \psi(s_n)$ and $\tilde{\psi}(S) = \{\tilde{\psi}(s) \mid s \in S\}$. We denote $\tilde{\psi}(S)$ for $S \subseteq \Sigma^*$ by $\tilde{S}$ if there is no ambiguity. Let P, Q be the disjoint sets of strings over Σ, and let Γ be an alphabet with $|\Gamma| < |\Sigma|$. Then we say a mapping ψ from Σ to Γ is an alphabet indexing of Σ by Γ for P and Q if ψ satisfies $\tilde{P} \cap \tilde{Q} = \emptyset$.

Definition 1. Alphabet Indexing Problem.

Instance: A finite alphabet Σ, two disjoint finite sets of strings P, Q over Σ, and an indexing alphabet Γ with $|\Sigma| > |\Gamma|$.

Question: Is there an indexing of Σ by Γ for P and Q?

This condition preserves examples in the positive set and the negative set being consistent by the transformations. However, unfortunately, the following theorem holds.

Theorem 1. The alphabet indexing problem is NP-complete, even if the size of the indexing alphabet is two and the length of strings are bounded to four.

Proof. We give a reduction from 3-SAT to this problem. Given a 3-CNF formula F with variables $x_1, \cdots, x_n$ and clauses $c_1, \cdots, c_m$, we build an instance of consistent alphabet indexing problem $\Pi = (\Sigma, \Gamma, P, N)$ as follows.

(i) $\Sigma = \{x_i, \bar{x}_i \mid 1 \leq i \leq n\} \cup \{\mathtt{t}, \mathtt{f}\}$, $\Gamma = \{0, 1\}$.

(ii) Let $c_j = (l_1^j \vee l_2^j \vee l_3^j)$ be a clause in F. Then $P = \{x_i\bar{x}_i x_i\bar{x}_i \mid 1 \leq i \leq n\} \cup \{\mathtt{tftf}\} \cup \{l_1^j l_2^j l_3^j \mathtt{f} \mid 1 \leq j \leq m\}$, $Q = \{x_i x_i x_i x_i \mid 1 \leq i \leq n\} \cup \{\mathtt{ffff}\}$.

Then F is satisfiable if and only if Π has a strictly consistent indexing.

($\Rightarrow$) If F is satisfiable, there exists an assignment $b = (b_1, \ldots, b_n) \in \{0,1\}^n$ satisfying F. Then we have an indexing ψ computed as follows: (i) Let $\psi(\mathtt{t}) = 1, \psi(\mathtt{f}) = 0$, and (ii) Let $\psi(x_i) = b_i, \psi(\bar{x}_i) = 1 - b_i$ for all $1 \leq i \leq n$.

Since the assignment b satisfies F, we have at least one literal with value 1 for all clauses. Thus, for any string corresponding to the clause c_i, ψ maps at least one of three literal symbols to 1. So the all string $\tilde{\psi}(l_1^i l_2^i l_3^i \mathtt{f})$ in P is not identical to $\tilde{\psi}(\mathtt{ffff}) = 0000$ in Q.

($\Leftarrow$) If Π has an indexing ψ, we compute an assignment b satisfying F as follows. If $\psi(\mathtt{t}) = 1$, then $b = \psi(x_1), \ldots, \psi(x_n)$; otherwise, $b = \psi(\bar{x}_1), \ldots, \psi(\bar{x}_n)$. Since ψ satisfies $\tilde{\psi}(x_i\bar{x}_i) \neq \tilde{\psi}(x_i x_i)$, $\tilde{\psi}(\mathtt{tf}) \neq \tilde{\psi}(\mathtt{tt})$, the boolean value of x_i and $\bar{x}_i$ for all $1 \leq i \leq n$ (also $\mathtt{t}$ and $\mathtt{f}$) assigned by ψ satisfies $\neg x_i = \bar{x}_i$ (and $\neg\mathtt{t} = \mathtt{f}$). Next, since $\tilde{\psi}(l_1^i l_2^i l_3^i \mathtt{f}) \neq \tilde{\psi}(\mathtt{ffff})$ for any clause c_i with $1 \leq i \leq m$, at least one of the three literals

is not same with $\psi(\mathtt{f})$. Thus any clause in F has at least one liteal with value 1 for the assignment b, so it satisfies F. □

This theorem states that the problem considering whether the transformed strings are identical or not is NP-complete. In such case, only the strings having the same length have to be considered for finding an alphabet indexing. But the above result can be easily extended to the hardness of more general problem that considers the identity of substrings as follows.

