Scalable Computation of Milnor Invariants of String Link

Yoshimasa Takabatake[1], Tetsuji Kuboyama[2]
Akira Yasuhara[3], Hiroshi Sakamoto[1]

[1] Kyushu Institute of Technology, 680-4 Kawazu, Iizuka-shi, Fukuoka, 820-8502,
[2] Gakushuin University, 1-5-1 Mejiro Toshima Tokyo, 171-8588,
[3] Tokyo Gakugei University, Koganei-shi, Tokyo 184-8501, Japan
takabatake@donald.ai.kyutech.ac.jp, ori-ds2013@tk.cc.gakushuin.ac.jp,
yasuhara@u-gakuei.ac.jp, hiroshi@ai.kyutech.ac.jp

Abstract. We propose an efficient algorithm for computing *Milnor invariants* of string[1] links in Knot Theory. Although Milnor invariants are a significant tool for studying string links, little has been studied due to their computational intractability. Given string links in a 3-dimensional space, the invariants is obtained in the form of polynomials in non-commuting variables, i.e., a string formed by such variables. Taking advantage of the non-commutativity, we handle the computation using a compact representation for string, called *grammar compression.* The size of the string exponentially increases in the computation process even if a resulting Milnor invariant is quite small. To address this problem, we propose a novel algorithm for computing Milnor invariants by leveraging a state-of-the-art grammar compression algorithm so that we can compute extremely long polynomials without explicit expansion. We show that our algorithm significantly improves a naive algorithm.

1 Introduction

In this study, we would produce a useful tool analyzing *string links* in low-dimensional topology adopting techniques in computer science. *Milnor invariant* is a family of invariants of a disjoint union of n curves, called n-string link, in the cylinder [6, 14, 15]. For each sequence $I = i_1, \ldots, i_m$ in $\{1, \ldots, n\}$, the Milnor invariant $\mu_L(I)$ of n-string link L is defined. We call the length m of the sequence I the *length* of $\mu_L(I)$. An algorithm for computing Milnor invariants is given in [15]. The length-m Milnor invariants are given as coefficients of degree-$(m-1)$ polynomials in non-commuting variables with integer coefficients. Although the algorithm itself is very simple, the number of multiplications in a polynomial exponentially increases according to the length of required Milnor invariant. In fact, to the best of the authors' knowledge, no efficient algorithm for computing

[1] In this paper, we use the word *string* in two meanings: (1) a curve in 3-dimensional space; (2) a sequence of letters. To avoid confusion, we mean (1) by sans serif fonts "string," and (2) otherwise.

Milnor invariants has been proposed and only very few examples have been reported so far due to the intractability although the Milnor invariant is one of the most important invariants for the classical link theory (Knot Theory). It is extremely challenging to develop an efficient algorithm for computing Milnor invariants to shed light on uncharted properties of string links in Knot Theory.

A polynomial in Milnor invariants is regarded as a set of strings due to its non-commutativity. A string is represented by a *context-free grammar* (CFG), that is a compact representation of string called *grammar compression*. Thus, we propose a fast algorithm for computing Milnor invariants by using a string compression algorithm by CFG. Our algorithm allows us to compute Milnor invariants for large length.

Given a link, it is transformed to a set of sequences of non-commuting variables, called *longitudes*. Iteration of their recursive expansion generates the k-th longitude, which is an intermediate for getting the final Milnor invariant. Because the size of the k-th longitude grows exponentially according to k, the naive algorithm that explicitly generates the intermediates cannot compute the invariants for non trivial links. In this paper, we propose a novel algorithm for computing Milnor invariants without explicit expansion of longitudes by taking advantage of grammar compression.

Assuming a set of symbols, a sequence formed by such symbols is called *string*. Given a string S from a set of symbols, a CFG G is considered as a compression of S if $size(G) < size(S)$ and G deterministically derives S. Many interesting theories [8, 11, 16, 18, 2, 17] and practical algorithms [9, 10, 13, 1] for grammar compression have been widely introduced as well as its application to pattern matching and information retrieval [4, 12, 3, 20, 19, 5, 7].

There is an interesting correspondence between generating the k-th longitude for a string link and generating a string S using G, i.e., we can translate generating the k-th longitude to deriving S by a CFG G. Leveraging this correspondence, we can avoid the explicit generation of the k-th longitude, whose size (possibly) grows exponentially according to k. Instead of explicit expansion, we can construct a parsing derivation tree of S in which no variable appears twice or more. The proposed algorithm traverses this tree in depth-first order. The size of the parsing derivation tree is quite smaller than the size of the explicit derivation tree, i.e., the length of S itself.

Finally, we examine the performance of our method using non-trivial string links. From our experiments, we confirm that our algorithm significantly improves the scalability of computing Milnor invariants compared to a naive algorithm.

