Fragmentary Pattern Matching: Complexity, Algorithms and Applications for Analyzing Classic Literary Works

Hideaki Hori[1], Shinichi Shimozono[2⋆], Masayuki Takeda[3,4], and Ayumi Shinohara[3]

[1] Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology, Iizuka 820-8502, Japan
hori@daisy.ai.kyutech.ac.jp
[2] Department of Artificial Intelligence, Kyushu Institute of Technology
Iizuka 820-8502, Japan
sin@ai.kyutech.ac.jp
[3] Department of Informatics
Kyushu University 33, Fukuoka 812-8581, Japan
{takeda,ayumi}@i.kyushu-u.ac.jp
[4] PRESTO, Japan Science and Technology Corporation, Japan

Abstract. A fragmentary pattern is a multiset of non-empty strings, and it matches a string w if all the strings in it occur within w without any overlaps. We study some fundamental issues on computational complexity related to the matching of fragmentary patterns. We show that the fragmentary pattern matching problem is NP-complete, and the problem to find a fragmentary pattern common to two strings that maximizes the pattern score is NP-hard. Moreover, we propose a polynomial-time approximation algorithm for the fragmentary pattern matching, and show that it achieves a constant worst-case approximation ratio if either the strings in a pattern have the same length, or the importance weights of strings in a pattern are proportional to their lengths.
Keywords: fragmentary pattern, string resemblance, string matching, NP-completeness, polynomial-time approximation

1 Introduction

Waka is a form of traditional Japanese poetry with 1300-year history. A Waka poem has five lines and thirty-one syllables, arranged thus: 5-7-5-7-7. Since one syllable is represented by one Kana character in Japanese, a Waka poem consists of thirty-one Kana characters. In [13], we attempted to discover similar poems semi-automatically from an accumulation of about 450,000 Waka poems in a machine-readable form. One of the aims is to find unheeded instances of *Honkadori*, a technique based on specific allusion to earlier famous poems. The

⋆ This research is partially supported by Grants-in-Aid for Encouragement of Young Scientists, Japan Society for the Promotion of Science, No. 12780286.

P. Eades and T. Takaoka (Eds.): ISAAC 2001, LNCS 2223, pp. 719–730, 2001.

approach we took is very simple: Arrange all possible pairs of poems in decreasing order of their similarity, and scholarly scrutinize a first part.

The key to success in this approach would be how to develop an appropriate similarity measure. Traditionally, the scheme of weighted edit distance with a weight matrix may have been used to quantify affinities between strings (see e.g. [10]). This scheme, however, requires a fine tuning of quadratically many weights in a matrix with the size of alphabet, by a hand-coding or a heuristic criterion. As an alternative idea, we introduced a new framework called *string resemblance systems* (SRSs for short) [13]. In this framework, similarity of two strings is evaluated via a pattern that matches both of them, with the support by an appropriate function that associates the quantity of resemblance to candidate patterns. This scheme bridges a gap among optimal pattern discovery (e.g. [12]), machine learning (e.g. [2,3]) and similarity computation (e.g. [6,10]).

An SRS is specified by (1) a *pattern set* to which common patterns belong, and (2) a *pattern score function* that maps each pattern in the set to the quantity of resemblance. For example, if we choose the set of patterns with *variable-length don't-cares* (*VLDC's*) and define the pattern score to be the number of non-variable symbols in a pattern, then we obtain one of the traditional measures, the longest common subsequence (LCS): a common pattern `a⋆d⋆a⋆` for both `acdeba` and `abdac`, whose score is three. With this framework researchers can easily design and modify their measures not only for generic purposes but also for definite usages. In fact, we designed several similarity measures as combinations of a pattern set and a score function along with this framework, and reported successful results in discovering instances of Honkadori [13].

Some of the similarity measures employed in [13] base upon a class of *fragmentary patterns*, or *order-free patterns*. A fragmentary pattern is formally a multiset of non-empty strings. It matches a string w if all the strings in it occur within w without any overlaps. Although the computational complexity of matching a fragmentary pattern had not been clarified, the potential intractability to deal with it could be ignored for comparing Waka poems, since the lengths of the poems are only approximately 31.