Let P, Q be the sets of string over Σ, and let Γ be an alphabet with $2 \leq |\Gamma| < |\Sigma|$. Let P' and Q' be the sets of strings defined as follows.

$$P' = \{s' \mid s' \text{ is a substring of } s \in P \text{ with } |s'| > 3\}$$

$$Q' = \{s' \mid s' \text{ is a substring of } s \in Q \text{ with } |s'| > 3\}$$

Then, by applying theorem 1, finding an alphabet indexing of Σ by Γ for $P' - Q'$ and $Q' - P'$ is NP-complete.

Collorary 1. Finding an alphabet indexing that allows to distinguish the pair of sets of strings by their containments of substrings is NP-hard if the substrings for classification are restricted to have length longer than 3.

3 Reducing the size of an indexing alphabet

In the section 2, we discussed the problem for finding an indexing by an alphabet of fixed size and showed intractability of the problem. In this section, we consider another problem for finding the minimum size of alphabets that can have an alphabet indexing, and for coping with this problem, we design a greedy algorithm that reduces the size of the alphabet forming the strings.

Definition 2. Let Σ be an alphabet and let P, Q be disjoint sets of strings over Σ.

Find the smallest indexing alphabet $\Gamma \subseteq \Sigma$ and an alphabet indexing $\psi : \Sigma \to \Gamma$ satisfying $\tilde{\psi}(P) \cap \tilde{\psi}(Q) = \emptyset$.

Since this problem is NP-hard by the theorem 1, we ought to consider an approximation scheme for minimizing the size of the indexing alphabet. The following greedy heuristic algorithm may cope with this situation.

From now on, we assume an order on Σ.

Algorithm *Greedy-Reducing*

Let $\Gamma = \Sigma$ and $\psi(\gamma_i) = \gamma_i$ for all $\gamma_i \in \Sigma$.

repeat

for each $(a, b) \in \Gamma \times \Gamma$ with $a \neq b$:

Let $D \leftarrow \{\sigma_i \mid \psi(\sigma_i) = a\}$, and let $\varphi_{(a,b)}$ be a mapping obtained by letting $\psi(\sigma_i) = b$ for all $\sigma_i \in D$.

end.

Choose $\varphi_{(a,b)}$ satisfying $\tilde{\varphi}_{(a,b)}(P) \cap \tilde{\varphi}_{(a,b)}(Q) = \emptyset$ and minimizing $|\tilde{\varphi}_{(a,b)}(P) \cup \tilde{\varphi}_{(a,b)}(Q)|$. /* *break ties arbitrary.* */

$\psi \leftarrow \varphi_{(a,b)}$.

$\Gamma \leftarrow \Gamma - \{a\}$.

until no more replacements can not be found.

Definition 3. Greedy Alphabet Reducing Problem

Given a finite alphabets Σ with $|\Sigma| > 2$, two finite disjoint sets $P, Q \subseteq \Sigma^*$ and a positive integer $K \leq |\Sigma|$, is the size of Γ obtained by *Greedy-Reducing* smaller than K?

For this problem, we obtained the following result that asserts that the algorithm cannot be efficiently parallelizable unless NC = P.

Theorem 2. The greedy alphabet reducing problem is P-complete.

Proof. We give a reduction from circuit value problem to greedy alphabet reducing problem. A *boolean circuit* is a sequence $\gamma = (\gamma_1(0,0), \ldots, \gamma_m(k,l))$, where each $\gamma_i(0,0)$ is an input gate assigned a value 1 or 0 and $\gamma_k(i,j)$ is a NAND gate $\gamma_k = \neg(\gamma_i \wedge \gamma_j)$ for some $0 < i \leq j < k \leq m$. The problem is to determin whether the last gate γ_m outputs the value 1 or 0. It is well known that this problem is P-complete.

