Subcaterpillar Isomorphism Between Caterpillars: Subtree Isomorphism Restricted Text and Pattern Trees to Caterpillars

Tomoya Miyazaki and Kouich Hirata
Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan

Keywords: Subcaterpillar Isomorphism Between Caterpillars, Subcaterpillar Isomorphism, Subtree Isomorphism. Rooted Labeled Caterpillar, Rooted Labeled Unordered Tree, Caterpillar Inclusion.

Abstract: In this paper, as a pattern matching for *rooted labeled caterpillars* (*caterpillars*, for short), we discuss a *subcaterpillar isomorphism between caterpillars* whether or not a pattern caterpillar is a subcaterpillar of a text caterpillar. Then, we design the algorithms to solve it by simplifying the algorithms for subcaterpillar isomorphism (between a caterpillar and a tree) when a pattern caterpillar is a subcaterpillar of a text tree designed by Miyazaki and Hirata (2022). These algorithms run in $O(hH\sigma)$ time and $O(h)$ space, where h is the height of a pattern caterpillar, H is the height of a text caterpillar and σ is the number of labels in the caterpillars. Finally, we give experimental results of computing these algorithms by comparing with subcaterpillar isomorphism and caterpillar inclusion.

1 INTRODUCTION

The pattern matching for tree-structured data such as HTML and XML documents for web mining or DNA and glycan data for bioinformatics is one of the fundamental tasks for information retrieval or query processing. As such pattern matching for *rooted labeled unordered trees* (a *tree*, for short), a *subtree isomorphism* is the problem of determining, for a *pattern tree* P and a *text tree* T, whether or not there exists a subtree of T which is isomorphic to P. It is known that the subtree isomorphism can be solved in $O(p^{1.5}t/\log p)$ time (Shamir and Tsur, 1999), where p is the number of vertices in P and t is the number of vertices in T. On the other hand, it cannot be solved in $O(t^{2-\varepsilon})$ time for every ε $(0 < \varepsilon < 1)$ under SETH (Abboud et al., 2018).

Recently, by focusing on a *rooted labeled caterpillar* (a *caterpillar*, for short) (*cf.*, (Gallian, 2007)) as the restriction of trees, Miyazaki and Hirata have discussed the *subcaterpillar isomorphism* when a pattern tree is a caterpillar (Miyazaki and Hirata, 2022). Then, they have designed the algorithms of the subcaterpillar isomorphism running in (i) $O(tDh\sigma)$ time and $O(Dh)$ space and (ii) $O(tD\sigma)$ time and $O(D(h+H))$ space, respectively[1]. Here, h is the height of P, H is the height of T, D is the degree of T and σ is the number of alphabets for labels in a pattern and a text. Note that these algorithms return all of the positions in T where P is a subcaterpillar of T.

As another pattern matching for tree-structured data, it is known the inclusion problem of determining whether or not a text tree T achieves to a pattern tree P by deleting vertices in T is NP-complete (Kilpeläinen and Mannila, 1995). This statement also holds even if P is a caterpillar (Kilpeläinen and Mannila, 1995). On the other hand, Miyazaki *et al.* (Miyazaki et al., 2022) have shown that, if both P and T are caterpillars, then we can solve the inclusion problem in $O((h+H)\sigma)$ time. We call this problem a *caterpillar inclusion*. Note that this algorithm returns "yes" if a patter caterpillar P is included in a text caterpillar T and "no" otherwise.

In this paper, we investigate a *subcaterpillar isomorphism between caterpillars* that is a subcaterpillar isomorphism when both a pattern tree P and a text tree T are caterpillars. The subcaterpillar isomorphism between caterpillars is the special problem of not only the subcaterpillar isomorphism but also the caterpillar inclusion, because it is regarded as a caterpillar inclusion that T achieves P by deleting leaves or the roots in T.

In this paper, by simplifying the algorithms (i) and (ii) for subcaterpillar isomorphism, we design two algorithms CatCatIso and CatCatIso2 for subcaterpillar isomorphism between caterpillars. Then, both CatCatIso and CatCatIso2 run in $O(hH\sigma)$

[1] In this paper, we ignore the time complexity of the initialization of storing structures by traversing data, as same as (Miyazaki et al., 2022).

