Pseudoknot-Generating Operation

Da-Jung Cho[1], Yo-Sub Han[1], Timothy Ng[2], and Kai Salomaa[2]

[1] Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120–749, Republic of Korea
{dajung,emmous}@cs.yonsei.ac.kr
[2] School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
{ng,ksalomaa}@cs.queensu.ca

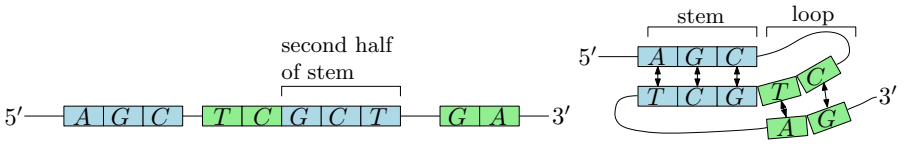
Abstract. A pseudoknot is an intra-molecular structure formed primarily in RNA strands and much research has been done to predict efficiently pseudoknot structures in RNA. We define an operation that generates all pseudoknots from a given sequence and consider algorithmic and language theoretic properties of the operation. We give an efficient algorithm to decide whether a given string is a pseudoknot of a regular language L—the runtime is linear if L is given by a deterministic finite automaton. We consider closure and decision properties of the pseudoknot-generating operation. For DNA encoding applications, pseudoknot structures are undesirable. We give polynomial-time algorithms to decide whether a regular language L contains a pseudoknot or a pseudoknot generated by some string of L. Furthermore, we show that the corresponding questions for context-free languages are undecidable.

Keywords: pseudoknots, pseudoknot-generating operation, closure and decision properties, formal languages

1 Introduction

A ribonucleic acid (RNA) often forms secondary structures according to the base-pairing with Adenine (A), Uracil (U), Guanine (G) and Cytosine (C) [5]. These bases A, G, C and U complementarily bind and form a double helix called *stem*, and double helix with unpaired loop known as *stem-loop*. A RNA structure generally has stems and various kinds of loops as a structural motif, which then gives rise to well-known structures such as hairpin or pseudoknot. RNA structures play an important role in cells and give insights to molecular evolution and function of RNA molecule [19]. Therefore, in bioinformatics, it is one of the most important and fundamental problems to predict RNA structures made up of a set of stems with optimal thermodynamic energy. Note that stabilized optimal foldings of a RNA sequence are closely related to the minimum free energy of RNA secondary structures based on the theory of thermodynamics.

A pseudoknot structure contains at least two stem-loops that occur in RNA with intramolecular base-pairing: Second half of one stem is embedded in between the two halves of another stem. (See Fig. 1 for an example of pseudoknot structure.) Pseudoknot structures appear in many natural RNA molecules



(a) a sequence for pseudoknot structure (b) folding pseudoknot structure

Fig. 1: A pseudoknot structure example: (a) A sequence contains a pseudoknot structure in which the second half of stem (blue box) exists between the two halves of another stem (green boxes) (b) A sequence folds into a pseudoknot.

and are closely related with the ribosomal frameshifting that allows viruses to create many protein structures from a relatively small genome [9]. Since the ribosomal frameshifting affects on encoding protein and the pseudoknot structure gives a tertiary structure of molecule, it is a major topic of biomolecular computing to predict pseudoknot structures [3, 9]. This led researchers to study efficient methods that predict pseudoknot structures [2, 4]. From a formal language viewpoint, several researchers [8, 15, 17, 18] characterized the pseudoknot structure and suggested pseudoknot predicting algorithms. Given an input, the problem of predicting or aligning arbitrary pseudoknot structures is NP-hard [2, 11]. Möhl et al. [17] presented an algorithm that computes the edit-distance of two RNA structures with arbitrary pseudoknots and showed that the algorithm is applicable in practice. Kari and Seki [15] formalized particular case of pseudoknot structures under formal language theory and investigated its properties. Evans [8] proposed the first polynomial-time algorithm for finding maximum common substructures that include pseudoknots. Rinaudo et al. [18] generalized several RNA structures and presented an alignment algorithm based on tree decomposition approach.