However, the computational complexity is crucial and must be paid attention to when comparing longer texts by a fragmentary pattern. For example, searching for a fragmentary pattern in long texts arises in detecting instances of *Hikiuta*. Hikiuta is a rhetorical device used in *Monogatari* (tales), which is based on a specific allusion to a famous poem and appears in the narrative, conversation, and letters. A prose passage of the tale and the poem, therefore, share a phrase or part of phrase when this device is used. Other possible applications in molecular biology require that methods can process efficiently for huge size of sequences.

The purpose of this paper is to settle some fundamental issues on computational complexity related to the matching of fragmentary patterns and the string resemblance system adopting them. Firstly, we show that a matching decision of a fragmentary pattern is NP-complete. This indicates that if a pattern contains strings whose suffices and prefixes can overlap, then finding a set of non-overlapping occurrences of the strings becomes intractable. Also, we prove that

the problem to find a fragmentary pattern that is common to two strings and maximizes the pattern score is NP-hard. Furthermore we present a polynomial-time approximation algorithm for the maximization version of the fragmentary pattern matching, and show that the algorithm achieves a constant worst-case approximation ratio if (i) the strings in a pattern have the same length, or (ii) the importance weights of strings in a pattern are the lengths of them.

The rest of this paper is organized as follows. Section 2 gives a brief sketch of the framework of string resemblance systems. Section 3 defines the class of fragmentary patterns and then proves that the pattern matching problem for this class is NP-complete. Section 4 discusses the complexity required for computing similarity between two strings for SRSs with the fragmentary patterns. Section 5 considers combinatorial optimization versions of the fragmentary pattern matching and gives an approximation algorithm. Section 6 describes applications to two typical problems arisen in analysis of classic Japanese literary works.

2 A Unifying Framework for String Similarity

This section briefly sketches the framework of string resemblance systems according to [13]. Gusfield [10] pointed out that in dealing with string similarity the language of alignments is often more convenient than the language of edit operations. Our framework is a generalization of the alignment based scheme and is based on the notion of *common patterns*.

Before describing our scheme, we introduce some notations. The set of all strings over a finite alphabet Σ is denoted by Σ^*. The length of a string $s \in \Sigma^*$ is denoted by $|s|$. The *empty string* ε is the string of length zero. The set $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ thus denotes the set of all non-empty strings.

A *pattern system* is a triple $\langle \Sigma, \Pi, L \rangle$ of a finite alphabet Σ, a set Π of descriptions called *patterns*, and a function L that maps a pattern $\pi \in \Pi$ to a *language* $L(\pi) \subseteq \Sigma^*$. A pattern $\pi \in \Pi$ *matches* $w \in \Sigma^*$ if w belongs to $L(\pi)$. Also, π is a *common pattern of* w *and* u for strings $w, u \in \Sigma^*$, if π matches both of them. Usually, a set Π of patterns is expressed as a set of strings over an alphabet $\Sigma \cup X$, where X is a finite alphabet which is disjoint to Σ.

Definition 1. *A* string resemblance system *(SRS) is a quadruple* $\langle \Sigma, \Pi, L, Score \rangle$*, where* $\langle \Sigma, \Pi, L \rangle$ *is a pattern system and* $Score$ *is a* pattern score function *that maps a pattern in* Π *to a real number.*

The similarity $\mathrm{SIM}(x, y)$ between strings x and y *with respect to an SRS* $\langle \Sigma, \Pi, L, Score \rangle$ *is defined by*

$$\mathrm{SIM}(x, y) = \max\{Score(\pi) \mid \pi \in \Pi \text{ and } x, y \in L(\pi)\ \}.$$

When the set $\{Score(\pi) \mid \pi \in \Pi \text{ and } x, y \in L(\pi)\ \}$ *is empty or the maximum does not exist,* $\mathrm{SIM}(x, y)$ *is undefined.*

The definition given above regards the similarity computation as *optimal pattern discovery*. In this sense, our framework bridges a gap between similarity

computation and pattern discovery. In [13], the class of homomorphic SRSs was defined, and it was shown that the class covers most of the well-known and well-studied similarity (dissimilarity) measures, including the edit distance, the weighted edit distance, the Hamming distance, the LCS measure. Also this class was extended to the semi-homomorphic SRSs in [13], into which for example the similarity measures for musical sequence comparison developed in [11] falls.

