Constructing a Grammar for Infering Common Structure of Two Strings and Its Application to Compression

Munehito Binou[1], Keisuke Otaki[1], Madori Ikeda[1], Ryo Yoshinaka[1] and Akihiro Yamamoto[1]

Graduate School of Informatics, Kyoto University, Japan

Abstract. We propose a method to infer a common structure of a given pair of strings by constructing a grammar for generating exactly the pair. This method can be used as a method of grammar-based compression for two strings. We transform a given pair into a grammar whose production rules are generated based on a common subsequence of the two strings. The size of the grammar is often smaller than that of the original pair, so our method works as compression. For the non-terminal symbol X appearing in the right hand side of each production rule has the property that X corresponds to a pair of strings and a non-terminal symbol Y different from X must correspond to another pair. Therefore we assume that every non-terminal is indexed with a pair of strings. Production rules are recursively generated. We applied our method to pairs of cDNA sequences, and confirm that we successively generate production rules for non-terminal symbols recursively and construct grammar smaller than given pairs.

1 Introduction

In this study, we propose a method to extract a common structure of a given pair of strings. The structure represented in the form of a grammar for generating exactly the given pair. We also intend that the grammar should be smaller than the size of the given strings so that our method could be regarded as a grammar-based compression.

We are inspired by observation that a grammar generated by a grammar-based compression algorithm represents a structure of every input string. Grammar-based compression, which is a kind of lossless compression, was proposed by Yang and Kieffer [6]. It is to construct a grammar for a given string by integrating substrings appearing repeatedly in the string into production rules. In other words, by eliminating the redundancy of subsequences appearing many times, it reduces the original data size. With the obtained grammar can be regarded as a structure of the input string, we can construct exact one parse tree for the string. This means that we can identify the grammar with the parse tree, and this is why we regard the grammar as a structure of a string.

Many grammar-based compression algorithms for a single string have been studied. Examples of grammar-based compression algorithms are SEQUITUR [4] and Re-pair [2]. They construct a context-free grammar for generating exactly one given string and represent a structure of the given string by its production rules. Two parse trees of the grammars obtained by which SEQUITUR processed two strings, *acdacda* and *bcdbcbc*, are shown in Fig. 1, where S, A and B are non-terminal symbols. The structures of two parse trees in Fig. 1 are different. So it is not guaranteed that constructed grammars represent some common structure of two strings.

In order to overcome the problem, we introduce a *pairwise context-free grammar* that generates a pair of two strings. Our algorithm constructs a pairwise context-free grammar for generating exactly a pair of given two strings. The grammar represents a common structure of the given two strings by its production rules.

In the following section we introduce a pairwise context-free grammar for implementing our method. The method is presented in Section 3, and an experimental result is shown in Section 4. We give our conclusion in Section 5.

2 Pairwise Context-Free Grammar

We define *pairwise context-free grammars* (*Pairwise-CFG*, for short). A grammar of this type can be regarded as a synchronous context-free grammar [1].

Let N and Σ be non-empty finite sets satisfying $N \cap \Sigma = \emptyset$. Every element of N is called a *non-terminal symbol*, and every element of Σ is a *terminal symbol*. A rule is an expression of the form $X \to \langle u, v \rangle$ or $X \to \alpha$ where $X \in N$, $u, v \in \Sigma^+$ and $\alpha \in (\Sigma \cup N)^+$. We call the former *a separate rule* and

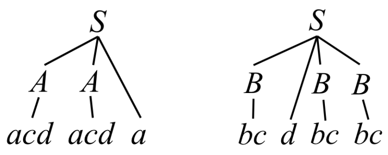


Fig. 1. Parse trees obtained by SEQUITUR

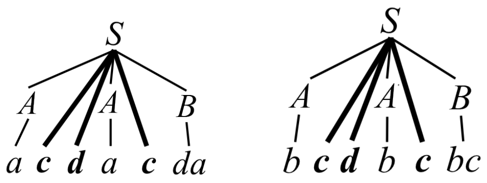


Fig. 2. Parse trees constructed based on a common subsequence

the latter *a common rule*. Every element of a finite set P is a rule. A Pairwise-CFG is $G = (N, \Sigma, P, S)$, where $S \in N$ is called the *start symbol*.