Miyazaki, T. and Hirata, K.
Subcaterpillar Isomorphism Between Caterpillars: Subtree Isomorphism Restricted Text and Pattern Trees to Caterpillars.
DOI: 10.5220/0011659600003411
In *Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2023)*, pages 89-94
ISBN: 978-989-758-626-2; ISSN: 2184-4313

time and $O(h)$ space. Furthermore, we give experimental results of computing CATCATISO and CATCATISO2, by comparing with subcaterpillar isomorphism between a caterpillar and a tree (Miyazaki and Hirata, 2022) and caterpillar inclusion (Miyazaki et al., 2022).

2 PRELIMINARIES

A *tree* is a connected graph without cycles. For a tree $T = (V,E)$, we denote V and E by $V(T)$ and $E(T)$. We sometimes denote $v \in V(T)$ by $v \in T$. A *rooted tree* is a tree with one vertex r chosen as its *root*, which we denote by $r(T)$.

For each vertex v in a rooted tree with the root r, let $UP_r(v)$ be the unique path from v to r. The *parent* of $v(\neq r)$, which we denote by $par(v)$, is its adjacent vertex on $UP_r(v)$ and the *ancestors* of $v(\neq r)$ are the vertices on $UP_r(v) \setminus \{v\}$. We denote $u < v$ if v is an ancestor of u, and we denote $u \leq v$ if either $u < v$ or $u = v$. The parent and the ancestors of the root r are undefined. We say that u is a *child* of v if v is the parent of u, and u is a *descendant* of v if v is an ancestor of u. We denote the set of all children of v by $ch(v)$. Two vertices with the same parent are called *siblings*. A *leaf* is a vertex having no children and we denote the set of all the leaves in T by $lv(T)$. We call a vertex that is not a leaf an *internal vertex*.

For a rooted tree $T = (V,E)$ and a vertex $v \in T$, the *complete subtree of T at v*, denoted by $T(v)$, is a rooted tree $S = (V',E')$ such that $r(S) = v$, $V' = \{w \in V \mid w \leq v\}$ and $E' = \{(u,w) \in E \mid u,w \in V'\}$.

The *height* $h(v)$ of a vertex v is defined as $|UP_r(v)| - 1$ and the *height* $h(T)$ of T is the maximum height for every vertex $v \in T$. The *degree* $d(v)$ of a vertex v is the number of the children of v, and the *degree* $d(T)$ of T is the maximum degree for every vertex in T.

We say that a rooted tree is *ordered* if a left-to-right order among siblings is given; *Unordered* otherwise. For a fixed finite alphabet Σ, we say that a tree is *labeled* over Σ if each vertex is assigned a symbol from Σ. We denote the label of a vertex v by $l(v)$, and sometimes identify v with $l(v)$. In this paper, we call a rooted labeled unordered tree over Σ a *tree*, simply.

Definition 1. Let T and S be trees.

1. We say that T is a *subtree* of S, denoted by $T \preceq S$, if T is a tree such that $V(T) \subseteq V(S)$ and $E(T) = \{(v,w) \in E(S) \mid v,w \in V(T)\}$.
2. We say that T and S are *isomorphic*, denoted by $T \simeq S$, if $T \preceq S$ and $S \preceq T$.
3. We say that T is a *subtree isomorphism* of S, denoted by $T \trianglelefteq S$, if there exists a tree $S' \preceq S$ such that $T \simeq S'$.

In this paper, we deal with a *subtree isomorphism problem* of P for T whether or not $P \trianglelefteq T$ for trees P and T. We call P a *pattern tree* and T a *text tree*. Then, the following theorem holds.

Theorem 1. (Shamir and Tsur, 1999) *Let P and T be trees where $p = |P|$ and $t = |T|$. Then, the problem of determining whether or not $P \trianglelefteq T$ is solvable in $O(p^{1.5}t/\log p)$ time.*

As the restricted form of trees, we introduce a *rooted labeled caterpillar* (a *caterpillar*, for short) as follows.