Interestingly, membership problems of homomorphic and semi-homomorphic pattern systems are assumed reasonably to be polynomial-time solvable, while membership problems of non-homomorphic pattern systems include NP-complete one, e.g. the Angluin pattern system [1]. The similarity computation for homomorphic and semi-homomorphic SRSs can be performed in polynomial time [13] by the idea of weighted edit graph (see, e.g., [10]) under the above assumption, while the similarity computation via the Angluin pattern system is NP-hard in general [14]. We emphasize that the fragmentary pattern system is included in the class of non-homomorphic pattern systems.

3 Fragmentary Patterns and Complexity of Their Matching

We focus on the class of fragmentary patterns in this section, and discuss the computational complexity of a matching or a searching of an arbitrary large fragmentary pattern, before looking into SRSs adopting this class.

A *fragmentary pattern over* Σ is a multiset $\{p_1, \ldots, p_\ell\}$ of $\ell > 0$ non-empty strings $p_1, \ldots, p_\ell \in \Sigma^+$, and is denoted by $\pi[p_1, \ldots, p_\ell]$. The size of a fragmentary pattern $\pi[p_1, \ldots, p_\ell]$ is the total length of strings $p_1, \ldots, p_\ell$, and denoted by $\|\pi\|$.

Definition 2 (Fragmentary pattern system). *The fragmentary pattern system on* Σ *is a pattern system* $\langle \Sigma, \Pi, L \rangle$ *such that* (i) Π *is the set of all fragmentary patterns over* Σ*, and* (ii) L *is the function that maps* $\pi[p_1, \ldots, p_\ell] \in \Pi$ *to the language* $L(\pi[p_1, \ldots, p_\ell])$ *that contains all strings expressed by*

$$s_0 \cdot p_{\sigma(1)} \cdot s_1 \cdot p_{\sigma(2)} \cdot s_2 \cdots s_{\ell-1} \cdot p_{\sigma(\ell)} \cdot s_\ell,$$

where $s_0, s_1, \ldots, s_\ell$ *are arbitrary strings in* Σ^* *and* $\langle \sigma(1), \ldots \sigma(\ell) \rangle$ *is an arbitrary permutation of integers* $1, \ldots, \ell$.

For example, the language of the pattern $\pi[abc, de]$ is denoted by a regular expression

$$L(\pi[abc, de]) = \Sigma^* abc \Sigma^* de \Sigma^* \cup \Sigma^* de \Sigma^* abc \Sigma^*.$$

In the context of a string pattern matching, the following notions are convenient. Let p and t be strings over Σ^*. An *occurrence position* i *of* p *in* t is an integer such that $p = t[i] \cdots t[i + |p| - 1]$. The range $[i, i + |p| - 1]$ on t represents the substring $t[i] \cdots t[i + |p| - 1]$ and is said to be an *occurrence of* p *in* t. A fragmentary pattern $\pi[p_1, \ldots, p_\ell]$ *matches* $t \in \Sigma^*$ if there is a sequence $\langle k_1, \ldots, k_\ell \rangle$ of integers such that (i) every k_i for $1 \leq i \leq \ell$ is an occurrence position of p_i

in t, and (ii) $k_i + |p_i| - 1 < k_j$ holds for any $k_i < k_j$, i.e. any pair of occurrences never overlap. We say such a sequence $\langle k_1, \ldots, k_\ell \rangle$ *an occurrence of* π *in* t.

Then the following is a fundamental problem for a fragmentary pattern system $\langle \Sigma, \Pi, L \rangle$ on Σ.

Definition 3. Fragmentary Pattern Matching (Frag-Matching)
Given a fragmentary pattern $\pi \in \Pi$ *and a string* $w \in \Sigma^*$*, determine whether* w *belongs to* $L(\pi)$*.*

This may rather seem to be tractable. Actually, if no pair of strings in a fragmentary pattern shares a common string as a prefix and a suffix, then strings in a pattern cannot overlap and thus this problem is solvable in polynomial time. It is a simple 'AND' query of multiple string patterns. However, in general, the following theorem holds.