Let a Pairwise-CFG $G = (N, \Sigma, P, S)$, and s, t, s' and $t' \in (\Sigma \cup N)^*$. For two pairs $\gamma = \langle sXt, s'Xt' \rangle$ and $\delta = \langle sut, s'vt' \rangle \in (\Sigma \cup N)^+ \times (\Sigma \cup N)^+$, if there is a separate rule $X \to \langle u, v \rangle \in P$, we write $\gamma \underset{G}{\Rightarrow} \delta$. In addition, for two pairs $\gamma = \langle sXt, s'Xt' \rangle$ and $\delta = \langle s\alpha t, s'\alpha t' \rangle \in (\Sigma \cup N)^+ \times (\Sigma \cup N)^+$, if there is a common rule $X \to \alpha \in P$, we write $\gamma \underset{G}{\Rightarrow} \delta$. If there is a sequence of pairs $w_0, w_1, \ldots, w_n (n \geq 0)$ which satisfies the following condition for $\gamma, \delta \in (\Sigma \cup N)^+ \times (\Sigma \cup N)^+$, we sometimes write $\gamma \overset{*}{\underset{G}{\Rightarrow}} \delta$ and we call it a derivation of δ from γ.

$$\gamma = w_0 \underset{G}{\Rightarrow} w_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} w_n = \delta.$$

Our algorithm constructs a Pairwise-CFG which generates exactly one pair. That is, there is exactly one pair $\langle \sigma_1, \sigma_2 \rangle \in \langle \Sigma^+, \Sigma^+ \rangle$ which satisfies $\langle S, S \rangle \overset{*}{\underset{G}{\Rightarrow}} \langle \sigma_1, \sigma_2 \rangle$. An example of a Pairwise-CFG G and its derivation is shown below.

Example 1. Let $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$ be a Pairwise-CFG, where P consists of the following three rules:

$$S \to AcdAcB, \quad A \to \langle a, b \rangle, \quad B \to \langle da, bc \rangle.$$

Then the following holds:

$$\begin{aligned} \langle S, S \rangle &\underset{G}{\Rightarrow} \langle AcdAcB, AcdAcB \rangle \\ &\underset{G}{\Rightarrow} \langle acdAcB, bcdAcB \rangle \\ &\underset{G}{\Rightarrow} \langle acdacB, bcdbcB \rangle \\ &\underset{G}{\Rightarrow} \langle acdacda, bcdbcbc \rangle. \end{aligned}$$

3 Compression by Constructing a Pairwise Context-Free Grammar

From a given pair of strings our algorithm constructs a grammar by recursively discovering common subsequences. Two parse trees of a grammar constructed based on a common subsequence are shown in Fig. 2. The trees are identical except the leaves that are obtained by the separate rules, and so the trees obtained by deleting such leaves and edges for them could be regarded as a common structure of those strings. In this way, we can infer a common structure of two strings.

A common subsequence of two strings can be found when a sequence alignment of the two strings is fixed. A sequence alignment, which has been used in the analysis of DNA sequences, is to insert gap symbols in the two or more strings so that the same symbols in the given two strings are aligned in the same position.