Definition 2. We say that a tree is a *caterpillar* (*cf.* (Gallian, 2007)) if it is transformed to a rooted path after removing all the leaves in it. For a caterpillar C, we call the remained rooted path a *backbone* of C and denote it by $bb(C)$.

It is obvious that $r(C) = r(bb(C))$ and $V(C) = V(bb(C)) \cup lv(C)$ for a caterpillar C, that is, every vertex in a caterpillar is either a leaf or an element of the backbone.

We call a subtree isomorphism when P is a caterpillar, that is, the problem of determining whether or not $P \trianglelefteq T$, a *subcaterpillar isomorphism*. Then, the following theorem holds.

Theorem 2. (Miyazaki and Hirata, 2022) *Let P be a caterpillar and T a tree, where $t = |T|$, $h = h(P)$, $H = h(T)$, $D = d(T)$ and $\sigma = |\Sigma|$. Then, the problem of determining whether or not $P \trianglelefteq T$ is solvable (i) in $O(tDh\sigma)$ time and $O(Dh)$ space and (ii) in $O(tD\sigma)$ time and $O(D(h+H))$ space.*

We refer the algorithm of (i) (*resp.*, (ii)) in Theorem 2 to CATTREEISO (*resp.*, CATTREEISO2). Note that both CATTREEISO and CATTREEISO2 return all of the positions in T where P is a subcaterpillar of T.

Finally, we introduce a *tree inclusion* and a *caterpillar inclusion*. For a tree T and a vertex $v \in T$, the *deletion* of v in T is to delete a non-root vertex v in T with a parent v', making the children of v become the children of v' that are inserted in the place of v as a subset of the children of v'. We denote the result of the deletion of v in T by $delete(T,v)$. See Figure 1.

Definition 3. Let P and T be trees. Then, we say that P is an *inclusion* of T, denoted by $P \sqsubseteq T$, if either $P \simeq T$ or there exists a sequence of vertices $v_1,\ldots,v_k$ in T such that $T_0 \simeq T$, $T_k \simeq P$ and $T_{i+1} \simeq delete(T_i, v_{i+1})$ $(0 \leq i \leq k-1)$.

For trees P and T, if $P \trianglelefteq T$ then $P \sqsubseteq T$, because T achieves P by deleting leaves or roots in T. On the other hand, the converse does not hold in general.

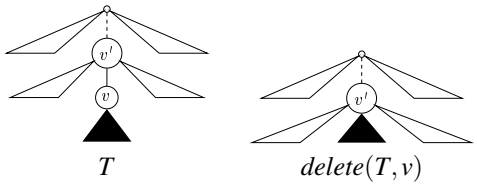

Figure 1: $delete(T,v)$.

We call the tree inclusion when both P and T are caterpillars a *caterpillar inclusion*. Then, the following theorems hold.

Theorem 3. (Kilpeläinen and Mannila, 1995) *For trees P and T, the problem of determining whether or not $P \sqsubseteq T$ is NP-complete. This statement also holds even if the maximum height of T is at most 3.*

Theorem 4. (Miyazaki et al., 2022) *Let P and T be caterpillars, where $h = h(P)$, $H = h(T)$ and $\sigma = |\Sigma|$. Then, the problem of determining whether or not $P \sqsubseteq T$ is solvable in $O((h+H)\sigma)$ time.*

We refer the algorithm in Theorem 4 to CATCATINC. Note that CATCATINC returns "yes" if $P \sqsubseteq T$ and "no" otherwise.

3 SUBCATERPILLAR ISOMORPHISM BETWEEN CATERPILLARS

In this paper, we focus on a *subcaterpillar isomorphism between caterpillars* that is a subcaterpillar isomorphism when both P and T are caterpillars. In other words, we focus on the problem of whether or not $P \trianglelefteq T$ for caterpillars P and T. We call P and T a *pattern caterpillar* and a *text caterpillar*, respectively. Throughout of this section, we refer $p = |P|$, $t = |T|$, $h = h(P)$, $H = h(T)$, $D = d(T)$ and $\sigma = |\Sigma|$.