Given a circuit $\gamma = (\gamma_1(0,0), \ldots, \gamma_m(k,l))$, we construct a greedy alphabet reducing problem as follows.

(i) $\Sigma = \{0, 1, \gamma_1, \ldots, \gamma_m\}$.

(ii) P and Q consist of the following:

(a) For all $1 \leq i < m$, P includes the strings $001\langle i\rangle, 011\langle i\rangle, 101\langle i\rangle, 110\langle i\rangle$, where $\langle i\rangle$ is the binary string coding i with fixed length $\lceil \log m \rceil$.

(b) For each input gate $\gamma_k(0,0)$ assigned $b_k \in \{0,1\}$, P includes $\bar{b}_k\bar{b}_k\gamma_k\langle 1\rangle$, $\cdots$, $\bar{b}_k\bar{b}_k\gamma_k\langle m-k\rangle$, where $\bar{b}_k$ is the complement of b_k.

(c) For each NAND gate $\gamma_k(i,j)$ of $1 < k < m$, P includes $\gamma_i\gamma_j\gamma_k\langle 1\rangle$, $\cdots$, $\gamma_i\gamma_j\gamma_k\langle m-k\rangle$.

(d) For the last NAND gate $\gamma_m(i,j)$, P includes $\gamma_i\gamma_j\gamma_m\langle 0\rangle$, $001\langle 0\rangle$, $011\langle 0\rangle$, $101\langle 0\rangle$ and Q includes $000\langle 0\rangle$, $010\langle 0\rangle$, $100\langle 0\rangle$, $111\langle 0\rangle$, $110\langle 0\rangle$.

(iii) $k = 3$.

Notice that the symbols 1 and 0 can not be identified. From now on, we refer all symbols identified with 1 to $\hat{1}$ and with 0 to $\hat{0}$.

We show that each γ_i for $1 \leq i < m$ will be identified with 1 or 0 (i.e. $\psi(\gamma_i) = \psi(0)$ or $\psi(\gamma_i) = \psi(1)$) at i-th iteration of *Greedy-Reducing* algorithm, and if and only if

the last gate of γ have a value 1 for the input to γ, γ_m can be identified with 1 (i.e. $\psi(\gamma_m) = \psi(1)$).

Base step: Since the strings having γ_1 as the third symbol consist of only 1, 0 and γ_1, the algorithm can reduce $m-1$ strings of P by identifying γ_1 with b_1.

In contrast, by identifying γ_i with any other symbol for any $i > 1$, at most $m-i < m-1$ strings can be reduced, since there are only $m-i$ strings that have γ_i as the third symbol. Notice that any string having γ_i as the first or the second symbol can not be reduced.

Thus γ_1 will be identified with b_1 in the first iteration of the algorithm.

Induction step: Assume that only symbols $\gamma_1, \cdots, \gamma_{j-1}$ are already identified with either $\hat{1}$ or $\hat{0}$ (i.e., there is a symbol $c \in \Gamma$ such that $\psi(\gamma_i) = \psi(c)$ for $1 \leq i < j$).

Since the strings having γ_j as the third symbol consist of only $\hat{1}$ and $\hat{0}$ except γ_j, the algorithm can reduce $m-j$ strings of P by identifying γ_j with the symbol expressing the boolean value for γ_j. On the other hand, for any $k > j$, P contains strings having γ_k as the third symbol only $m-k < m-j$, and the strings having γ_k as the first or second symbol must have still not identified symbol γ_l with $l > k$ as the third symbol. So at most $m-k < m-j$ strings can be reduced by identifying γ_k with other symbol.

Therefore the algorithm identifies γ_j with the symbol expressing the value of the gate γ_j for its inputs. □

4 A local search strategy for finding an indexing

In this section, we introduce an approximated alphabet indexing, namely, a pseudo-indexing, which allows the transformed sets having intersection. Then we consider the problem of combinatorial optimization version and a local search algorithm for finding a pseudo-indexing.