Two examples of alignment for two strings, *acdacda* and *bcdbcbc* are shown in Fig. 3. In general, a sequence alignment does not have a unique solution. Their common subsequences *cdc* and *dc* can be found in Fig. 3(a) and Fig. 3(b), respectively.

a	$\boldsymbol{c}$	$\boldsymbol{d}$	a	$\boldsymbol{c}$	d	a
b	$\boldsymbol{c}$	$\boldsymbol{d}$	b	$\boldsymbol{c}$	b	c

(a)

a	c	$-$	$\boldsymbol{d}$	a	$\boldsymbol{c}$	d	a
$-$	b	c	$\boldsymbol{d}$	b	$\boldsymbol{c}$	b	c

(b)

Fig. 3. Sequence alignment

Every rule in a grammar constructed by our method from an input pair $\langle \sigma_1, \sigma_2 \rangle$ is generated based on a common subsequence of the input pair. Each non-terminal X derives just a pair $\langle u, v\rangle \in \Sigma^+ \times \Sigma^+$ and a non-terminal Y different from X must derive a pair different from $\langle u, v \rangle$. Therefore we assume that every non-terminal is indexed with a pair $\langle u, v\rangle \in \Sigma^+ \times \Sigma^+$. Rules are recursively generated. One possible rule for a non-terminal $X_{\langle u,v\rangle}$ is $X_{\langle u,v\rangle} \to \langle u, v\rangle$. We call $\langle u, v\rangle$ the *separate candidate* for $\langle u, v\rangle$. Other possible rules have the form $X_{\langle u,v\rangle} \to w_0 X_{\langle u_1,v_1\rangle} w_1 \cdots X_{\langle u_n,v_n\rangle} w_n$ where $w_0, w_n \in \Sigma^*$, $w_i \in \Sigma^+$ for $i = 1, \ldots, n-1$, $X_{\langle u_i,v_i\rangle} \in N$ for $i = 1, \ldots, n$, $u = w_0 u_1 w_1 \cdots w_n$, $v = w_0 v_1 w_1 \cdots w_n$ such that $w_0 w_1 \ldots w_n \in \Sigma^+$. We call $w_0 X_{\langle u_1,v_1\rangle} w_1 \cdots X_{\langle u_n,v_n\rangle} w_n$ a *common candidate* for $\langle u, v\rangle$. Rules for non-terminals $X_{\langle u_i,v_i\rangle}$ are recursively generated. A candidate is either a separate or common candidate.

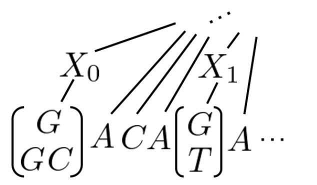
Note that there can be two or more candidates for two strings by our method. In such a case, we have to choose one among them. A criterion for choosing one is based on either compression rate and another is based on computation time. In the case of focusing on compression rate, we assign a score on each candidate we find and choose a rule which has the highest score. In focusing on computation time, we simply pick the one firstly found. In the latter case, we search for a longest common candidate. If no common candidate is found, we take the separate candidate.

Our algorithm is shown below.

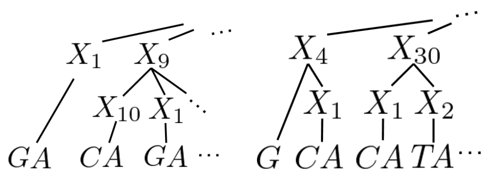
Algorithm 1

```
Input: Given two strings $\sigma_1, \sigma_2$
Output: A finite set $P$ of rules for a Pairwise-CFG
  $P \leftarrow \emptyset$
  $Open \leftarrow \{X_{\langle \sigma_1,\sigma_2\rangle}\}$
  $Closed \leftarrow \emptyset$
  while $Open \neq \emptyset$ do
    Select any element $X_{\langle u,v\rangle}$ from $Open$.
    $Open \leftarrow Open - \{X_{\langle u,v\rangle}\}$
    $p_{\langle u,v\rangle} \leftarrow$Candidate$(u, v)$
    $P \leftarrow P \cup \{X_{\langle u,v\rangle} \to p_{\langle u,v\rangle}\}$
    $Closed \leftarrow Closed \cup \{X_{\langle u,v\rangle}\}$
    for all $X_{\langle u',v'\rangle}$ appearing in $p_{\langle u,v\rangle}$ do
      if $X_{\langle u',v'\rangle} \notin Open$ and $X_{\langle u',v'\rangle} \notin Closed$ then
        $Open \leftarrow Open \cup \{X_{\langle u',v'\rangle}\}$
      end if
    end for
    $p_{\langle u,v\rangle} \leftarrow \lambda$
  end while
  return $P$
```