For a pattern caterpillar P, we refer the backbone of P to a sequence $\langle v_1, \ldots, v_n \rangle$ such that $(v_i, v_{i+1}) \in E(P)$ and $v_n = r(P)$. We denote the children of v_i by $ch(v_i)$. For a text caterpillar T, we refer the backbone of T to a sequence $\langle w_1, \ldots, w_m \rangle$ such that $(w_j, w_{j+1}) \in E(T)$ and $w_m = r(T)$. We denote the children of w_j by $ch(w_j)$.

Suppose that $P \trianglelefteq T$ and let $P' \preceq T$ be a subcaterpillar in T such that $P \simeq P'$ and $bb(P') = \langle v'_1, \ldots, v'_n \rangle$, where $v'_n = r(P')$. Then, we call the index j such that $v'_1 = w_j$ in T a *matching position* of P in T.

As same as the algorithms of CATTREEISO and CATTREEISO2, we use a *multiset* of labels in order to compare two sets of vertices. A *multiset* on Σ is a mapping $S : \Sigma \to \mathbf{N}$. For two multisets S_1 and S_2, $S_1 \subseteq S_2$ if $S_1(a) \leq S_2(a)$ for every $a \in \Sigma$.

For a set V of vertices, we denote the multiset of labels occurring in V by $\widetilde{V}$. Then, it is necessary for the subcaterpillar isomorphism to check whether or not $\widetilde{ch(v_i)} \subseteq \widetilde{ch(w_j)}$ for $v_i \in bb(P)$ and $w_j \in bb(T)$. It is realized to check $\left(\widetilde{ch(v_i)}\right)(a) \leq \left(\widetilde{ch(w_j)}\right)(a)$ for every $a \in \Sigma$ in $O(\sigma)$ time (*cf.* (Muraka et al., 2019)).

By simplifying the algorithm CATTREEISO in Theorem 2 (i), we design the algorithm CATCATISO in Algorithm 1 to determine whether or not $P \trianglelefteq T$ and to output all of the matching positions if $P \trianglelefteq T$. Here, the table $match(i)$ stores j such that $v_i \in bb(P)$ is corresponding to $w_j \in bb(T)$.

```
   procedure CATCATISO(P,T)
       /* P : caterpillar, bb(P) = ⟨v_1,...,v_n⟩ */
       /* T : caterpillar, bb(T) = ⟨w_1,...,w_m⟩ */
1      for i = 1 to n − 1 do match(i) ← 0;
2      for j = 1 to m do
3          for i = n − 1 downto 1 do
4              if match(i) ≠ 0 then
5                  k ← match(i); match(i) ← 0;
6                  if l(v_{i+1}) = l(w_j) and
                   ch(v_{i+1}) ⊆ ch(w_j) then
7                      if i + 1 = n then output k;
8                      else match(i + 1) ← k;
9          if l(v_1) = l(w_j) and ch(v_1) ⊆ ch(w_j) then
10             match(1) ← j;
```

Algorithm 1: CATCATISO.

Example 1. Consider the pattern caterpillar P and the text caterpillar T in Figure 2. Here, $bb(P) = \langle v_1, v_2, v_3 \rangle$ and $bb(T) = \langle w_1, w_2, w_3, w_4, w_5, w_6 \rangle$, so it holds that $n = 3$ and $m = 6$.

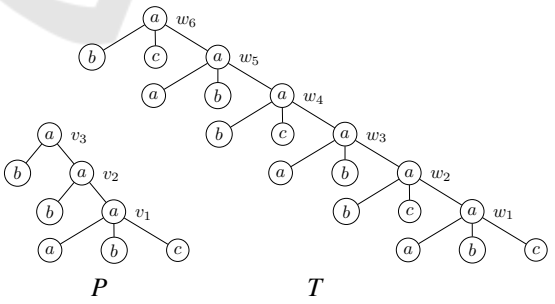

Figure 2: A pattern caterpillar P and a text caterpillar T in Example 1.

For P and T, the algorithm CATCATISO(P,T) stores the values of $match(i)$ and outputs the matching positions as Table 1 Then, the matching positions of P in T are 1, 2 and 4.