Theorem 1. Fragmentary Pattern Matching *is NP-complete.*

Firstly, we prove this theorem by a reduction from 3Sat to Frag-Matching, with which a reduced instance requires an alphabet whose size depends on the size of a given 3CNF formula. After showing it, we briefly discuss how those symbols can be expressed over an alphabet of fixed size. The problem 3Sat (e.g. [8]) is, given a set $C = \{c_1, \ldots, c_m\}$ of 3 literal clauses over a set $X = \{x_1, \ldots, x_n\}$ of Boolean variables, to determine whether C is satisfiable.

Proof. In the following we show a logspace algorithm that builds an instance (t_C, P_C) of Fragmentary Pattern Matching over an alphabet

$$\Sigma_C = \{x_1, \ldots, x_n, c_1, \ldots, c_m, \#\}$$

for a 3Sat instance (X, C).

We introduce some gadgets utilized to construct t_C and P_C. For each $1 \leq i \leq n$, we define $t_1^i = x_i x_i \, c_1 x_i \, c_2 x_i \cdots c_m x_i \, x_i \#$, and $t_2^i = t_2^i(1) \cdot \cdots \cdot t_2^i(m)$ where

$$t_2^i(j) = \begin{cases} c_j c_j x_i c_j \# \text{ if } c_j \text{ contains } x_i, \\ c_j x_i c_j c_j \# \text{ if } c_j \text{ contains } \neg x_i, \\ c_j x_i c_j \# \quad \text{if neither } x_i \text{ nor } \neg x_i \text{ is in } c_j, \end{cases}$$

for $1 \leq j \leq m$. With these gadgets, we define $t_1 = t_1^1 \cdots t_1^n$ and $t_2 = t_2^1 \cdots t_2^n$, and as the concatenation $t_C = t_1 \cdot t_2$. The pattern P_C is defined by the union of $P_1 = \cup_{i=1}^n \{x_i x_i\}$, $P_2 = \cup_{j=1}^m \{c_j c_j\}$ and $P_3 = \cup_{i=1}^n \{c_1 x_i, x_i c_1, \ldots, c_m x_i, x_i c_m\}$. Note that P_C contains only strings of the length two. Clearly, this algorithm runs with logarithmic space.

The gadgets defined above have the following properties: (i) P_1 matches t_1, while any string in it does not match t_2; (ii) for each $1 \leq i \leq n$, the string $x_i x_i \in P_1$ is either the prefix of t_1^i, or the suffix of t_1^i; and (iii) P_2 matches t_2, while any string in it does not match t_1. Also, for each $1 \leq i \leq n$ and $1 \leq j \leq m$, either $c_j x_i$ or $x_i c_j$ in P_3 matches t_1^i, and the remaining one matches t_2^i.

Now, we prove that (X, C) is satisfiable if and only if P_C matches t_C. Firstly, we show that if there is a truth assignment $f : X \to \{\texttt{true}, \texttt{false}\}$ that satisfies (C, X), then an occurrence of P_C in t_C exists.

According to the assignment f, we split $P_1 \cup P_3$ into two sets: we define, for each $1 \leq i \leq n$,

$$Q_1^i = \{x_i x_i, c_1 x_i, \ldots, c_m x_i\}, \quad Q_2^i = \{x_i c_1, \ldots, x_i c_m\}$$

if $f(x_i)$ is `true`, and otherwise (if $f(x_i) = \texttt{false}$) define

$$Q_1^i = \{x_i x_i, x_i c_1, \ldots, x_i c_m\}, \quad Q_2^i = \{c_1 x_i, \ldots, c_m x_i\}.$$

Note that Q_1^i and Q_2^i matches t_1^i and t_2^i, respectively, without depending on whether $f(x_i)$ is `true` or `false`. Then, since f satisfies C, for each $1 \leq j \leq m$, there must be an index $1 \leq i \leq n$ such that either x_i or $\neg x_i$ satisfies c_j.

This can be interpreted with the above definition that for each $1 \leq j \leq m$ there is a variable index $1 \leq i \leq n$ such that either (a) $c_j c_j x_i c_j \#$ occurs in t_2^i and $x_i c_j$ is in Q_2^i, or (b) $c_j x_i c_j c_j \#$ occurs in t_2^i and $c_j x_i$ is in Q_2^i. Then, in t_2^i there remains a substring $c_j c_j$ to which a string $c_j c_j$ in P_2 of the pattern matches. This guarantees that P_2 can, with all Q_2^i's, match t_2, and thus the whole fragmentary pattern P_C matches t_C.