While most researchers considered a problem of predicting pseudoknot structures from a (long) sequence, we consider pseudoknot structures from a different angle: A sequence may be expanded (namely, append a new sequence itself) to form a pseudoknot structure. We consider this process and define a new operation *pseudoknot-generating* operation that generates all pseudoknot structures (from now we just call *pseudoknots* in short) from a given sequence. Then the resulting sequences fold itself into pseudoknots. Thus the input string becomes a seed string to generate pseudoknots. We establish the closure properties of the pseudoknot-generating operation on a string and present an algorithm that determines whether or not a string is a pseudoknot. Since pseudoknot structures are related to some biological mutations, they are crucial for detecting mutational patterns of a DNA sequence. We also study the closure properties of pseudoknot-generating operation on languages and examine several questions related to pseudoknots with respect to languages. From a biological view point, we can think of the pseudoknot-generating operation on a language as a procedure to generate all possible pseudoknots that may cause a mutation from a set of subsequences. In particular, we theoretically demonstrate that one can

check whether or not two sets of DNA sequences contain common mutational seed sequences. Furthermore, we define the pseudoknot-freeness and investigate the decidability problem for pseudoknot-freeness for regular and context-free languages.

In Section 2, we recall some notation and define the pseudoknot-generating operation. We consider the pseudoknot-generating operation and design several algorithms for recognizing generated pseudoknots from strings and finite automata in Section 3. Then, we study closure and decision properties of the pseudoknot-generating operation, and, investigate the pseudoknot-free languages in Section 4.

2 Preliminaries

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ. The size $|\Sigma|$ of Σ is the number of characters in Σ. A language over Σ is a subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string. Given a string $x = x_1 \cdots x_n$, $|x|$ is the number of characters in x, $x(i)$ denotes the ith character x_i of x and $x(i,j) = x_i x_{i+1} \cdots x_j$ is the substring of x from position i to position j, where $i \leq j$. Given two strings x and y in Σ^*, x is a *prefix* of y if there exists $z \in \Sigma^*$ such that $xz = y$ and x is a *suffix* of y if there exists $z \in \Sigma^*$ such that $zx = y$. Furthermore, x is said to be a *substring* or an *infix* of y if there are two strings u and v such that $uxv = y$.

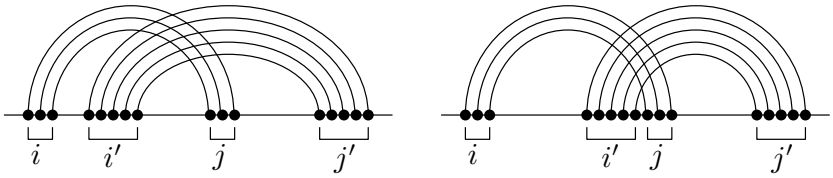
An FA A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. If F consists of a single state f, then we use f instead of $\{f\}$ for simplicity. Let $|Q|$ be the number of states in Q and $|\delta|$ be the number of transitions in δ. Then, the size of A is $|A| = |Q| + |\delta|$. For a transition $\delta(p, a) = q$ in A, we say that p has an *out-transition* and q has an *in-transition*. If $\delta(q, a)$ has a single element q', then we denote $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$ for simplicity.

A string x over Σ is accepted by A if there is a labeled path from s to a final state such that this path spells out x. We call this path an *accepting path*. Then, the language $L(A)$ of A is the set of all strings spelled out by accepting paths in A. We say that a state of A is *useful* if it appears in an accepting path in A; otherwise, it is *useless*. Unless otherwise mentioned, in the following we assume that all states of A are useful.

Given a string x, we say that x has a pseudoknot if there exists a substring w of x such that $w = w_1 w_2 w_3 w_4 w_1^R w_5 w_3^R$ for some strings $w_1, w_2, w_3, w_5 \in \Sigma^+$ and $w_4 \in \Sigma^*$. We call the string w *pseudoknot string* (or *pseudoknot* in short). We consider a restricted pseudoknot in which $w_4 = \lambda$, which means that half of one stem is adjacent to half of another stem. (See Fig. 2 for an example of restricted pseudoknot.)

Given a string x, we define the restricted pseudoknot-generating operation

$$\mathbb{PK}_{\mathbb{R}}(x) = \{x_1 x_2 x_3 x_1^R x_4 x_3^R \mid x = x_1 x_2 x_3 \text{ and } x_1, x_2, x_3, x_4 \in \Sigma^+\}.$$



(a) simple pseudoknot (b) restricted pseudoknot

Fig. 2: An example of two types of pseudoknot on a sequence of length n: (a) a simple pseudoknot with two base-pairings (i, j) and (i', j') (b) a restrict version of pseudoknot with two base-pairings (i, j) and (i', j') such that the first half of (i', j') is immediately followed by the second half of (i, j), where $0 \leq i < i' < j < j' \leq n$.