In the algorithm u and v are substrings of σ_1 and σ_2 respectively and u' and v' are those of u and v. A candidate for two strings, u and v, is denoted by $p_{\langle u,v\rangle}$ and the non-terminal corresponding to $p_{\langle u,v\rangle}$ is $X_{\langle u,v\rangle}$. Every element in the set *Open* is a non-terminal symbol corresponding to a rule that has not been in P, and every element in *Closed* is a non-terminal corresponding to a rule in P. A function Candidate(u, v) returns a candidate for two given strings u and v by focusing on either compression rate or computation time.



(a) A part of a parse tree obtained by Method 2



(b) A part of parse trees obtained by SEQUITUR

Fig. 4.

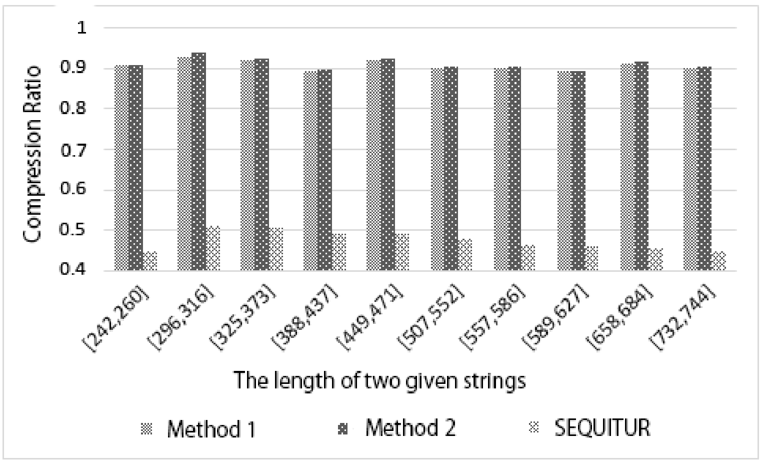


Fig. 5. Compression Ratio

4 Experiment

We downloaded twenty cDNA data of mice randomly chosen from NBRP[1] for our experiments. We sorted the data according to their length, divided them into ten pairs, and applied our method to them. We implemented our algorithm in Python, and applied it to pairs of cDNA sequences in the following environments.

- OS : Windows7
- CPU : Intel(R) Core(TM) i7-4770 3.40Ghz
- Memory : 8.00 GB RAM

We let Method 1 and Method 2 respectively stand for methods focusing on compression rate and computation time in choosing common subsequences. In Method 1 a candidate is selected whose score is the highest among 100000 candidates.

By checking the outputs, which are parse trees, and we confirmed that both Method 1 and Method 2 infer a common structure of two given strings by constructing a Pairwise-CFG. For example, we illustrate a parse tree generated with a grammar obtained by Method 2 in Fig. 4(a). We can see that they are identical except the leaves that are obtained by the separate rules, and so the trees obtained by deleting such leaves and edges for them are shared as a common structure of those strings.

We compare our methods with SEQUITUR. It is not guaranteed that SEQUITUR infers a common structure of inputs. A part of parse trees obtained by applying two strings whose lengths are 242 and 260 respectively to SEQUITUR is shown in Fig. 4(b). Here, G, C, A and T are terminals and $X_0, X_1, \cdots$ are non-terminal symbols. In Fig. 4(b), the structures of the two parse trees are different. Accordingly, SEQUITUR did not infer any common structures of the given two strings. We consider that this is because SEQUITUR does not consider any common features of them, and we can infer a common structure of them by constructing a Pairwise-CFG because it is based on their common subsequence.