Let Σ be a finite alphabet. Let A be a polynomial-time algorithm that outputs a

polynomial-time computable function $h : \Sigma^* \to \{0,1\}$ for input $P, Q \subseteq \Sigma^*$. We say that A is *robust* if, for any input P and Q, A produces h satisfying $P - Q \subseteq L_h$ and $Q - P \subseteq \overline{L_h}$, where $L_h = \{s \in \Sigma^* \mid h(s) = 1\}$ and $\overline{L_h} = \{s \in \Sigma^* \mid h(s) = 0\}$. Then we can use outputs of A as a classifier that completely separates strings in the input except those in $P \cap Q$. For a such algorithm, we can define a performance measure of a pseudo-indexing by evaluating accuracy of hypothesis over P, Q obtained for chosen and transformed training examples $\tilde{P}', \tilde{Q}'$. This can be formulated as a combinatorial optimization problem.

From this point of view, we applied a local search strategy to the knowledge acquisition system[7]. This algorithm starts from any pseudo-indexing and iterates finding a better indexing from its neighborhoods, until no more improved one can be found, and then outputs a pseudo-indexing and a decision tree that classifies a string. Informally, a decision tree is a rooted tree that consists of nodes with queries and leafs with classifications. To classify an input, we choose nodes and evaluate their queries, from the root to the leaves according to the answers. When we reach to a leaf, we classify the input according to the label on that leaf. To each query nodes, Arikawa et al.[1] attached simple regular patterns. This learning algorithm is robust and outputs hypothesis for any training examples.

Now we formulate a local search strategy for the knowledge acquisition system by the polynomial-time local search problem (PLS)[3]. First we review the definitions of this class PLS.

Definition 4. Let Λ be a finite alphabet. A *polynomial-time local search problem* L is either a maximization or minimization problem specified as follows:

(1) $I(L) \subseteq \Lambda^*$, a set of *instances.*

(2) For each instance $\phi \in I(L)$, we associate it with the following:

(i) $S_L(\phi)$, a finite subset of Λ^* called the set of *feasible solutions.* An element s

in $S_L(\phi)$ is called a *solution* of ϕ.

(ii) $N_L(\phi, s)$, a subset of $S_L(\phi)$ called the *neighbors* of s, where s is a solution in $S_L(\phi)$. We call a solution in $N_L(\phi, s)$ a *neighbor* of s.

(iii) $C_L : I(L) \times S_L(\phi) \rightarrow \mathbf{Z}$, the *cost function* for $S_L(\phi)$, where $\mathbf{Z}$ is the set of nonnegative integers. The value $C_L(\phi, s)$ is called the *cost* of s.

We require that $I(L), S_L, N_L$ and C_L are polynomial-time computable with respect to $|\phi|$. A solution $s \in S_L(\phi)$ is called *locally optimal* if s has no better neighbors, i.e., $C_L(\phi, s') \leq C_L(\phi, s)$ for all s' in $N_L(\phi, s)$ when L is a maximization problem. In addition, the following two polynomial-time algorithms must exist for L: (i) $Initial_L$, given $\phi \in I(L)$, produces any feasible solution in $S_L(\phi)$, and (ii) $Improve_L$, given $\phi \in I(L)$ and $s \in \Lambda^*$, produces a better neighbor if s is in $S_L(\phi)$ and not locally optimal, otherwise simply returns s.

The algorithms $Initial_L$ and $Improve_L$ allow us to apply the local search algorithm. For the class of PLS problems, the PLS-reductions are defined as follows:

Definition 5. Let L and K be problems in PLS. We say that L is PLS-*reducible* to K if there are polynomial-time computable functions f and g such that for each instance ϕ of L, (i) $f(\phi)$ is an instance of K, (ii) $g(f(\phi), s)$ is a solution of ϕ if s is a solution of $f(\phi)$, and (iii) if $s \in S_K(f(\phi))$ is a locally optimal solution of $f(\phi)$, then $g(f(\phi), s) \in S_L(\phi)$ is also a locally optimal solution of ϕ.

Notice that the PLS-reducibility is transitive, and if we can find locally optimal solutions for a PLS-complete problem in polynomial time then we can also find locally optimal solutions for any PLS problem in polynomial time.