For a language L,

$$\mathbb{PK}_{\mathbb{R}}(L) = \bigcup_{x \in L} \mathbb{PK}_{\mathbb{R}}(w).$$

We define the iterated operation of $\mathbb{PK}_{\mathbb{R}}$ to be, for $i \geq 0$,

$$\mathbb{PK}_{\mathbb{R}}{}^{(0)}(L) = L, \quad \mathbb{PK}_{\mathbb{R}}{}^{(i+1)}(L) = \mathbb{PK}_{\mathbb{R}}(\mathbb{PK}_{\mathbb{R}}{}^{i}(L)), \quad \mathbb{PK}_{\mathbb{R}}{}^{*}(L) = \bigcup_{i=0}^{\infty} \mathbb{PK}_{\mathbb{R}}{}^{i}(L).$$

In the following, we only consider restricted pseudoknots and call them simply pseudoknots unless we need to distinguish restricted pseudoknots from general pseudoknots.

3 Algorithms for recognizing generated pseudoknots

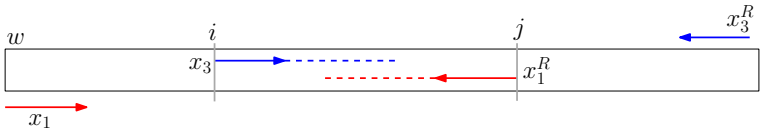


Fig. 3: A naive approach for checking whether or not w is a pseudoknot. For each substring $w(i, j)$, we check whether or not $w(i, j)$ is a catenation of x_3 and x_1^R for a prefix x_1 and a suffix x_3 of w—$w(i, j) = x_3 x_1^R$—by comparing characters from both directions.

We first study the problem for checking whether or not a string $w = w_1 w_2 \cdots w_n$ is a pseudoknot; namely, is $w = x_1 x_2 x_3 x_1^R x_4 x_3^R$ for some $x_1, x_2, x_3, x_4 \in \Sigma^+$. The main idea of our approach is to check if there exists a substring $x_3 x_1^R$ of w such that x_1 is a prefix and x_3^R is a suffix of w. A naive approach is to consider

all possible substrings and check this condition. We design a better algorithm that checks the required condition more efficiently based on the Aho-Corasick algorithm [1].

Before we describe the whole algorithm, we first present an algorithm that finds the shortest length of the matching prefix of the input pattern string with respect to the input for each index of the input. This algorithm is crucial for checking whether or not w is a pseudoknot.

Procedure ShortestMatchingLength(w, T)

```
/* w is a length m pattern and T is a length n text                    */
Construct a DFA A = (Q, Σ, δ, 0, Q \ {0}) for w, where Q = {0, 1, ..., m}

/* construct the goto function 𝔾                                       */
𝔾(0, a ≠ w_1 ∈ Σ) ← 0
for i ← 0 to m − 1 do
    𝔾(i, w_{i+1}) ← i + 1

/* construct the failure function 𝔽 and the output function 𝕆          */
𝔽(1) ← 0
for i ← 1 to m do
    if 𝔾(i, a) = i + 1 then
        v ← 𝔽(i)
        while 𝔾(v, a) ≠ ∅ do
            v ← 𝔽(v)
        𝔽(i + 1) ← 𝔾(v, a)
        𝕆(i + 1) ← min(𝕆(i + 1), 𝕆(𝔽(i + 1)))

/* read T using 𝔾, 𝔽, 𝕆                                               */
q ← 0
for i ← 1 to n do
    while 𝔾(q, T(i)) ≠ ∅ do
        q ← 𝔽(q)
    q = 𝔾(q, T(i))
    if 𝕆(q) ≠ ∅ then
        SML[q] ← 𝕆(q)
return SML
```

Given an input pattern string w and a text T, **Proc.** ShortestMatchingLength is a modified Aho-Corasick algorithm that finds the shortest length of the matching prefix of w at each index of T; if u the shortest matching prefix of w, then the reversal u^R of u appears as an infix of T. The two main differences from the original Aho-Corasick algorithm are

1. it receives only one string w as an input pattern and regards all prefixes of w as matching patterns
2. the output function $\mathbb{O}$ returns the shortest length of the matching pattern instead of reporting all matching patterns: $\mathbb{O}(i{+}i) \leftarrow \min(\mathbb{O}(i{+}1), \mathbb{O}(\mathbb{F}(i{+}i)))$.