In addition, the comparisons of compression ratio and calculation time among Method 1, Method 2 and SEQUITUR are shown in Fig. 5 and Fig. 6, respectively. We apply the concatenation of two given

[1] http://www.nbrp.jp

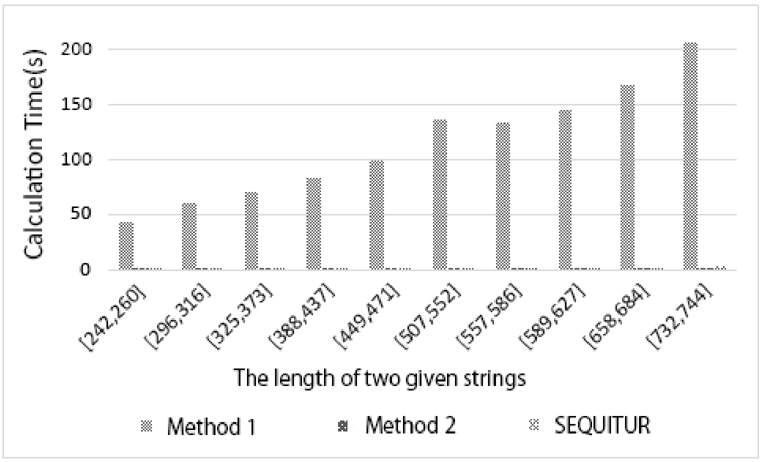


Fig. 6. Calculation Time

strings to SEQUITUR. The value of the horizontal axis in both figures is the pair of lengths of two given strings. The compression ratio is calculated as the result of dividing the total length of the given two strings by the total number of characters on the right-hand sides of all rules in obtained Pairwise-CFGs.

We can confirm that compression ratio of Method 1 is almost the same as that of Method 2 and that both of them are worse than SEQUITUR by observing Fig. 5. Additionally, for all methods, we can confirm that the longer the given two strings are, the longer its calculation time takes, from Fig. 6. Method 2 and SEQUITUR is faster than Method 1.

Method 2 compresses the original strings almost as highly as Method 1 with far shorter computation time of Method 1. We consider that this would be because Method 2 selects a candidate which has the most number of terminals of searched candidates, while in Method 1, for a candidate, the larger the number of terminals is in it, the higher its score.

5 Conclusion

In this paper, we propose a method that infers a common structure of two given strings. Our algorithm searches for common subsequences by applying Needleman-Wunsch algorithm [3] and constructs a grammar by using one of the found common subsequences. Constructed grammars by our algorithm represent a common structure of given two strings.

As our future work, in the case of focusing on compression rate, we should reduce computational complexity by pruning candidates whose scores are low in advance. Another subject in the future work is to develop an algorithm for more than two strings by applying a multiple sequence alignment algorithm [5].

References

1. Chiang, D.: Hierarchical Phrase-Based Translation. *Computational Linguistics*, Vol. 33, No. 2 (2007) 201–228
2. Larsson, N. J. and Moffat, A.: Offline Dictionary-Based Compression. *In Proceedings of the Data Compression Conference 1999* (DCC '99), IEEE Computer Society (1999) 296–305
3. Needleman, S. B. and Wunsch, C. D.: A General Method Application to the Search for Similarities in the Amino Acid Sequences of Two Proteins. *Journal of Molecular Biology*, Vol. 48 (1970) 443–453
4. Nevil-Manning, C. G. and Witten, I. H.: Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, Vol. 7 (1997) 67–82
5. Wang, L. and Jiang, T.: On the Complexity of Multiple Sequence Alignment. *Journal of Computational Biology*, Vol. 1, No. 4 (1994) 337–348
6. Yang, E., Kieffer, J. C.: Efficient Universal Lossless Data Compression Algorithm on a Greedy Sequential Grammar Transform – Part One: Without Context Models. *IEEE Transactions on Information Theory*, Vol. 46, No. 3 (2000) 755–777