On the other hand, by simplifying the algorithm CATTREEISO2 in Theorem 2 (ii), we design another

Table 1: The execution of the algorithm CATCATISO(P,T).

j	1	2	3	4	5	6
$match(1)$	1	2	0	4	0	6
$match(2)$	0	1	2	0	4	0
output			1	2		4

algorithm CATCATISO2 in Algorithm 2. The difference between CATCATISO2 and CATCATISO is that CATCATISO2 does not always access all the values of $match(i)$ for $1 \leq i \leq n-1$ but just access the values of $match(i)$ such that $i \in CHK$.

```
    procedure CATCATISO2(P,T)
        /* P : caterpillar, bb(P) = [v_1,...,v_n] */
        /* T : caterpillar, bb(T) = [w_1,...,w_m] */
1       CHK ← ∅;
2       for i = 1 to n−1 do match(i) ← 0;
3       for j = 1 to m do
4           foreach i ∈ CHK do
5               k ← match(i); match(i) ← 0;
                CHK ← CHK \ {i};
6               if l(v_{i+1}) = l(w_j) and
                ch~(v_{i+1}) ⊆ ch~(w_j) then
7                   if i+1 = n then output k;
8                   else match(i+1) ← k;
                    CHK ← CHK ∪ {i+1};
9           if l(v_1) = l(w_j) and ch~(v_1) ⊆ ch~(w_j) then
10              match(1) ← j; CHK ← CHK ∪ {1};
```

Algorithm 2: CATCATISO2.

Example 2. Consider the pattern caterpillar P and the text caterpillar T in Example 1 (Figure 2). Then, the algorithm CATCATISO2(P,T) stores the values of $match(i)$ and outputs the matching positions, and additionally updates the set of CHK as Table 2. Hence, the matching positions of P in T are 1, 2 and 4.

Table 2: The execution of the algorithm CATCATISO2(P,T).

j	1	2	3	4	5	6
$match(1)$	1	2	0	4	0	6
$match(2)$	0	1	2	0	4	0
output			1	2		4
CHK	1	1,2	2	1	2	1

For subcaterpillar isomorphism between caterpillars, the following theorem holds.

Theorem 5. *Let P and T be caterpillars, where $h = h(P)$, $H = h(T)$ and $\sigma = |\Sigma|$. Then, the algorithms of* CATCATISO(P,T) *and* CATCATISO2(P,T) *output all the matching positions of P in T correctly in $O(hH\sigma)$ time and $O(h)$ space.*

Proof. The following proof of the correctness is similar as (Miyazaki and Hirata, 2022).

The algorithm CATCATISO first stores the candidate j of the matching point corresponding to v_1 to $match(1)$ if $l(v_1) = l(w_j)$ and $\widetilde{ch(v_1)} \subseteq \widetilde{ch(w_j)}$ (line 9). Then, for the current j, the algorithm CATCATISO removes the candidate k from $match(i)$ and stores k to $match(i+1)$ if $l(v_{i+1}) = l(w_j)$, $\widetilde{ch(v_{i+1})} \subseteq \widetilde{ch(w_j)}$ and $i+1 < n$ (lines 6 and 8). If $i+1 = n$, then the algorithm CATCATISO outputs k (line 7).

Hence, every output k at line 7 satisfies that $l(v_i) = l(par^{i-1}(w_k))$ and $\widetilde{ch(v_i)} = \widetilde{ch(par^{i-1}(w_k))}$ for every i $(1 \leq i \leq n)$, where $par^0(v) = v$ and $par^{i+1}(v) = par(par^i(v))$. As a result, the algorithm SUBCATISO outputs all of the matching points of P in T.

On the other hand, the difference between the algorithms CATCATISO and CATCATISO2 is the usages of the set CHK, which is stored to all the indices i such that $match(i) \neq 0$ for $1 \leq i \leq n-1$. Hence, the algorithm CATCATISO2 can access all the values of $match(i)$ such that $match(i) \neq 0$ for every $1 \leq j \leq m$, which implies the correctness of the algorithm CATCATISO2.

Next, consider the computational complexity of the algorithms.

For the algorithm CATCATISO, we can check the lines 6 and 9 in $O(\sigma)$ time. Also, the for-loop in line 3 is repeated at $h-1$ times and the for-loop in line 2 is repeated at H times. Then, the total running time of CATCATISO is $O(hH\sigma)$ time. Also the space is the size of the table $match$, which is $O(h)$.