Next we show that if P_C matches t_C then a truth assignment associated with an occurrence of P_C satisfies C.

By the construction of (t_C, P_C), for each $1 \leq i \leq n$, either the pattern

$$\{x_i x_i, c_1 x_i, \ldots, c_m x_i\}, \text{ or } \{x_i x_i, x_i c_1, \ldots, x_i c_m\}$$

must match t_1^i; otherwise we lose all the possible places where $x_i x_i$ in P_1 occurs. With respect to this choice, we define the set $P_T^i \subseteq P_3$ as either $\{x_i c_1, \ldots, x_i c_m\}$ or $\{c_1 x_i, \ldots, c_m x_i\}$ for each $1 \leq i \leq n$. Also, we define $P_T = \bigcup_{i=1}^n P_T^i$ and $P_F = P_3 - P_T$. Then, $P_1 \cup P_T$ matches t_1, and this requires that $P_2 \cup P_F$ matches t_2. For each $1 \leq i \leq m$, there is an index $1 \leq i \leq n$ such that either (a) t_2 contains $c_j c_j x_i c_j \#$ and $x_i c_j$ is in P_F, or (b) t_2 contains $c_j x_i c_j c_j \#$ and $c_j x_i$ is in P_F. Otherwise we have no positions to which $c_j c_j$ matches without overlaps.

According to the occurrence of P_C in t_C inspected as above, we define a truth assignment f as follows: $f(x_i) = \texttt{true}$ if P_T^i includes $c_j x_i$ $(1 \leq j \leq m)$; $f(x_i) = \texttt{false}$ if P_T^i includes $x_i c_j$ $(1 \leq j \leq m)$. Then, since P_F and P_2 must match t_2, like the discussion on Q_2^i's and P_2 in above, the assignment f implies that for each clause in C there is at least one literal having `true`. Therefore, C is satisfiable if P_C matches t_C.

The above two properties complete this proof. □

The reduction presented here can be easily modified to one that reduces to an instance of Frag-Matching over an alphabet consisting of a fixed number of symbols. For example, an alphabet $\Sigma = \{0, 1, \$\}$ could be used to represent finitely many symbols in Σ_C by distinguished binary strings of the same length, followed with the separator symbol '$.' The coding sizes of P_C and t_C is expanded only $\log |\Sigma_C|$ times the original represented with Σ_C. Even the unary coding scheme can be applied.

Corollary 1. Fragmentary Pattern Matching *is NP-complete even if either* (i) *the size of the alphabet is fixed, or* (ii) *strings in a pattern are of the same length, or both.*

4 Complexity of Similarity Computation by Fragmentary Patterns

We now consider the computation of similarity between two strings and its computational complexity. In the following, we assume the values of score function are integers.

Definition 4. Similarity Computation with SRS $\langle \Sigma, \Pi, L, Score \rangle$.
Given two strings $w_1, w_2 \in \Sigma^*$, find a pattern $\pi \in \Pi$ with $\{w_1, w_2\} \subseteq L(\pi)$ that maximizes $Score(\pi)$.

Let $\#$ be a symbol not in Σ, and π a fragmentary pattern $\pi[u_1, \ldots, u_\ell]$ over Σ. For a fragmentary pattern π' over Σ, we write $\pi' \preceq \pi$ if π' matches the string $u_1\# \ldots u_\ell\#$ in $(\Sigma \cup \{\#\})^*$. Here, the function L is naturally extended to one that maps a pattern to the language $L(\pi_1)$ over $\Sigma \cup \{\#\}$. We write as $\pi_1 \prec \pi_2$ if $\pi_1 \preceq \pi_2$ and the two multisets π_1 and π_2 are not identical. A pattern score function $Score$ is *strictly increasing with respect to* $\prec$ if $\pi_1 \prec \pi_2$ implies $Score(\pi_1) < Score(\pi_2)$. For example, let $Score_1(\pi) = \|\pi\|$ and $Score_2(\pi[u_1, \ldots, u_\ell]) = \sum_{i=1}^{\ell} |u_i|^2$. Then, $Score_2$ is strictly increasing, while $Score_1$ is not.