It is easy to verify that **Proc.** ShortestMatchingLength runs in $O(m+n)$ time, where $m = |w|$ and $n = |T|$.

Now we design the whole algorithm that determines whether or not w is a pseudoknot using **Proc.** ShortestMatchingLength. First, we consider all prefixes of w up-to length $\frac{n}{2}$—candidates for being w_1 in the pseudoknot—and compute the set $w_p[i]$ of the shortest length of the matching prefix of each index using **Proc.** ShortestMatchingLength with $w = w_1 w_2 \cdots w_{\frac{n}{2}}$ and $T = w^R$. Next, we similarly consider all suffixes of w up-to length $\frac{n}{2}$—candidates for being w_3^R in the pseudoknot—and compute the set $w_s[i]$ of the shortest length of the matching suffix for each index $1 \leq i \leq n$ using **Proc.** ShortestMatchingLength with $w = w_{\frac{n}{2}+1} \cdots w_{n-1} w_n$ and $T = w$.

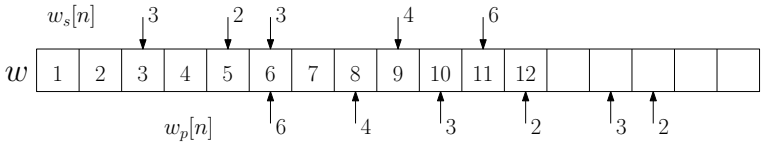


Fig. 4: An example of running **Proc.** ShortestMatchingLength for checking whether or not w is a pseudoknot.

Fig. 4 is an example of running **Proc.** ShortestMatchingLength for a string w and obtain $w_p[n]$ and $w_s[n]$. In this example, because of $w_s[9]$ and $w_p[10]$, we know that w is a pseudoknot. However, a pair of $w_s[5]$ and $w_p[6]$ is invalid since, at index 6, w cannot have a prefix of size 6 ($=w_p[6]$). Similarly, a pair of $w_s[11]$ and $w_p[12]$ is invalid for checking the pseudoknot structure for w since, after index 11, w cannot have a $w_1^R w_4 w_3^R$, where $|w_3^R| = 6$.

Lemma 1. *Given a string w of length n, we can determine whether or not w is a pseudoknot in $O(n)$ time.*

Note that if w is a pseudoknot, then we can find all indices i of w such that $w(1,i) = x_1 x_2 x_3$ and $w(i+1,n) = x_1^R x_4 x_3^R$ from the algorithm. Let $I_{pk}(w)$ be the set of such indices.

Corollary 1. *Given two pseudoknot strings x and y, we can determine whether or not both $x, y \in \mathbb{PK}_{\mathbb{R}}(w)$ for a string w in linear-time in the size of x and y. We can also identify such w using $I_{pk}(x)$ and $I_{pk}(y)$ within the same runtime.*

We next consider a problem of determining whether or not w is in $\mathbb{PK}_{\mathbb{R}}(L(A))$ of a given FA A. Our approach is simple: we read w character by character with A and find all indices j of w when we enter a final state of A while reading w. Namely, $w(1,j) \in L(A)$. Let $I_p(w, A)$ be the set of such indices.

Lemma 2. *Given a string w and an FA A,*

$$w \in \mathbb{PK}_{\mathbb{R}}(L(A)) \text{ iff } I_{pk}(w) \cap I_p(w, A) \neq \emptyset.$$

Note that a pseudoknot string may have several different pseudoknots. Therefore, even if $w(1,i) = x_1x_2x_3$ and $w(i+1,n) = x_1^R x_4 x_3^R$ for an index i of w, $w(1,i)$ may not be accepted by A. This is why we have considered all possible indices in $I_{pk}(w)$. We now establish the following result based on pseudoknot checking algorithm and Lemma 2.

Theorem 1. *Given a string w of size n and an FA A of size $m = |A|$, we can determine whether or not $w \in \mathbb{PK}_{\mathbb{R}}(L(A))$ in $O(mn)$ time. If A is a DFA, then the runtime becomes $O(n)$.*

Proof. It takes $O(n)$ time to compute $I_{pk}(w)$ and $O(mn)$ time to compute $I_p(w, A)$. If A is a DFA, then we can compute $I_p(w, A)$ in $O(n)$ time. □

Note that pseudoknots of RNA are closely related with the frameshifting mutation of protein expressions and commonly found in viral genomes, in particular influenza virus [7]. This leads researchers to consider the structural comparison among several sequences to find the common mutational pattern, in particular, pseudoknots [6]. Here, we investigate a necessary condition of $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ for two strings x and y and show that it is decidable to check whether or not two strings have a common pseudoknot in $\mathbb{PK}_{\mathbb{R}}(x)$ and $\mathbb{PK}_{\mathbb{R}}(y)$.