2 Preliminary

We introduce the basic notions of string links (or links for short). When a link is given, it is transformed into a set of substitution rules called longitudes, that derives Milnor invariants of the link. The derivation can be simulated by a CFG, because the derivation of Milnor invariants is also *context-free*. Thus, we introduce the notion of grammar compression for efficiently computing the Milnor invariants.

2.1 String link and their diagram

Let D^2 be the unit disk in 2-dimensional Euclidean space $\mathbb{R}^2$ equipped with n marked points $x_1, \ldots, x_n$ $(n \geq 1)$ lying on the diameter in the x-axis of $\mathbb{R}^2$. An n-*string link* is a disjoint union of n curves $K_1, \ldots, K_n$ in the cylinder $D^2 \times [0,1] (\subset \mathbb{R}^2 \times \mathbb{R} = \mathbb{R}^3)$ such that each K_i runs from bottom x_i to top x_i, for example see Fig. 1(a). We say that two n-string links are *equivalent* if one can be deformed into the other continuously with fixing the end points. Let π be the projection from $\mathbb{R}^3$ to the xz-plane, that is $\pi(x, y, z) = (x, z)$. Then for an n-string link L, the restriction map of π from L to $\pi(L)$ is injective except for finitely many points $c_1, \ldots, c_m$ in $\pi(L)$ (see Fig. 1(b)). For each c_j, there are two points c_{j1} and c_{j2} in L whose images are c_j, where the distance from c_{j1} to the xz-plane is smaller than the distance from c_{j2}. We delete small neighborhoods of c_{j1} $(j = 1, \ldots, m)$ from L, then the projection image of resulting curves is called *diagram* of L (see Fig. 1(c)). We call the each component of the diagram a *segment*. Since the diagram of L can be regarded as L itself, we denote the diagram of L by L.

2.2 Algorithm for computing the Milnor invariants

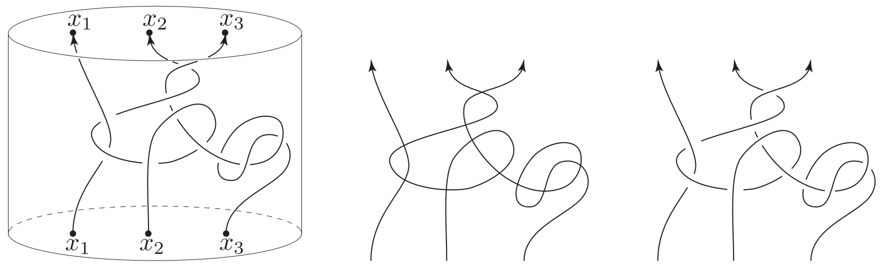
Given a diagram of string link, it is transformed[2] into a binary relation, called *initial data*, over a set of characters. The initial data is equivalent to a set L of production rules of a CFG. Milnor invariant [15] is computed by expansion of L. In this subsection, we recalled these notions.

Initial data. Let $L = K_1 \cup \cdots \cup K_n$ be the diagram of an n-string link.

1. For the segments in K_i $(i = 1, \ldots, n)$, give labels from $a_{i,1}$ to a_{i,r_i} by traveling along K_i from bottom x_i to top x_i, where r_i is the number of the components of arcs in K_i. (See Fig. 2(Segments) for example.)
2. For each $i \in \{1, \ldots, n\}$, let $u_\ell \in \{a_{jk} \mid 1 \leq j \leq n,\ 1 \leq k \leq r_j\}$ be the labeled arc between $a_{i,l}$ and $a_{i,l+1}$. We assign $\varepsilon(\ell) \in \{-1, 1\}$ as the following rule: If we see a_ℓ on the right-hand side when traveling along u_ℓ, then we define $\varepsilon(\ell) = 1$, otherwise $\varepsilon(\ell) = -1$. Then, we have the string $\ell_i = u_1^{\varepsilon(1)} u_2^{\varepsilon(2)} \cdots u_{r_i-1}^{\varepsilon(r_i-1)}$. We denote by $\ell_i[j]$ the partial string $u_1^{\varepsilon(1)} \cdots u_{j-1}^{\varepsilon(j-1)}$ of ℓ_i and by $\ell_i[j]^{-1}$ the partial string $u_{j-1}^{-\varepsilon(j-1)} \cdots u_1^{-\varepsilon(1)}$. (See Fig. 2(Initial data).)

We call the set of strings ℓ_i $(i = 1, \ldots, n)$ the initial data of L, and denote this set also by L if there is no confusion. The Milnor invariants are obtained from the initial data.

[2] This transformation can be performed manually, or an image recognition algorithm is available in case of complicated diagram.