Theorem 2. Similarity Computation with SRS *with the fragmentary pattern system is NP-hard in general.*

Proof. We show the NP-completeness of a decision version of Similarity Computation with the class of pattern score functions that are strictly increasing: Given two strings $w_1, w_2 \in \Sigma^*$ and a nonnegative integer k, determine whether a pattern $\pi \in \Pi$ satisfying $\{w_1, w_2\} \subseteq L(\pi)$ and $Score(\pi) \geq k$ exists.

We give a reduction from Fragmentary Pattern Matching $\langle \Sigma, \Pi, L \rangle$ to Similarity Computation with SRS $\langle \Sigma', \Pi', L', Score \rangle$. A triple $\langle \Sigma', \Pi', L' \rangle$ is the fragmentary pattern system on $\Sigma' = \Sigma \cup \{\#\}$, and $Score$ is a pattern score function defined on the set of fragmentary patterns Π' over Σ', whose limitation to $\Pi \subseteq \Pi'$ is strictly increasing with respect to $\prec$.

For a given instance $\pi = \pi[u_1, \ldots, u_\ell] \in \Pi$ and $w \in \Sigma^*$ of Fragmentary Pattern Matching, we construct an instance (w'_1, w'_2, k) of Similarity Computation by letting $w'_1 = u_1\# \ldots u_\ell\#$, $w'_2 = w$, and $k = Score(\pi)$. Since $\#$ does not occur in w'_2, there is a pattern $\pi' \in \Pi'$ with $\{w'_1, w'_2\} \subseteq L'(\pi')$ and $Score(\pi') \geq k$ if and only if $w \in L(\pi)$. This completes the proof. □

On the other hand, there are pattern score functions that are not trivial and with which similarity can be efficiently computed. For example, with the pattern score function that can be considered as an *order-free version* of LCS, we can readily show that:

Theorem 3. SIMILARITY COMPUTATION WITH RESPECT TO SRS *with the fragmentary pattern system is solvable in linear time using* $O(|\Sigma|)$ *space for the pattern score function* $Score(\pi) = \|\pi\|$.

5 Maximization of Fragmentary Pattern Matching

More than a powerful pattern class for the similarity computation, fragmentary patterns can be used as a conjunction of queries for texts in which word-boundaries are not evident. By viewing the matching problem as a combinatorial optimization problem, a fragmentary pattern can be thus applied like an at-least-k-of-m rule. It is regarded as a generalization of the membership problem of fragmentary patterns, to classify noisy inputs with a specified robustness.

So now we consider the problem to find a maximal subset of a given set of strings that matches a text as a fragmentary pattern. Firstly, we introduce some notions of combinatorial optimization problems. In the following we only deal with and thus define 'maximization versions' of combinatorial optimization problems. (See e.g. [4,5] for details.)

A *maximization problem* P is specified by (i) the set I_P of instances, (ii) the set $S_P(x)$ of *solutions* of each instance $x \in I$, and (iii) the *measure* $m_P(x, s)$ that maps a pair of an instance x and a solution s of x to a nonnegative integer. The ultimate goal of a maximization problem is to find an *optimum solution*, that is, a solution whose measure is maximum. An *approximation algorithm* A *for* P is an algorithm that produces for any instance $x \in I_P$ a solution $s \in S_P(x)$. Furthermore, for a rational number $r > 1$, A is said to be an *r-approximation algorithm for P* if A always produces a solution whose measure is no less than $1/r$ times the measure of an optimum solution. A maximization problem P is in class APX if there is a polynomial-time r-approximation algorithm for P with some constant r.

A maximization version of our pattern matching problem is formalized as follows.

Definition 5. MAXIMUM FRAGMENTARY PATTERN MATCHING (MAX FRAG-MATCHING)
Given a weighted instance of FRAG-MATCHING*, i.e. a triple* (π, w, t) *of a fragmentary pattern* $\pi \in \Pi$*, a weight* $w : \pi \to \mathrm{Z}^+$ *and a string* $t \in \Sigma^*$*, find a fragmentary pattern* $\pi' \subseteq \pi$ *that matches* t *and maximizes the total weight* $\sum_{u \in \pi'} w(u)$ *in* π'.