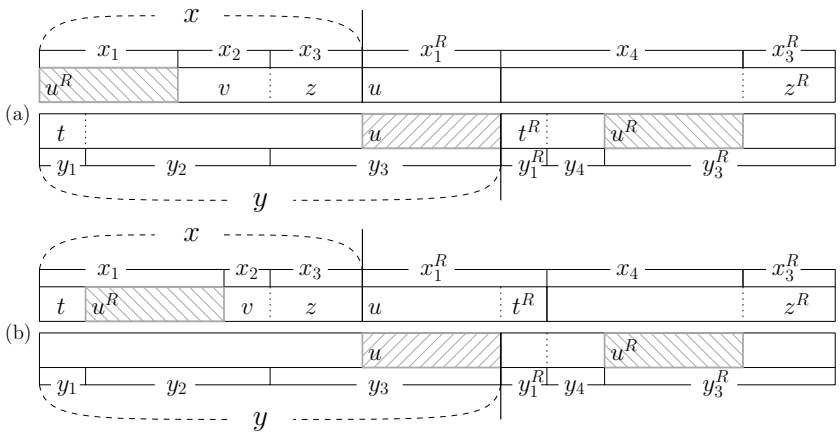


Fig. 5: An example of two strings x and y such that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$. First, x is a prefix of y. Second, the longer part u (slanted line box in the figure) of y appears as (a) a prefix of x or (b) an infix (but not prefix) of x in the reversed form u^R, where $t, v, z \in \Sigma^+$.

Let x and y be two strings, where $|x| < |y|$. Fig. 5 shows that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ if and only if x is a prefix of y, and $y(|x|+1, |y|)$ (= u in the

figure) appears as an infix of $x(1, |x| - 2)$—we consider $x(1, |x| - 2)$ to ensure $t, v, z \in \Sigma^+$, if exists. There are two possible cases for being an infix as follows:

(a) Since u^R appears as a prefix of $x(1, |x|-2)$, we can select an arbitrary prefix t of $x(1, |x| - 2)$ for y_1 as depicted in Fig. 5 (a).
(b) Since u^R appears an an infix (but not a prefix) of $x(1, |x| - 2)$, we can select the prefix t of x_1 such that $x_1 = tu^R$ for y_1 as depicted in Fig. 5 (b).

We can check this in $O(|x|)$ time using the KMP algorithm [16] and, therefore, obtain the following result.

Lemma 3. *Given two strings x and y such that $|x| < |y|$, we can determine whether or not $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$ in $O(|x|)$ time.*

Proof. The proof is immediate from Fig. 5. □

We know if two strings have a common pseudoknot in $\mathbb{PK}_{\mathbb{R}}(x)$ and $\mathbb{PK}_{\mathbb{R}}(y)$. We next investigate the inclusion between $\mathbb{PK}_{\mathbb{R}}(x)$ and $\mathbb{PK}_{\mathbb{R}}(x)$ when $|x| < |y|$.

Lemma 4. *Given two strings x and y, if $|x| < |y|$, then it is impossible that $\mathbb{PK}_{\mathbb{R}}(x) \subset \mathbb{PK}_{\mathbb{R}}(y)$.*

Given a string z, it is straightforward to verify that $\mathbb{PK}_{\mathbb{R}}(z)$ is regular and $\mathbb{PK}_{\mathbb{R}}(z)$ is infinite from the definition of the operation. We consider the case of applying the $\mathbb{PK}_{\mathbb{R}}$ operation on the resulting language several times, and prove that the iterated $\mathbb{PK}_{\mathbb{R}}$ does not preserve the regularity.