Definition 6. The problem *alphabet indexing by local search* is a maximization problem given by an instance $\Pi = (\Sigma, \Gamma, P, Q, \Omega, P', Q', A)$, where Σ and Γ are finite alphabets with $|\Sigma| > |\Gamma|$, P and Q are sets of strings over Σ, Ω is a weight function

from $P \cup Q$ to nonnegative integers, A is a robust learning algorithm, and P' and Q' are the sets of training examples $P' \subseteq P, Q' \subseteq Q$.

The set of feasible solutions S_Π is the set of all mappings from Σ to Γ. The cost C_Π is defined as follows. Let h be the hypothesis obtained by the algorithm A for the sets of transformed examples $\tilde{P}'$ and $\tilde{Q}'$. Then $C_\Pi(\psi) = f\left(||\tilde{P} \cap L_h||, ||\tilde{Q} \cap \overline{L_h}||\right)$, where $||S|| = \sum_{s \in S} \Omega(s)$, L_h and $\overline{L_h}$ are the language defined by $h : \Sigma^* \to \{0,1\}$ and f is a function from pairs of nonnegative integers to nonnegative integers, $f : \mathrm{Z} \times \mathrm{Z} \to \mathrm{Z}$, satisfying $f(a,b) = f(b,a)$ and $f(a,b) < f(a,c)$ for any $0 \leq a, b < c$.

The neighborhood $N_\Pi(\psi)$ for $\psi \in S_\Pi$ is $N_\Pi(\psi) = \{\phi \in S_\Pi \mid \delta(\phi,\psi) = 1\}$, where $\delta(\phi,\psi) = |\{\sigma \in \Sigma \mid \phi(\sigma) \neq \psi(\sigma)\}|$.

For this problem, the following theorem holds.

Theorem 3. Alphabet indexing by local search is PLS-complete in the weighted case, and finding a locally optimal indexing for the unweighted problem is P-complete.

Proof. In either weighted and unweighted case, polynomial-time local search problem 2-SAT[6] is PLS-reducible to the alphabet indexing by local search. It is known that k-SAT (WEIGHTED k-CNF SATISFIABILITY) for $k \geq 2$ is PLS-complete in the weighted case and P-complete in the unweighted case[6].

An instance of k-SAT is a boolean formula in k-CNF with a nonnegative integer weight on each clause. A solution is a boolean assignment to the variables, and its cost is the sum of the weights of all satisfied clauses. The neighborhoods of a solution are the set of assignments in Humming distance 1. This neighborhood structure is called FLIP. Here we show a PLS-reduction[3] from 2-SAT to AILSP in both weighted and unweighted cases. A PLS-reduction from a PLS-problem L to another PLS-problem M is defined by two polynomial-time computable functions f and g. f is a mapping from L to M and g is a mapping from the pairs of $f(l) \in M$ and a solution s of $f(l)$ to a solution of l, satisfying that $g(f(l), s)$ is a locally optimal solution of l if s is a

locally optimal solution. We first show a reduction in weighted case, then we modify it for unweighted case.

Weighted case:

Given a 2-CNF formula $F(x_1, \cdots, x_n)$ with n variables and m weighted clauses $c_1, \ldots, c_m$, we transform F to an instance of the as follows.

According to the variables of F, we define $\Sigma = \{x_i \mid 1 \leq i \leq n\} \cup \{\mathtt{t}, \mathtt{f}\}$ and $\Gamma = \{0, 1\}$. The sets of strings P, Q are defined as follows. Let $c_j = (l_1^j \vee l_2^j)$ be a clause in F. If l_k^j is a positive literal x_i, let $t_k^j = x_i$ and $f_k^j = \mathtt{t}$. Otherwise, let $t_k^j = \mathtt{f}$ and $f_k^j = x_i$. Then we define $P = \{s(c_j) \mid 1 \leq j \leq m\} \cup \{\mathtt{tttt}\}$ and $Q = \{\mathtt{tftf}\}$, where $s(c_j) = f_1^j t_1^j f_2^j t_2^j$ for any clause c_j with $1 \leq j \leq m$. Let $P' = P$ and $Q' = Q$. Finally, we define the weight function Ω. For each $s(c_j)$ in P, we give the weight of the corresponding clause c_j, and for $\mathtt{tttt}$, we give the weight $\omega + 1$, where ω is the sum of the weights of all clauses in F. We give the weight $2\omega + 2$ to the negative string.