For this maximization problem, let us consider the following simple polynomial-time algorithm.

Algorithm *Greedy*
Input: An instance triple (π, w, t);
Output: A fragmentary pattern $\pi' \subseteq \pi$ that matches t.

1. Let $\pi' = \emptyset$, and let I be an empty list of occurrences.
2. For each $u \in \pi$, in the weight-descending order with respect to w, do the following:

   a. Find an occurrence of u in t, say $[k, \ell]$, which does not overlap any occurrences in I; If no such an occurrence can be found, then continue to the next iteration to proceed to the next string in π.
   b. Add u to π', and add the occurrence $[k, \ell]$ to I.
3. Output π'.

This algorithm runs in $O(n \log n + m)$ time with the number n of strings in π and the length m of string t, by employing appropriate sorting, set managing and string matching algorithms. Furthermore, with certain kinds of restrictions on input strings or weight functions, the following lemmas hold:

Lemma 1. *If all the strings in π have the same length, then the algorithm Greedy is a 3-approximation algorithm, i.e. guarantees an output whose total weight is at least $1/3$ times the total weight of an optimum solution.*

Proof. Let $\pi^* \subseteq \pi$ be an optimum fragmentary pattern for t. An addition of string u to π' with some occurrence, in an iteration at the step 2-b, can interfere at most two strings in π^* matching t. For these two strings, there are following three cases: (i) each of the two strings has the weight less than $w(u)$, (ii) the two strings are already chosen in π', or (iii) the two strings are interfered by some string already chosen in π'. Therefore the addition of u disables the contributions of weights from π^* no more than $2w(u)$, while in π^* the two strings and u may contribute totally at most $3w(u)$. By repeating this process, we finally obtain a solution whose total weight is at least $\frac{1}{3}$ times the optimum. □

Lemma 2. *If the weight of each string is the length of it, then the algorithm Greedy is a 4-approximation algorithm.*

Proof. This can be shown by a discussion similar to the previous proof. An addition of u to π' at each iteration of the Step 2-b may block some strings in π^* occurring in the text. Since $|u| = w(u)$ contiguous symbols are occupied by the occurrence of u, the total weight of those blocked strings is at most $w(u) - 2 + 2w(u) < 3w(u)$. The string u may also be included in π^*, so the algorithm is guaranteed to choose a fragmentary pattern whose total weight is no less than $\frac{1}{4} = \frac{w(u)}{w(u)+3w(u)}$ times the optimum. □

Note that the restricted subproblem considered in lemma 1 includes instances constructed in the reduction presented in Section 3. Also the case dealt with lemma 2 seems likely to occur in practical applications, since shorter strings may have less meaning in general, and in automated pattern discovery some automatic weighting scheme will be requested.

Corollary 2. Max Fragmentary Pattern Matching *is in the class APX [5] if strings in a fragmentary pattern have the same length. Also the problem is in APX if the weight function is equal to or stronger than the length of string.*

6 Applications for Classic Literary Works

Honkadori is a technique of composing a Waka poem as an allusive-variation of a model poem. In [13], we developed a similarity measure appropriate for finding instances of Honkadori, based on a measure to quantify affinities between two lines which falls into the class of semi-homomorphic SRSs mentioned in Section 2. With this measure we have succeeded to discover instances of Honkadori which have never been pointed out in the long research history of Waka poetry. In [13] we also showed two similarity measures, which are defined as SRSs with fragmentary pattern systems. The difference of the two measures lies in the pattern score functions. Each of the pattern score functions can be described as

$$Score(\pi[u_1, \ldots, u_\ell]) = \sum_{i=1}^{\ell} f(u_i) \tag{1}$$

with a function f that maps a string in Σ to a real number. One measure is obtained by letting

$$f(u) = \begin{cases} |u|, \text{ if } |u| > \ell; \\ 0, \quad \text{otherwise}, \end{cases} \tag{2}$$

where ℓ is a threshold in ignoring short fragments in a common pattern. In [13], we set $\ell = 1$. This measure is suitable for discovering instances of Honkadori with word-order alternations, as shown in Fig. 1.