Theorem 2. *There exists a string z such that ${\mathbb{PK}_{\mathbb{R}}}^2(z)$ and ${\mathbb{PK}_{\mathbb{R}}}^*(z)$ are not regular.*

Proof. By choosing $\Sigma = \{a, ¢, \$\}$ and $z = ¢a\$$, it can be verified that ${\mathbb{PK}_{\mathbb{R}}}^2(z)$ and ${\mathbb{PK}_{\mathbb{R}}}^*(z)$ are not regular. □

4 Pseudoknot-generating operation on languages

We investigate the properties of pseudoknot on a set of strings. The pseudoknot operation on a string implies that we generate a pseudoknot family related by a common structural motif for pseudoknot. Note that a given string expands and becomes a pseudoknot string by the pseudoknot operation on a string. We extend this view point into languages in which a set of strings represents a set of all subsequences of a long RNA sequence. This implies that the pseudoknot operation on a language generates all possible pseudoknots that may partially occur as a mutation.

4.1 Closure and decision properties of the pseudoknot-generating operation

We first consider the closure property of the pseudoknot operation and determine whether or not two sets of pseudoknots on languages contain a common string.

Theorem 3. *Regular and context-free languages are not closed under* $\mathbb{PK}_{\mathbb{R}}$.

Next, given regular languages L and R we consider the problem of checking whether or not there is a pseudoknot generated both by a string of L and a string of R. Note that, by Theorem 3, we know that $\mathbb{PK}_{\mathbb{R}}(L)$ need not be even context-free in general. This means that we cannot simply first produce a representation of the languages $\mathbb{PK}_{\mathbb{R}}(L)$ and $\mathbb{PK}_{\mathbb{R}}(R)$, respectively, and then check whether or not they have a non-empty intersection. Instead, our algorithm is based directly on finite automata for the original language L and R.

Let $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ be two FAs for L and R. Then, we first construct an FA $C = (Q_A \times Q_B, \Sigma, \delta_C, s_A \times s_B, F_A \times F_B)$ for $L(A) \cap L(B)$ by the standard Cartesian product, where

$$\delta_C((p, q), a) = \{(p', q') \mid p' \in \delta_A(p, a) \text{ and } q' \in \delta_B(q, a)\}.$$

Our algorithm is similar to the idea illustrated in Fig. 5: We check if there exists a pair of strings—say $x \in L(A)$, $y \in L(B)$ and $|x| < |y|$ (the other case is symmetric)—such that $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$. Since x is a prefix of y, there exists a path from s_B to a nonfinal state q that spells out x in B. We search for such paths in C. For each state (f, q) of C, where $f \in F_A, q \in Q_B$, we define two FAs as follows:

- $\overleftarrow{C_{f,q}} = (Q_A \times Q_B, \Sigma, \delta_C, s_A \times s_B, \{(f, q)\})$; in other words, (f, q) is the only final state of C.
- $\overrightarrow{B_q} = (Q_B, \Sigma, \delta_B, q, F_B)$; in other words, q is the new start state of B.

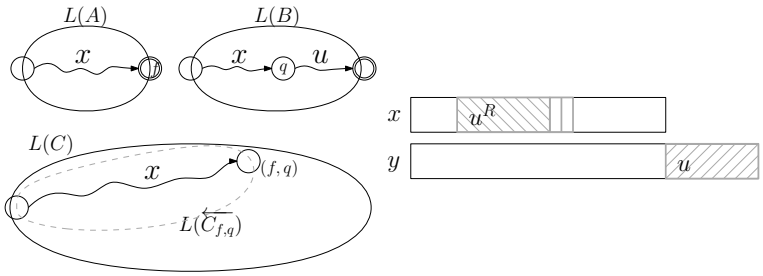


Fig. 6: An example of two FAs A and B such that $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$. Note that u^R is an infix of the string $x(2, |x| - 2)$.

Lemma 5. *There exists a state (f, q) of C such that*

$$L(\overleftarrow{C_{f,q}}) \quad \cap \quad \Sigma^* \cdot (L(\overrightarrow{B_q})^R \cdot \Sigma^2) \cdot \Sigma^* \quad \neq \quad \emptyset$$

or a state (p, f') of C such that

$$L(\overleftarrow{C_{p,f'}}) \quad \cap \quad \Sigma^* \cdot (L(\overrightarrow{A_p})^R \cdot \Sigma^2) \cdot \Sigma^* \quad \neq \quad \emptyset$$

if and only if $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$.

Once we have an intersection FA C, there are at most $|Q_A||Q_B|$ states in the form of (f, q) or (p, f'). Then, for each state, say (f, q), we need to check whether or not $L(\overleftarrow{C_{f,q}}) \cap \Sigma^* \cdot (L(\overrightarrow{B_q})^R \cdot \Sigma^2) \cdot \Sigma^*$ is empty. Since the size of $C_{f,q}$ is at most $|A||B|$ and the size of $\overrightarrow{B_q}$ is at most $|B|$, it takes $O(|A||B|^2)$ time. Therefore, in the worst-case, the total runtime is

$$O(n^2) \text{ (the number of states) } \times O(n^6) \text{ (intersection test) } = O(n^8),$$

where n is the maximum number of states between A and B.