Next we define g, the mapping from solutions to boolean assignments. For any solution ψ of this instance, we compute an assignment for F as follows. If $\psi(\mathtt{t}) = 1$, we give the boolean value represented by $\psi(x_i)$ for each variable x_i with $1 \leq i \leq n$. Otherwise, we give $x_i = \neg\psi(x_i)$. It is clear that this mapping gives a boolean assignment for any pseudo-indexing.

The remaining part of the reduction is to show that if an indexing ψ is locally optimal then g gives a locally optimal assignment of F. To do this, we show that the following lemma holds.

Lemma 1. Any locally optimal indexing satisfies $\psi(\mathtt{t}) \neq \psi(\mathtt{f})$.

Assume that a locally optimal indexing ψ satisfies $\psi(\mathtt{t}) = \psi(\mathtt{f})$. This implies that $\tilde{\psi}(\mathtt{tttt})$ is identical to $\tilde{\psi}(\mathtt{tftf})$, and thus any language must include one of them if and only if it includes the other. If h classifies both those strings in L_h, the cost

of ψ is at most $f(\omega+\omega+1,0)$, and otherwise (both not in L_h) the cost is at most $f(\omega,2\omega+2)$. On the other hand, an indexing ψ' satisfying $\psi'(\mathtt{t}) \neq \psi'(\mathtt{f})$ allows to distinguish $\tilde{\psi}(\mathtt{tttt})$ from $\tilde{\psi}(\mathtt{tftf})$ and has the cost at least $f(\omega+1,2\omega+2)$. Such a solution can be obtained from ψ by flipping $\psi(\mathtt{t})$ or $\psi(\mathtt{f})$, and the cost of ψ' dominates the cost of any indexing that atisfies $\psi(\mathtt{t}) = \psi(\mathtt{f})$. This contradicts the assumption. Therefore any locally optimal indexing satisfies $\psi(\mathtt{t}) \neq \psi(\mathtt{f})$.

Now, we show that if an indexing is locally optimal then the boolean assignment for F obtained by g is also locally optimal. Let ψ be a locally optimal indexing, and let W be the sum of the weights of all strings that have two continuous same symbols in $\tilde{\psi}(P)$, except $\mathtt{tttt}$. All those strings correspond to clauses and have such substrings only if they have a variable symbol x_i satisfying $\tilde{\psi}(\mathtt{t}x_i) = \tilde{\psi}(\mathtt{tt})$ or $\tilde{\psi}(x_i\mathtt{f}) = \tilde{\psi}(\mathtt{ff})$, i.e., the clause has x_i satisfying $\psi(x_i) = \psi(\mathtt{t})$ for a positive literal or $\psi(x_i) = \psi(\mathtt{f})$ for a negative literal. Since ψ is a locally optimal solution, ψ satisfies $\psi(\mathtt{t}) \neq \psi(\mathtt{f})$, and thus any string that has two continuous same symbols can be distinguished from $\mathtt{tftf}$. Therefore, W is equal to the sum of the weights of all clauses satisfied by the assignment ψ, and the cost of ψ is $f(W+\omega+1,2\omega+2)$. Suppose that ψ is locally optimal and the assignment obtained by ψ is not locally optimal. Then the assignment must have a flip to make improved solution with cost $W' > W$. But this implies that ψ has also an improved neighbor, since the same flip for ψ improves the cost to $f(W'+\omega+1,2\omega+2)$. This contradicts the assumpition. Thus the theorem holds in weighted case.

Unweighted case:

Let $F(x_1,\cdots,x_n)$ be a 2-CNF formula with n variables and m unweighted clauses. We transform the unweighted formula to an unweighted alphabet indexing problem as follows.