(a) 3-string link (b) projection image of link (c) link diagram

Fig. 1. An example of 3-string link

Longitude with depth *d*. For each non negative integer d, the map f_d from the initial data to a set of strings consisting of $a_{i,j}^{\pm 1}$ $(1 \leq i \leq n,\ 1 \leq j \leq r_i)$ is defined as follows:

(0) f_0 is the identity map;
(1) f_1 maps each $a_{i,j}$ to $\ell_i[j]^{-1}a_{i,1}\ell_i[j]$ for $j \geq 2$ and $f_1(a_{i,1}) = a_{i,1}$; and
(d) f_d is the composition $f_1 \circ \cdots \circ f_1$, where f_1 occurs d times.

Next, we consider the map g from the set of strings consisting of $a_{i,j}^{\pm 1}$ $(1 \leq i \leq n,\ 1 \leq j \leq r_i)$ to the set of strings consisting of $\alpha_1^{\pm 1}, \dots, \alpha_n^{\pm 1}$ defined as $g(a_{i,j}) = \alpha_i$ for each i.

For each d, we have the set of strings $g \circ f_d(\ell_1), \dots, g \circ f_d(\ell_n)$ which are composed of $\alpha_1^{\pm 1}, \dots, \alpha_n^{\pm 1}$. We call $\alpha_i^{-w_i} g \circ f_d(\ell_i)$ is the *i-th longitude with depth d* and denote it by $\ell_i^{(d)}$, where w_i is the sum of indices of α_i in $g \circ f_0(\ell_i)$. (Here the i-th longitude can be assume to be an element of the free group generated by $\alpha_1, \dots, \alpha_n$, that is we may assume that $\alpha_j\alpha_j^{-1}$ and $\alpha_j^{-1}\alpha_j$ are empty string.)

Milnor invariant. Let E_k be the map from the set of strings consisting of $\alpha_1^{\pm 1}, \dots, \alpha_n^{\pm 1}$ to the set of the polynomials in n non-commuting variables $X_1, \dots, X_n$ with integer coefficients defined as

$$E_k(\alpha_i) = 1 + X_i,\ E_k(\alpha_i^{-1}) = 1 - X_i + X_i^2 - X_i^3 + \cdots + (-1)^k X_i^k,$$

where let k be d in general. Then, we have

$$E_k(\ell_i^{(d)}) = 1 + \sum_{1 \leq r \leq k} \mu_L(j_1, \dots, j_r, i) X_{j_1} \cdots X_{j_r} + (\text{terms of degree} > k).$$

Thus, a coefficient $\mu_L(j_1, \dots, j_r, i)$ is defined for each sequence $j_1, \dots, j_r, i$ $(1 \leq r \leq d)$ in $\{1, \dots, n\}$. We call $\mu_L(j_1, \dots, j_r, i)$ is the Milnor invariant for L with a sequence $j_1, \dots, j_r, i$. It is known that two equivalent string links have the same Milnor invariants, but not necessarily vice versa.

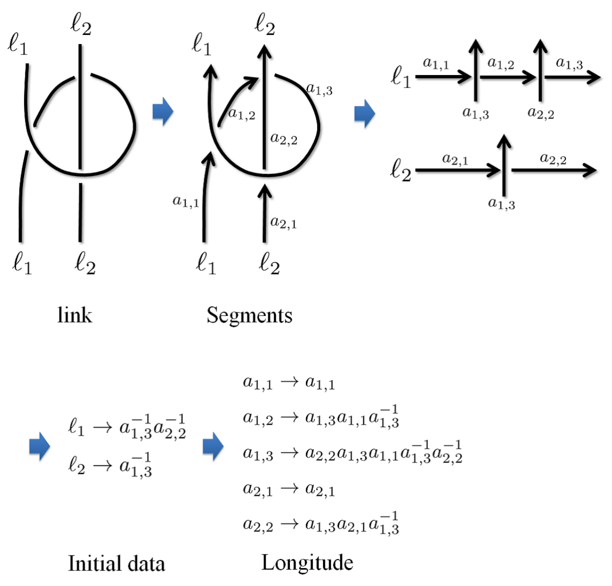

ℓ_1
ℓ_2
link
Segments
$a_{1,1}$
$a_{1,2}$
$a_{1,3}$
$a_{2,1}$
$a_{2,2}$
$\ell_1 \to a_{1,3}^{-1}a_{2,2}^{-1}$
$\ell_2 \to a_{1,3}^{-1}$
Initial data
$a_{1,1} \to a_{1,1}$
$a_{1,2} \to a_{1,3}a_{1,1}a_{1,3}^{-1}$
$a_{1,3} \to a_{2,2}a_{1,3}a_{1,1}a_{1,3}^{-1}a_{2,2}^{-1}$
$a_{2,1} \to a_{2,1}$
$a_{2,2} \to a_{1,3}a_{2,1}a_{1,3}^{-1}$
Longitude

Fig. 2. An example of string link, its segments, initial data, and longitudes. The longitudes of $a_{i,j}^{-1}$ are omitted because of the symmetry.