Theorem 4. *Given two FAs A and B, we can determine whether or not*

$$\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$$

in polynomial-time.

Often we need to verify if there exists a pseudoknot in the input set—a set of strings. In biology, a RNA sequence might first fold into non-pseudoknot, and then form a more complex structure including pseudoknots. According to this phenomenon, Jabbari and Condon [10] considered non-pseudoknots for predicting pseudoknots capturing all possible pre-structures of pseudoknots.

When an input set has a finite number of elements, we may check them one by one. However, if the set is infinite, then we need a better algorithm. We consider this problem when the set is a regular language. Before we present an algorithm, we define the inverse restricted pseudoknot-generating operation ${\mathbb{PK}_{\mathbb{R}}}^{-1}$ to be

$${\mathbb{PK}_{\mathbb{R}}}^{-1}(w) = \{x_1x_2x_3 \mid w \in x_1x_2x_3x_1^Rx_4x_3^R, x_1, x_2, x_3, x_4 \in \Sigma^+\}.$$

Lemma 6. *Let A be an NFA. Then there exists an NFA A' such that*

$$L(A') = {\mathbb{PK}_{\mathbb{R}}}^{-1}(L(A)).$$

Corollary 2. *Given an NFA A, we can determine if A accepts a pseudoknot.*

Note that it is decidable to determine whether or not a given regular language contains a pseudoknot. Here, we contrast the result of Corollary 2 by showing that it is undecidable whether or not a context-free language contains a pseudoknot. We use a reduction from the *Post Correspondence Problem* (PCP) [20]. Recall that an instance of PCP consists of two lists of strings $((u_1, \ldots, u_n), (v_1, \ldots, v_n))$, $u_i, v_i \in \Sigma^*$, $1 \leq i \leq n$, and a solution of this instance is a sequence of integers $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. It is well known that PCP is unsolvable [20].

Proposition 1. *For a given context-free language L, it is undecidable whether or not L contains a pseudoknot.*

4.2 Pseudoknot-free languages

Analogously with the definition of restricted code classes, such as prefix- or suffix-codes [12], we define that a language L is $\mathbb{PK}_{\mathbb{R}}$-free (informally just pseudoknot-free) if no string of L is a pseudoknot generated by another string of L.

Definition 1. *We say that a language L is $\mathbb{PK}_{\mathbb{R}}$-free if $L \cap \mathbb{PK}_{\mathbb{R}}(L) = \emptyset$.*

In DNA coding applications, pseudoknots are in general undesirable because they can result in undesired bonds in DNA sequences [13, 14]. This means that if we can efficiently check the property of $\mathbb{PK}_{\mathbb{R}}$-freeness, it might be worthwhile to add a preprocessing stage for predicting pseudoknot-structures. Note that some approaches for prediction pseudoknots consider also "non-pseudoknots" because an RNA sequence can fold a non-pseudoknot to form a pseudoknot.

We consider the case when L is regular and show that we can decide whether or not L is $\mathbb{PK}_{\mathbb{R}}$-free. We use a similar construction for constructing an FA for ${\mathbb{PK}_{\mathbb{R}}}^{-1}$.

Lemma 7. *Let A be an FA. Then there exists an NFA A' that accepts a set of strings $u = u_1u_2u_3$ such that $u_1^R u_4 u_3^R u_1 u_2 u_3 \in L(A)$.*

Theorem 5. *Given an FA A, we can determine whether or not $L(A)$ is $\mathbb{PK}_{\mathbb{R}}$-free in polynomial-time.*

Here, we observe that deciding $\mathbb{PK}_{\mathbb{R}}$-freeness of a context-free language is undecidable based on Proposition 1.

Theorem 6. *For a given context-free language L it is undecidable whether or not L is $\mathbb{PK}_{\mathbb{R}}$-free.*

5 Conclusions

We have considered one of RNA structures called pseudoknot and specific phenomenon in which a sequence expands itself and forms a pseudoknot. We have defined the restrict version of the pseudoknot-generating operation: For a string x, $\mathbb{PK}_{\mathbb{R}}(x)$, roughly speaking, consists of all possible continuations of x that can fold back onto x to form a pseudoknot.