Poem alluded to. (Kokin-Shū #125)
KA-HA-TSU-NA-KU/I-TE-NO-YA-MA-FU-KI/CHI-RI-NI-KE-RI
HA-NA-NO-SA-KA-RI-NI/A-HA-MA-SHI-MO-NO-WO

Allusive-variation. (Shin-Kokin-Shū #1162)
A-SHI-HI-KI-NO/YA-MA-FU-KI-NO-HA-NA/CHI-RI-NI-KE-RI
I-TE-NO-KA-HA-TSU-HA/I-MA-YA-NA-KU-RA-MU

Fig. 1. An instance of Honkadori with word-order alternations

Although Similarity Computation for this score function is NP-hard, the length of Waka poems we dealt with was approximately 31. Thus we could have performed the computation in feasible time.

The other measure is obtained by letting $f(u)$ be the *rarity* of string u, that is, $f(u)$ is the logarithm of inverse of the probability of occurring u in database. The idea of rarity was shown to be effective in identifying only close affinities which are hardly seen elsewhere, possibly excluding known stereotype expressions [13].

Hikiuta is a poetic device used in tales, which is based on a specific allusion to a famous poem. We wish to find a portion of a tale which alludes to a poem. We use an SRS with fragmentary pattern system to quantify the affinities between

a substring of a tale and a poem. For this purpose, the length of a substring to be compared to a poem has to be limited by an appropriate threshold called *window size*, as in the *episode matching* (e.g. [9]). Our problem is then formalized as follows:

> Given a short string, called *poem*, a long string, called *tale*, a window size $k > 0$, and a threshold t, to find all substrings of the tale that are of length k and resemble the poem with a similarity value higher than t.

Preliminary experimental results suggest that the pattern score function defined by Eq. 1 and Eq. 2 with a relatively large value of ℓ might be suitable for effectively detecting instances of Hikiuta within a tale. A practically efficient approach would be a filtering technique based on searching of fragments of the poem that are of length greater than the threshold ℓ within the tale, in which such index structures as the directed acyclic word graphs (e.g. [7]) will play a key role, and verification of candidate areas of the tale.

Acknowledgments

The authors would be grateful to the anonymous referees for their careful reading of the draft and useful comments.

References

1. D. Angluin. Finding patterns common to a set of strings. *J. Comput. Sys. Sci.*, 21:46–62, 1980. 722
2. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. 720
3. D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988. 720
4. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag, Berlin, 1999. 726
5. G. Ausiello, P. Crescenzi, and M. Protasi. Approximate solution of NP optimization problems. *Theor. Comput. Sci.*, 150: 1–55, 1995. 726, 727
6. A. Z. Broder. On the resemblance and containment of documents. In *Proc. Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29, 1997. 720
7. M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, New York, 1994. 729
8. M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman & Co., New York, 1979. 723
9. G. Das, R. Fleischer, L. Gasieniec, D. Gunopulos, and J. Karkkainen. *Episode Matching.* In *Proc. 8th Annual Symposium on Combinatorial Pattern Matching (CPM'97)*, pages 12–27, 1997. 729
10. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, 1997. 720, 721, 722

11. T. Kadota, M. Hirao, A. Ishino, M. Takeda, A. Shinohara, and F. Matsuo. Musical sequence comparison for melodic and rhythmic similarities. In *Proc. 8th International Symposium on String Processing and Information Retrieval (SPIRE2001)*, 2001, to appear. 722
12. S. Shimozono, H. Arimura, and S. Arikawa. Efficient discovery of optimal word-association patterns in large databases. *New Gener. Comput.*, 18(1):49–60, 2000. 720
13. M. Takeda, T. Fukuda, I. Nanri, M. Yamasaki, and K. Tamari. Discovering instances of poetic allusion from anthologies of classical Japanese poems. *Theor. Comput. Sci.*, 2001, to appear. 719, 720, 721, 722, 728
14. K. Yamamoto, M. Takeda, A. Shinohara, T. Fukuda, and I. Nanri. Discovering repetitive expressions and affinities from anthologies of classical Japanese poems. In *Proc. 4th International Conference on Discovery Science (DS2001)*, 2001, to appear. 722