2.3 Grammar compression

By taking advantage of the property that a polynomial in Milnor invariants is regarded as a set of strings due to its non-commutativity, we apply the grammar compression to compute Milnor invariants. Therefore, we briefly introduce the framework that we employ in our algorithm as follows.

For a finite set C, $|C|$ denotes its cardinality. *Alphabet* Σ is a finite set of letters and $\sigma = |\Sigma|$ is a constant. $\mathcal{N}$ is a recursively enumerable set of *nonterminals* with $\Sigma \cap \mathcal{N} = \emptyset$. A sequence of symbols from $\Sigma \cup \mathcal{N}$ is called a *string*. The set of all possible strings from Σ is denoted by Σ^*. For a string S, the expressions $|S|$, $S[i]$, and $S[i,j]$ denote the length of S, the i-th symbol of S, and the substring of S from $S[i]$ to $S[j]$, respectively. Let $[S]$ be the set of symbols composing S.

A CFG is represented by $G = (\Sigma, N, P, X_s)$, where N is a finite subset of $\mathcal{N}$, P is a finite subset of $N \times (\Sigma \cup N)^*$, and $X_s \in N$. A member of P is called a production rule and X_s is called the start symbol. The set of strings in Σ^* derived from X_s by G is denoted by $L(G)$.

A CFG G is called *admissible* if, for each $A \in N$, exactly one $A \to \alpha \in P$ exists and $|L(G)| = 1$. An admissible G deriving a string $S \in \Sigma^*$ is called a grammar compression of S.

The derivation tree T of G is represented by a rooted ordered tree satisfying (1) for each $A \to \alpha \in P$, there is an internal node in T labeled with $A \in N$ whose children are $X_1, X_2, \ldots, X_k$ such that $X_1X_2\cdots X_k = \alpha$, and (2) the *yield* of T, i.e., the sequence of corresponding leaves is equal to S.

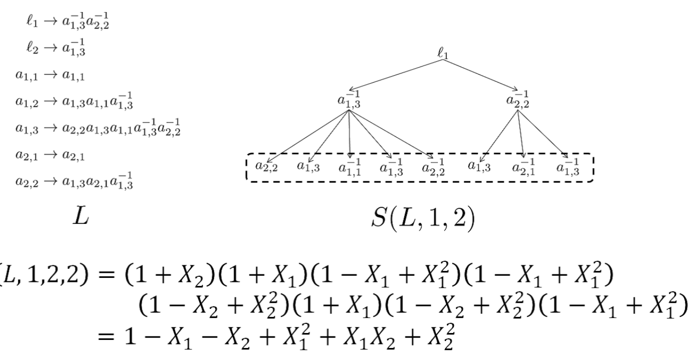

$\ell_1 \to a_{1,3}^{-1}a_{2,2}^{-1}$
$\ell_2 \to a_{1,3}^{-1}$
$a_{1,1} \to a_{1,1}$
$a_{1,2} \to a_{1,3}a_{1,1}a_{1,3}^{-1}$
$a_{1,3} \to a_{2,2}a_{1,3}a_{1,1}a_{1,3}^{-1}a_{2,2}^{-1}$
$a_{2,1} \to a_{2,1}$
$a_{2,2} \to a_{1,3}a_{2,1}a_{1,3}^{-1}$
L
ℓ_1
$a_{1,3}^{-1}$
$a_{2,2}^{-1}$
$a_{2,2}$ $a_{1,3}$ $a_{1,1}^{-1}$ $a_{1,3}^{-1}$ $a_{2,2}^{-1}$ $a_{1,3}$ $a_{2,1}^{-1}$ $a_{1,3}^{-1}$
$S(L,1,2)$
$M(L,1,2,2) = (1+X_2)(1+X_1)(1-X_1+X_1^2)(1-X_1+X_1^2)$
$(1-X_2+X_2^2)(1+X_1)(1-X_2+X_2^2)(1-X_1+X_1^2)$
$= 1-X_1-X_2+X_1^2+X_1X_2+X_2^2$

Fig. 3. The Milnor invariant $M(L, q, d, k)$, the derivation of $T(L, q, d)$ for $q = 1$, $d = 2$, $k = 2$, and the set L of longitudes given in Fig. 2.

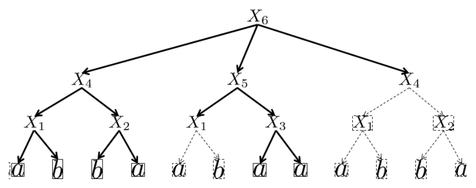


Fig. 4. A partial parse tree deriving the string *abbaabaaabba*. The partial parse tree is shown by solid lines. Broken lines are removed from the original derivation tree.