We have investigated (closure-)properties of pseudoknot-generating operation on a string and designed linear-time algorithm for determining whether or not given string is a pseudoknot. We have shown that for two strings x and y, it is decidable whether or not $\mathbb{PK}_{\mathbb{R}}(x) \cap \mathbb{PK}_{\mathbb{R}}(y) \neq \emptyset$. Moreover, we have examined the pseudoknot-generating operation on languages, and showed that regular and context-free languages are not closed under pseudoknot-generating. On the other hand, we have established that given two FAs A and B, it is decidable whether or not $\mathbb{PK}_{\mathbb{R}}(L(A)) \cap \mathbb{PK}_{\mathbb{R}}(L(B)) \neq \emptyset$ in polynomial-time in the size of A and B. Furthermore, we have shown that it is decidable whether or not a given regular language is $\mathbb{PK}_{\mathbb{R}}$-free in polynomial-time. However, it is undecidable to determine whether or not a given context-free language is $\mathbb{PK}_{\mathbb{R}}$-free.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.
2. T. Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1):45–62, 2000.
3. I. Brierley, P. Digard, and S. C. Inglis. Characterization of an efficient coronavirus ribosomal frameshifting signal: requirement for an RNA pseudoknot. *Cell*, 57(4):537–547, 1989.
4. A. Condon, B. Davy, B. Rastegari, S. Zhao, and F. Tarrant. Classifying RNA pseudoknotted structures. *Theoretical Computer Science*, 320(1):35–50, 2004.
5. R. M. Dirks, M. Lin, E. Winfree, and N. A. Pierce. Paradigms for computational nucleic acid design. *Nucleic Acids Research*, 32(4):1392–1403, 2004.
6. G. Doose and D. Metzler. Bayesian sampling of evolutionarily conserved RNA secondary structures with pseudoknots. *Bioinformatics*, 28(17):2242–2248, 2012.
7. Z. Du and D. W. Hoffman. An NMR and mutational study of the pseudoknot within the gene 32 mRNA of bacteriophage t2: insights into a family of structurally related RNA pseudoknots. *Nucleic Acids Research*, 25(6):1130–1135, 1997.
8. P. A. Evans. Finding common RNA pseudoknot structures in polynomial time. *Journal of Discrete Algorithms*, 9(4):335–343, 2011.
9. D. P. Giedroc, C. A. Theimer, and P. L. Nixon. Structure, stability and function of RNA pseudoknots involved in stimulating ribosomal frameshifting. *Journal of Molecular Biology*, 298(2):167–185, 2000.
10. H. Jabbari and A. Condon. A fast and robust iterative algorithm for prediction of RNA pseudoknotted secondary structures. *BMC Bioinformatics*, 15(1):147, 2014.
11. T. Jiang, G. Lin, B. Ma, and K. Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–388, 2002.
12. H. Jürgensen and S. Konstantinidis. Codes. In *Word, Language, Grammar*, volume 1 of *Handbook of Formal Languages*, pages 511–607. 1997.
13. L. Kari, S. Konstantinidis, and S. Kopecki. Transducer descriptions of DNA code properties and undecidability of antimorphic problems. *arXiv:1503.00035*, 2015.
14. L. Kari and K. Mahalingam. Watson–crick palindromes in DNA computing. *Natural Computing*, 9(2):297–316, 2010.
15. L. Kari and S. Seki. On pseudoknot-bordered words and their properties. *Journal of Computer and System Sciences*, 75(2):113–121, 2009.
16. D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.
17. M. Möhl, S. Will, and R. Backofen. Fixed parameter tractable alignment of RNA structures including arbitrary pseudoknots. In *Combinatorial Pattern Matching*, pages 69–81, 2008.
18. P. Rinaudo, Y. Ponty, D. Barth, and A. Denise. Tree decomposition and parameterized algorithms for RNA structure-sequence alignment including tertiary interactions and pseudoknots. In *Algorithms in Bioinformatics*, pages 149–164. 2012.
19. A. A. Saraiya, T. N. Lamichhane, C. S. Chow, J. S. Jr, and P. R. Cunningham. Identification and role of functionally important motifs in the 970 loop of escherichia coli 16s ribosomal RNA. *Journal of Molecular Biology*, 376(3):645–657, 2008.
20. J. Shallit. *A second course in formal languages and automata theory*, volume 179. Cambridge University Press Cambridge, 2009.