On the other hand, for the algorithm CATCATISO2, the foreach-loop in line 3 is repeated at most $h-1$ times. Then, the total running time of CATCATISO2 is $O(hH\sigma)$ time. Also the space is the sizes of the table $match$ and the set CHK, which is $O(h+h) = O(h)$. □

4 EXPERIMENTAL RESULTS

In this section, we give the experimental results of computing CATCATISO and CATCATISO2. Here, the computer environment is that OS is Ubuntu 18.04.4, CPU is Intel Xeon E5-1650 v3(3.50GHz) and RAM is 3.8GB.

We deal with caterpillars for N-glycans and all-

glycans from KEGG[2], CSLOGS[3], the largest 51,395 caterpillars (1%) in dblp[4] (refer to $\text{dblp}_{1\%}$) and SwissProt from UW XML Repository[5]. Also we deal with non-isomorphic caterpillars obtained by deleting the root in Nasa (refer to $\text{NASA}_{\circ}^{-}$), Protein (refer to $\text{Protein}_{\circ}^{-}$) and University (refer to $\text{University}_{\circ}^{-}$) from UW XML Repository. Table 3 illustrates the information of such caterpillars. Here, #, n, d, h, λ and β are the number of caterpillars, the average number of vertices, the average degree, the average height, the average number of leaves and the average number of labels.

Table 3: The information of caterpillars.

data	#	n	d	h	λ	β
N-glycans	513	6.40	1.84	4.22	2.19	3.24
all-glycans	7,984	4.74	1.49	3.02	1.72	2.84
CSLOGS	41,592	5.84	3.05	2.20	3.64	5.18
$\text{dblp}_{1\%}$	51,395	21.29	20.21	1.04	20.25	9.73
SwissProt	6,804	35.10	24.96	2.00	33.10	16.79
$\text{Nasa}_{\circ}^{-}$	33	7.27	5.15	1.64	5.64	3.18
$\text{Protein}_{\circ}^{-}$	5,150	4,97	3.63	1.16	3.81	4.57
$\text{University}_{\circ}^{-}$	26	1.35	0.35	0.19	1.15	1.35

We compare all the pairs (P,T) in the caterpillars in Table 3. The number of pairs is $\# \times (\# - 1)$, and Table 4 summarizes such number as #pairs.

Table 4: The number (#pairs) of all the pairs in caterpillars in Table 3.

data	#pairs	data	#pairs
N-glycans	262,656	SwissProt	46,287,612
all-glycans	63,736,272	$\text{Nasa}_{\circ}^{-}$	1,056
CSLOGS	1,729,852,872	$\text{Protein}_{\circ}^{-}$	26,517,350
$\text{dblp}_{1\%}$	2,641,394,630	$\text{University}_{\circ}^{-}$	650

First, we compare the running time of the algorithms CATCATISO and CATCATISO2 in Section 3 with the algorithms of CATTREEISO and CATTREEISO2 (Miyazaki and Hirata, 2022).

Then, Table 5 illustrates the total and average running time of computing $P \trianglelefteq T$ for all the data by CATCATISO and CATCATISO2. Here, the bold faces present the smaller total running time.

Table 5 shows that, whereas CATCATISO is faster than CATCATISO2 for N-glycans, $\text{dblp}_{1\%}$, $\text{Nasa}_{\circ}^{-}$ and $\text{University}_{\circ}^{-}$, CATCATISO2 is faster than CATCATISO for all-glycans, CSLOGS and SwissProt. On the other hand, since the difference of the computation time between CATCATISO and CATCATISO2 is not large, the usage of the set *CHK* in CATCATISO2 is not effective for our experimental data.