Definition 1. *(Rytter [16]) A partial parse tree is an equivalent representation of a derivation tree T built by traversing T in a depth-first manner and pruning out all the descendants under every node of a nonterminal appearing no less than twice.*

2.4 Computation of Milnor invariants

In the computation of Milnor invariants, we have the input L with three parameters q, d, and k, where L is the initial data described in Sec. 2.2, d is the depth of a longitude, and k is the order of the polynomial $E_k(\ell_q^{(d)})$ for the q-th longitude. In general case of Milnor invariants, we can assume $d = k$. However, from practical point of view, our algorithm independently takes such d and k to control the computational cost. In addition, we abuse the notation of $\ell_i = \alpha_i$; we also regard it as a production rule $\ell_i \to \alpha_i$, and L as the set of the corresponding production rules. Using the initial data L, we can construct the longitudes $a_{ij}^{\pm 1}$ for the map f_1 in Sec. 2.2. For example, the longitudes $a_{ij}^{\pm 1}$ are shown in Fig.2 (Longitude).

By $S(L, q, d)$, we denote the set of strings for the q-th longitude with depth d for the initial data L, i.e., $\ell_q^{(d)}$. Also, by $M(L, q, d, k)$, we denote the polynomial $E_k(\ell_q^{(d)})$.

Given a set L of longitudes for an n-strings link, we can compute the Milnor invariant determined by $M(L,q,d,k)$ for q-th string in the link ($1 \leq q \leq n$), and depth d as follows.

Initially, Let $S(L,q,0)$ be α_q for $\ell_q = \alpha_q \in L$. When we have $S(L,q,d)$, we can compute $S(L,q,d+1)$ by replacing each $v_{i,j} \in \{a_{i,j}, a_{i,j}^{-1}\}$ in $T(L,q,d)$ according to the production rules in L. In other words, $S(L,q,d)$ is the *yield* of the derivation tree such that the depth of each leaf is just d and ℓ_q is the root. An example of $S(L,q,d)$ is shown in Fig. 3($S(L,q,d)$).

Finally, we expand $S(L,q,d)$ replacing $a_{i,j}$ by $1 + X_i$ and $a_{i,j}^{-1}$ by $1 - X_i + X_i^2 + \cdots + (-1)^k X_i^k$, and obtain the polynomial up to the order k. Then, we obtain $M(L,q,d,k)$. An example of $M(L,q,d,k)$ is show in Fig. 3($M(L,q,d,k)$).

3 Algorithm for Fast Milnor Invariants Extraction

First, we consider a naive algorithm for computing $M(L,q,d,k)$ with a set L of longitudes for an n-string link, the q-th string in the link ($1 \leq q \leq n$), and the depth k of derivation.

A naive algorithm constructs $S(L,q,d)$ explicitly, and substitutes corresponding polynomials to $a_{i,j}$ or $a_{i,j}^{-1}$ for obtaining $M(L,q,d,k)$. On a standard RAM model, the computation time is bounded by $O(\mathit{prod})$ for the number *prod* of products of integers executed. In the case of the naive algorithm, $\mathit{prod} = \Theta(k|M||S|)$, where $|M|$ is the size of $M(L,q,d,k)$, and $|S|$ is the length of $S(L,q,d)$.

Adopting dynamic programming and partial parse tree traversal on the derivation tree, we reduce the time complexity to $O(dk|M||L|)$, where $|L|$ is the sum of length of all production rules. If the link is not trivial, the size of $S(L,i,k)$ exponentially grows with d. Thus, in such a case, we can expect that $d|L|$ is significantly smaller than $|S|$.

Next, we design an efficient algorithm for computing $M(L,q,d,k)$. As we have seen in the previous section, there is a close relation between the computation of (Milnor) invariant and the construction of grammar compression, i.e., a set of longitudes deriving an invariant defines a CFG, and if the depth of the derivation is fixed d, the CFG is regarded as a grammar compression.

Let $T(L,q,d)$ be the corresponding derivation tree for $S(L,q,d)$. Traversing this tree in a depth-first order, we can compute $M(L,q,d,k)$ as follows. Let x be a current node and label(x) denote the label of x, i.e., label$(x) \in \{\ell_k, a_{i,j}, a_{i,j}^{-1}\}$ for some i,j,k. For any leaf x, the polynomial on x, denoted by $p(x)$, is $1 + X_i$ or $1 - X_i + X_i^2 + \cdots + (-1)^k X_i^k$ depending on label$(x) \in \{a_{i,j}, a_{i,j}^{-1}\}$. For an internal node x, if we have already computed $p(y)$ satisfying label$(y) =$ label(x) and depth$(y) =$ depth(x), we can skip the computation since $p(x) = p(y)$. Otherwise, if all $p(x_i)$ are determined for the children $x_1, \ldots, x_n$ of x, $p(x)$ is computed as their products, i.e., $p(x) = p(x_1) \cdots p(x_n)$. We note that, by the definition of Milnor invariants, if the expansion of $p(x_1) \cdots p(x_n)$ generates higher order terms, all such terms are discarded, and thus, any term in $p(x)$ is bounded by order k. By this strategy, we can obtain $M(L,q,d,k)$ as $p(r)$ for the root r.