Let Σ, Γ be the alphabets constructed as in the weighted case. For each clause c_i in

F, we make $s(c_i)$ and add it to P. We add all $\{\mathtt{ttt}\underbrace{\mathtt{ttftf}\cdots\mathtt{tf}}_{i} \mid 1 \leq i \leq m+1\}$ to P, and add all $\{\mathtt{tft}\underbrace{\mathtt{ftftf}\cdots\mathtt{tf}}_{i} \mid 1 \leq i \leq 2m+2\}$ to Q. The mapping of the solutions g is computed as in the weighted case. Then any localy optimal indexing of this instance also satisfies lemma 1. If an indexing ψ satisfies $\psi(\mathtt{t}) = \psi(\mathtt{f})$, the cost of ψ is at most $f(m+m+1, m+1)$ (for h satisfying $\tilde{\psi}(\mathtt{tttt}), \tilde{\psi}(\mathtt{tftf}) \in L_h$) or $f(m, 2m+2)$ (for h satisfying $\tilde{\psi}(\mathtt{tttt}), \tilde{\psi}(\mathtt{tftf}) \notin L_h$.) On the other hand, if $\psi(\mathtt{t}) \neq \psi(\mathtt{f})$, the cost is at least $f(m+1, 2m+2)$. If an indexing is locally optimal, any string corresponding to a clause is distinguished from `tftf` after the transformation if and only if it has two continuous same symbols. Therefore, as in the weighted case, the boolean assignment computed by g is locally optimal if the indexing is locally optimal.

Thus the theorem holds for both weighted and unweighted cases. □

5 Conclusions

We have defined an alphabet indexing which is strictly consistent for positive and negative data. In such case, we have shown that finding an indexing is NP-complete. Additionally, we have shown that the greedy algorithm for reducing the alphabet forming positive and negative data is P-complete. We have also defined the pseudo-indexing as approximated one, and shown that the problem of finding a locally optimal pseudo-indexing by the algorithm in [7] is PLS-complete. Even the strings are unweighted, the result states that finding a locally optimal pseudo-indexing is P-complete.

As we have shown in this paper, finding a good indexing is computationally hard. However, by giving a suitable indexing to sequences of amino acid residues, we can reduce the computation time and spaces. Moreover, a good indexing may reduce the learning spaces or class and simplify the hypotheses. The experimental results in [7] can be considered as its empirical support. However, their theoretical foundations are left as open problems.

References

(1) Arikawa, S., Kuhara, S., Miyano, S., Mukouchi, Y., Shinohara, A. and Shinohara, T., "Machine discovery of a negative motif from amino acid sequences by decision trees over regular patterns", in *Proc. Int. Conf. Fifth Generation Computer Systems*, pp. 618–625, 1992.

(2) Arikawa, S., Kuhara, S., Miyano, S., Shinohara, A. and Shinohara, T., "A learning algorithm for elementary formal systems and its experiments on identification of transmembrane domains", in *Proc. 25th Hawaii International Conference on System Sciences*, pp. 675–684, 1992.

(3) Johnson, D. S., Papadimitriou, C. H. and Yannakakis, M., "How easy is local search?", *J. Comput. Sys. Sci.*, vol. 37, pp. 79–100, 1988.

(4) Kyte, J. and Doolittle, R. F., "A simple method for displaying the hydropathic character of protein", *J. Mol. Biol.*, vol. 157, pp. 105–132, 1982.

(5) "Protein identification resource", National Biomedical Research Foundation.

(6) Schäffer, A. A. and Yannakakis, M., "Simple local search problems that are hard to solve", *SIAM J. Comput.*, vol. 20, no. 1, pp. 56–87, 1991.

(7) Shimozono, S., Shinohara, A., Shinohara, T., Miyano, S., Kuhara, S. and Arikawa, S., "Finding alphabet indexing for decision trees over regular patterns", in *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, pp. 763–772. IEEE COMPUTER SOCIETY, 1993.

(8) Yanagihara, N., Suwa, M. and Mitaku, S., "A theoretical method for distinguishing between soluble and membrane proteins", *Biophysical Chemistry*, vol. 34, no. 1, pp. 69–77, 1989.