Table 5: The total and average running time (msec.) of computing $P \trianglelefteq T$ for all the data by CATCATISO and CATCATISO2.

data	CATCATISO total	CATCATISO ave.	CATCATISO2 total	CATCATISO2 ave.
N-glycans	**4,719**	0.02	4,753	0.02
all-glycans	617,932	0.01	**616,863**	0.01
CSLOGS	13,703,530	0.01	**13,636,908**	0.01
$\text{dblp}_{1\%}$	**73,453,734**	0.03	73,637,350	0.01
SwissProt	3,706,628	0.08	**3,696,157**	0.08
$\text{Nasa}_{\circ}^{-}$	**12**	0.01	17	0.02
$\text{Protein}_{\circ}^{-}$	**166,717**	0.01	170,123	0.01
$\text{University}_{\circ}^{-}$	**1**	0.00	4	0.01

Table 6 illustrates the total and average running time of computing $P \trianglelefteq T$ for all the data by CATTREEISO and CATTREEISO2.

Table 6: The total and average running time (msec.) of computing $P \trianglelefteq T$ for all the data by CATTREEISO and CATTREEISO2.

data	CATTREEISO total	CATTREEISO ave.	CATTREEISO2 total	CATTREEISO2 ave.
N-glycans	6,315	0.02	6,325	0.02
all-glycans	726,173	0.01	855,131	0.01
CSLOGS	15,444,331	0.01	19,242,569	0.01
$\text{dblp}_{1\%}$	78,063,181	0.03	80,077,842	0.01
SwissProt	3,790,730	0.08	3,934,301	0.09
$\text{Nasa}_{\circ}^{-}$	17	0.02	14	0.01
$\text{Protein}_{\circ}^{-}$	174,212	0.01	213,539	0.01
$\text{University}_{\circ}^{-}$	4	0.00	4	0.01

Tables 5 and 6 show that the algorithms of CATCATISO and CATCATISO2 are faster than the algorithms of CATTREEISO and CATTREEISO2. The reason is that, whereas CATTREEISO and CATTREEISO2 are necessary to traverse a whole text caterpillar, CATCATISO and CATCATISO2 just traverse the backbone of a text caterpillar.

Next, we compare the algorithms CATCATISO and CATCATISO2 with CATCATINC. Note that CATINC is a decision algorithm to return just "yes" or "no." Then, we use the decision versions of the algorithms of CATCATISO and CATCATISO2, designed by changing line 7 as follows and by adding the following line 11 to the last of the algorithms.

line 7 **if** $i+1=n$ **then output** "yes"; **halt**;
line 11 **output** "no";

We refer the decision versions of CATCATISO and CATCATISO2 to CATCATISO* and CATCATISO2*.

[2]Kyoto Encyclopedia of Genes and Genomes, http://www.kegg.jp/

[3]http://www.cs.rpi.edu/~zaki/www-new/pmwiki.php/Software/Software

[4]http://dblp.uni-trier.de/

[5]http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/www/repository.html

Then, Table 7 illustrates the total and average running time (msec.) of determining whether or not $P \trianglelefteq T$ by CATCATISO* and CATCATISO2* and of determining whether or not $P \sqsubseteq T$ by CATCATINC.

Table 7: The total and average running time (msec.) of determining whether or not $P \trianglelefteq T$ by CATCATISO* and CATCATISO2* and of determining whether or not $P \sqsubseteq T$ by CATCATINC.

	CATCATISO*/ISO2*		CATCATINC	
data	total	ave.	total	ave.
N-glycans	4,788/4,721	0.02	16,075	0.06
all-glycans	611,103/603,052	0.01	2,221,026	0.04
CSLOGS	13,211,929/13,497,908	0.01	83,129,368	0.05
$\text{dblp}_{1\%}$	73,315,987/73,548,291	0.03	143,584,440	0.05
SwissProt	3,706,628/3,696,157	0.08	7,291.159	0.16
$\text{Nasa}^{-}_{\circ}$	11/11	0.01	29	0.03
$\text{Protein}^{-}_{\circ}$	164,107/164,240	0.01	596,332	0.02
$\text{University}^{-}_{\circ}$	1/1	0.00	9	0.01

As stated in the previous sections, the algorithm CATCATINC runs in $O((h+H)\sigma)$ time (Theorem 4) and the algorithms CATCATISO* and CATCATISO2* run in $O(hH\sigma)$ time (Theorem 5). On the other hand, Table 7 shows that the algorithms CATCATISO* and CATCATISO2* are much faster than the algorithm CATCATINC. One of the reasons is that, whereas the main loop in the algorithm CATCATINC is repeated at near to $h+H$ times, the for-loop in the algorithms CATCATISO* and CATCATISO2* are repeated at much smaller than H times.