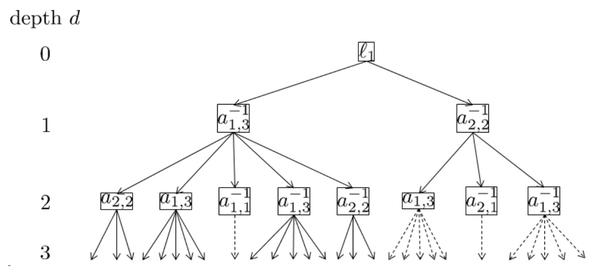


Fig. 5. Partial traverse of derivation tree defined by longitudes. The broken edges are imaginary because we can skip all of them.

This traverse is illustrated in Fig. 5. The complete description of the algorithm is given in Algorithm 1. The correctness of this algorithm is easily confirmed by the above outline. We show the time and space complexities.

Theorem 1. *The time complexity of Algorithm 1 is $O(kd|M||L|)$, and the space complexity of it is $O(d|L||M| + k|L|)$.*

Proof. Consider the derivation tree $T(L, q, d)$ deriving $S(L, q, d)$. Let $S(i)$ be the sequence of labels in $T(L, q, d)$ on depth $i = 0, 1, \ldots, d$, where $S(d) = S$. For each i, the number of different labels in $S(i)$ is at most $|L|$. Thus, using the lookup-table $\mathit{Table}[\mathit{label}][i]$, the number of nodes visited by the algorithm is bounded by $O(d|L|)$. Because the maximum size of a polynomial stored in $\mathit{Table}[\,][\,]$ is at most $|M|$, the time for production in a node is bounded by $O(k|M|)$. Therefore, the time complexity is $O(kd|M||L|)$. On the other hand, a polynomial of size at most $|M|$ is stored in $\mathit{Table}[\,][\,]$ with at most $k|L|$ entries, and the representation size of the initial polynomial $1 - X_i + X_i^2 + \cdots + (-1)^k X_i^k$ is $O(k)$ for each $i = 1, \ldots, |L|$. These derive the space complexity. $\square$

Compared to the time complexity $O(k|M||S|)$ of naive algorithm, $O(kd|M||L|)$ is expected to be significantly smaller because $d|L|$ is the size of CFG deriving the string $S(L, q, d)$. In the next section, we show the performance of our algorithm for real links.

4 Experiments

In this section, we show the experimental results for fast computation of Milnor invariants from given longitudes by the proposed algorithm.

We implement Algorithm 1 and examine its performance of computing Milnor invariants and compare our algorithm with the naive one. All experiments are made on a Linux (CentOS 6.2) machine with an 8-Core Intel(R) Xeon(R) CPU E7-8837 2.67GHz with 1TB memory.

For the experiments, we prepare three string links, denoted by Link1, Link2 and Link3 respectively. For each links, we show the results of the length $|S|$ of the

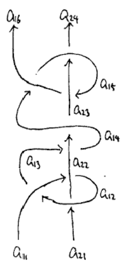
Algorithm 1 (Button-up Milnor invariants computation) Input is (L, q, d, k) where L is the longitudes regarded as the production rules, q is the index of *root* of derivation tree from ℓ_q, d is the maximum depth in the derivation tree, and k is the maximum order in the polynomial. Output is the Milnor invariants $M(L, q, d, k)$. $\mathit{Table}[\mathit{label}][i]$ is the lookup-table for referring whether the polynomial $p(\mathit{label})$ in depth i is already computed or not.

```
 1: $\mathit{Table}[*][*] \leftarrow$ nil;
 2: $\mathit{Table}[\text{label}(x)][d] \leftarrow 1 + X_i$ or $1 - X_i + X_i^2 + \cdots + (-1)^k X_i^k$ with leaf $x$;
 3: PARTIAL_TRAVERSE($\mathit{Table}[\,][\,], \mathit{root}, 0$);
 4: return $\mathit{Table}[\text{label}(\mathit{root})][0]$;
 5:
 6: procedure PARTIAL_TRAVERSE($\mathit{Table}[\,][\,], \mathit{root}, \mathit{depth}$)
 7:     $\mathit{current\_n} \leftarrow \mathit{root}$;
 8:     $\mathit{current\_d} \leftarrow \mathit{depth}$;
 9:     while $\mathit{Table}[\text{label}(\mathit{root})][\mathit{depth}] =$ nil do
10:         if $\mathit{Table}[\text{label}(x_i)][\mathit{current\_d} + 1] \neq$ nil for all the children $x_i$ then
11:             $\mathit{Table}[\text{label}(\mathit{current\_n})][\mathit{current\_d}] \leftarrow p(x_1) \cdots p(x_n)$;
12:             $\mathit{current\_d} \leftarrow \mathit{current\_d} - 1$;
13:             $\mathit{current\_n} \leftarrow \text{parent}(\mathit{current\_n})$;
14:         end if
15:         if $\mathit{Table}[\text{label}(x)][\mathit{current\_d} + 1] =$ nil for a child $x$ then
16:             PARTIAL_TRAVERSE($\mathit{Table}[\,][\,], x, \mathit{current\_d} + 1$);
17:         end if
18:     end while
19: end procedure
```

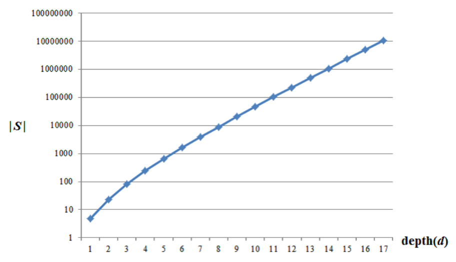
depth-d longitude, the time and memory consumption to compute the Milnor invariant by naive and proposed algorithms (See Fig. 6 - 8). In Fig. 6 - 8, each handwriting a_{ij} denotes the expression $a_{i,j}$ for the corresponding segment, and the name ℓ_i of string link is omitted in the all link images since the index i of ℓ_i is corresponding to the index i of $a_{i,1}$. In the legends of graphs on Fig. 6 - 8, "naive" stands for the naive algorithm, and "proposed" for our algorithm.

Although Link1 (Fig. 6) is constructed by only two strings, the length $|S|$ exponentially grows with the depth d. Thus, as shown in Fig. 6, the running time of the naive algorithm directly treating the expanded longitude is proportional to $|S|$. On the other hand, the time of proposed algorithm is significantly reduced thanks to our strategy of partial traverse. The result for Link2 and Link3(Fig. 7, 8) constructed by three strings also shows the scalability of the proposed algorithms.

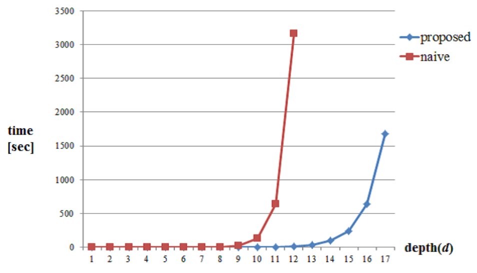
The memory consumptions of proposed algorithm is greater than that of naive algorithm. This matter is caused by our implementation for representation of polynomials; Two arrays are used for storing terms and coefficients. For example, $1+X+2X^2+X^4$ is represented by $("", X, XX, XXXX)$ and $(1, 1, 2, 1)$. We use 1 byte for each variable X and 4 byte for each coefficient. Thus we need huge memory for developing the lookup-table. On the other hand, a symbol in



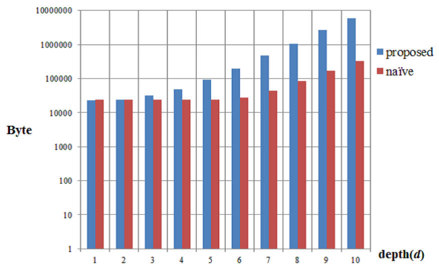
The link image of 2-string (Link1)



The longitude length $|S|$ for Link1 with depth d



Computation time for Link1



Memory consumption for Link1

Fig. 6. Experimental results in computing the Milnor invariants for Link1.

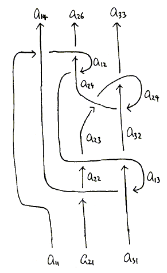
$S(L, q, d)$ requires just 1 byte in naive method. Improvement of this drawback is the most important future work.

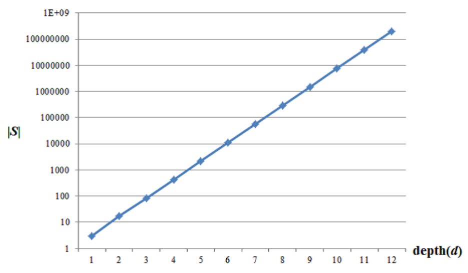
5 Conclusion

We propose an efficient and scalable algorithm for computing Milnor invariants of string links. Our algorithm improves the time and space complexities, and its performance is confirmed by experiments with non-trivial complicated string links. Our algorithm allows us to compute Milnor invariants for sufficiently large input string link that have not been computed previously due to its intractability. However, an improvement of memory consumption for real input is a challenge. Another challenge is to discover an interesting knowledge from the Milnor invariants computed by our algorithm so that it gives a clue to solve an open problem, or makes some progress in string link analysis.