Furthermore, Table 8 illustrates the number (#pairs) of pairs (P,T) such that $P \trianglelefteq T$ and $P \sqsubseteq T$ (Miyazaki et al., 2022) with its ratio (%) in all the pairs.

Table 8: The number (#pairs) of pairs (P,T) such that $P \trianglelefteq T$ and $P \sqsubseteq T$ with its ratio (%) in all the pairs.

	$P \trianglelefteq T$		$P \sqsubseteq T$	
data	#pairs	%	#pairs	%
N-glycans	17,505	6.67	21,919	8.35
all-glycans	646,170	1.01	907,776	1.42
CSLOGS	1,979,560	0.11	2,277,568	0.13
$\text{dblp}_{1\%}$	364,182,693	13.79	364,184,642	13.79
SwissProt	1,400,455	3.03	1,400,455	3.03
$\text{Nasa}^{-}_{\circ}$	108	10.23	108	10.23
$\text{Protein}^{-}_{\circ}$	3,701	0.01	3,701	0.01
$\text{University}^{-}_{\circ}$	1	0.15	1	0.15

Table 8 shows that, whereas #pairs such that $P \trianglelefteq T$ is smaller than #pair such that $P \sqsubseteq T$ for N-glycans, all-glycans, CSLOGS and $\text{dblp}_{1\%}$, the former is equal to the latter for SwissProt, $\text{Nasa}^{-}_{\circ}$, $\text{Protein}^{-}_{\circ}$ and $\text{University}^{-}_{\circ}$; Nevertheless, for these data, we can determine $P \trianglelefteq T$ faster than $P \sqsubseteq T$ shown in Table 7.

5 CONCLUSION

In this paper, we have designed the algorithms of CATCATISO and CATCATISO2 to solve the subcaterpillar isomorphism between caterpillars and given the experimental results of comparing them with the subcaterpillar isomorphism algorithms of CATTREEISO and CATTREEISO2 and the caterpillar inclusion algorithm CATCATINC.

Then, the algorithms of CATCATISO and CATCATISO2 are faster than the algorithms of CATTREEISO and CATTREEISO2 for subcaterpillar isomorphism between caterpillars. Also, whereas the algorithm CATCATINC is faster than the decision versions CATCATISO* and CATCATISO2* in theoretical, the latter is faster than the former in experimental.

Since Theorem 1 for the subtree isomorphism also holds for *unrooted* trees, it is a future work to extend the algorithms in this paper to *unrooted* subcaterpillar isomorphism between caterpillars. In particular, it is necessary to investigate whether or not the unrooted subcaterpillar isomorphism between caterpillars can avoid to the SETH-hardness of subtree isomorphism (Abboud et al., 2018).

REFERENCES

Abboud, A., Backurs, A., Hansen, T. D., v. Williams, V., and Zamir, O. (2018). Subtree isomorphism revisited. *ACM Trans. Algo.*, 14:27.

Gallian, J. A. (2007). A dynamic survey of graph labeling. *Electorn. J. Combin.*, 14:DS6.

Kilpeläinen, P. and Mannila, H. (1995). Ordered and unordered tree inclusion. *SIAM J. Comput.*, 24:340–356.

Miyazaki, T., Hagihara, M., and Hirata, K. (2022). Caterpillar inclusion: Inclusion problem for rooted labeled caterpillars. In *Proc. ICPRAM '22*, pages 280–287.

Miyazaki, T. and Hirata, K. (2022). Subcaterpillar isomorphism: Subtree isomorphism restricted pattern trees to caterpillars. In *Proc. FedCSIS '22*, pages 351–356.

Muraka, K., Yoshino, T., and Hirata, K. (2019). Vertical and horizontal distances to approximate edit distance for rooted labeled caterpillars. In *Proc. ICPRAM'19*, pages 590–597.

Shamir, R. and Tsur, D. (1999). Faster subtree isomorphism. *Algorithmica*, 33:267–280.