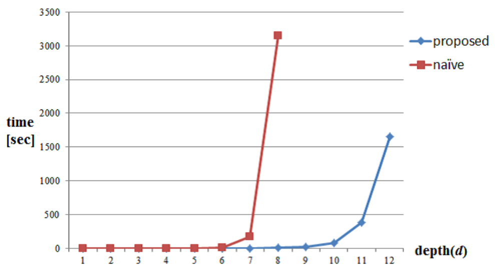
References

1. A. Apostolico and S. Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000.
2. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inform. Theory*, 51(7):2554–2576, 2005.

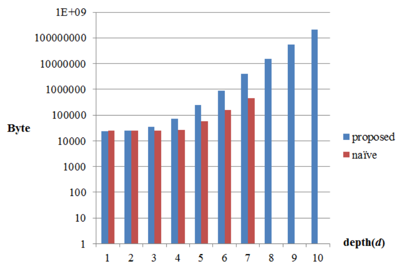

The link image of 3-string (Link2)



The longitude length $|S|$ for Link2 with depth d



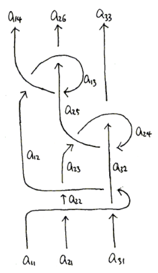
Computation time for Link2



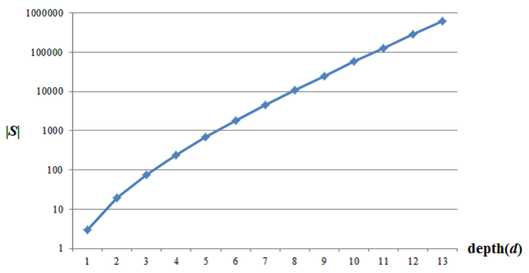
Memory consumption for Link2

Fig. 7. Experimental results in computing the Milnor invariants for Link2.

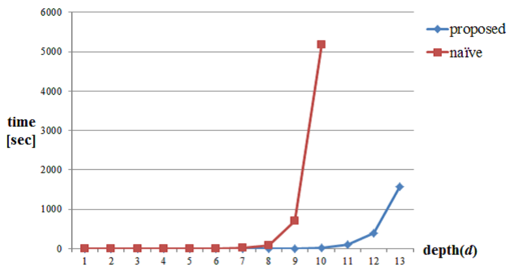
3. F. Claude and G. Navarro. Self-indexed grammar-based compression. *Fundam. Inform.*, 111(3):313–337, 2011.
4. F. Claude and G. Navarro. Improved grammar-based compressed indexes. In *SPIRE2012*, pages 180–192, 2012.
5. K. Goto, H. Bannai, S. Inenaga, and M. Takeda. Fast q-Gram Mining on SLP Compressed Strings. *J. Discrete Algorithms*, 18:89–99, 2013.
6. N. Habegger and X.S. Lin. The classification of links up to link-homotopy. *Journal of the American Mathematical Society*, 3(2):389–419, 1990.
7. S. Inenaga and H. Bannai. Finding characteristic substrings from compressed texts. In *PSC*, pages 40–54, 2009.
8. M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic J. Comp.*, 4(2):172–186, 1997.
9. J.C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inform. Theory*, 46:737–754, 2000.
10. N.J. Larsson and A. Moffat. Offline dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.
11. E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *SODA*, pages 205–212, 2002.
12. S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-Index: A Compressed Index Based on Edit-Sensitive Parsing. *Journal of Discrete Algorithms*, 18:100–112, 2013.
13. S. Maruyama, H. Sakamoto, and M. Takeda. An online algorithm for lightweight grammar-based compression. *Algorithms*, 5(2):213–235, 2012.
14. J. Milnor. Link groups. *The Annals of Mathematics*, 59(2):177–195, 1954.



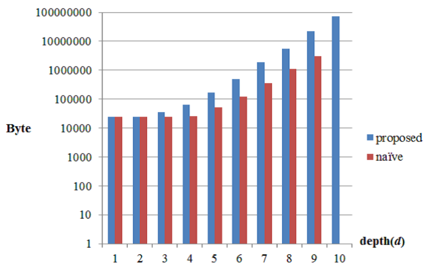
The link image of 3-string (Link3)



The longitude length $|S|$
for Link3 with depth d



Computation time for Link3



Memory consumption for Link3

Fig. 8. Experimental results in computing the Milnor invariants for Link3.

15. J. Milnor. Isotopy of links. In *Algebraic geometry and topology. A symposium in honor of S. Lefschetz*, pages 280–306. Princeton University Press, Princeton, NJ, 1957.
16. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.
17. Y. Tabei, T. Takabatake, and H. Sakamoto. A succinct grammar compression. In *CPM*, 2013. to appear.
18. Y. Takabatake, Y. Tabei, and H. Sakamoto. Variable-length codes for space-efficient grammar-based compression. In *SPIRE*, pages 398–410, 2012.
19. A. Tiskin. Towards approximate matching in compressed strings. In *CSR*, pages 401–414, Berlin, Heidelberg, 2011. Springer-Verlag.
20. T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *CPM*, volume 6661, pages 309–322, 2011.