Similar Subtree Search Using Extended Tree Inclusion

(Extended abstract)

Tomoya Mori[*], Atsuhiro Takasu[†], Jesper Jansson[‡], Jaewook Hwang[*], Takeyuki Tamura[*], and Tatsuya Akutsu[*]
[*]Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan 611-0011
Email: {tmori,jj,hwangjw,tamura,takutsu@kuicr.kyoto-u.ac.jp
[†]National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430
Email: takasu@nii.ac.jp
[‡]The Hakubi Project, Kyoto University, Gokasho, Uji, Kyoto, Japan 611-0011
Email: jj@kuicr.kyoto-u.ac.jp

I. Introduction

The comparison of tree-structured data has numerous applications in various scientific areas such as RNA secondary structures and glycans in biology and natural language processing. The similar subtree search problem studied in this paper is: given a pattern tree (representing a query) and a large text tree (e.g., taken from a database), find the roots of all subtrees of the text tree that are similar to the pattern tree. We use *tree edit distance* for tree similarity. For *ordered* trees, there exists a polynomial-time algorithm to compute it [2]. However, in many situations, it is more appropriate to consider *unordered* trees. There can be a significant gap between the ordered and unordered tree edit distances, so similar trees may be missed if we simply fix some arbitrary ordering and apply the ordered variant. Unfortunately, the unordered tree edit distance problem is known to be NP-hard [3]. To cope with this hardness, several approaches have been taken. Kilpeläinen and Mannila [4] considered the *tree inclusion problem* which takes as input a pattern tree and a text tree, and asks whether the text tree can be obtained from the pattern tree by applying insertion operations. They showed that the problem is NP-hard in general for unordered trees but can be solved in polynomial time when the maximum outdegree of the pattern tree is upper-bounded by a constant. On the negative side, their approach ignores the costs of insertion operations, which in many applications prohibits it from being practically useful for similar subtree search. To make the concept of tree inclusion more useful, we extend it so that the costs of insertion operations are incorporated and substitutions of node labels are possible.

II. Problem Definition

Let us first define the notation. For any rooted tree T, let $r(T)$ be the root of T, and $V(T)$ the set of nodes in T. For any $v \in V(T)$, $T(v)$ is the rooted subtree (connected subgraph) of T induced by v and all of its descendants. Similarly, for a set of nodes $R = \{v_1, \ldots, v_d\}$, $T(R)$ denotes the subforest (the set of rooted subtrees) of T induced by $v_1, \ldots, v_d$ and all their descendants. Let T_1 be the pattern tree and T_2 the text tree. In what follows, m and n denote the size (the number of nodes) of T_1 and T_2, respectively, and D denotes the maximum outdegree among all nodes in T_1. T_1 is *included* in T_2 if T_2 can be obtained by applying a sequence of insertion operations to T_1. If T_1 is included in T_2, we write $T_1 \prec T_2$. Furthermore, this relation is extended to the case where T_2 can be obtained by applying a sequence of insertion operations to a forest (i.e., a set of rooted trees).

At a first glance, tree inclusion may appear useful for similar subtree search. However, some drawbacks become apparent when trying to apply it to real data. For example, the costs of insertions are not accounted for. Another issue is that substitutions of node labels are not allowed in tree inclusion. To overcome these drawbacks, we introduce *costs* into the tree inclusion problem. Define $D_0(T, T')$ as the minimum cost of transforming T into T' by insertion and substitution operations, where $D_0(T, T') = +\infty$ if there is no such transformation. Also define:

The minimum-cost unordered tree inclusion problem (MinCostIncl): Given two rooted, unordered trees T_1 and T_2, determine the value of $D_{\prec}(T_1, T_2) = \min_{T_2'} D_0(T_1, T_2')$, where T_2' is taken over all connected subgraphs of T_2.

Note that T_2' corresponds to the subtree (of a large tree T_2) similar to T_1. Note also that the definition of $D_{\prec}$ ignores the costs of inserting irrelevant nodes into T_1. As defined above, the problem **MinCostIncl** is an extended unordered tree inclusion problem and it is still NP-hard in general.

III. Main Algorithm

We assume that all insertion and substitution operations have non-negative costs because otherwise, there would be cases in which the cost between two isomorphic trees could be greater than the cost between two non-isomorphic trees.

Our algorithm is called MinCostIncl. It extends the algorithm UnordInclusion of [4] referred to above by also taking the costs of insertions and deletions into account. For certain subsets of nodes $R \subseteq V(T_1)$, MinCostIncl computes a score denoted by $w_v(R)$, which gives the minimum cost of embedding the forest $T_1(R)$ into $T_2(v)$. Here, embedding a forest S into a tree T means that T is obtained by insertion and substitution operations on S, where insertion of a parent of some roots of the current forest is allowed. Then, the required cost is given by: $D_{\prec}(T_1, T_2) = \min_{v \in V(T_2)} w_v(\{r(T_1)\})$.

The pseudocode of MinCostIncl is listed in Algorithm 1. In the algorithm, all $w_v(S)$ (resp., $W_v(S)$) are implicitly

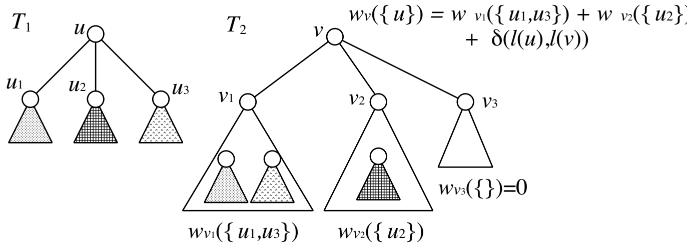

Fig. 1. Illustrating the computation of minimum-cost tree inclusion. Here, u is mapped to v.

initialized as $w_v(S) \leftarrow +\infty$ (resp., $W_v(S) \leftarrow +\infty$), and $W_v(R) < +\infty$ finally gives the minimum cost of embedding $T_1(R)$ into $T_2(v) - \{v\}$.

Algorithm 1 MinCostIncl(T_1, T_2)

```
for all $v \in V(T_2)$ from the leaves to the root do
  Let $v_1, \ldots, v_d$ be the children of $v$;
  $w_v(\emptyset) \leftarrow 0$;
  $W_v(\emptyset) \leftarrow 0$;
  for all $u \in V(T_1)$ from the leaves to the root do
    for all $R_1, \ldots, R_d$ such that $R_i \cap R_j = \emptyset$ (for all $i \neq j$),
    $w_{v_i}(R_i) < +\infty$, and $R_i \subseteq chd(u)$ do
      $R \leftarrow R_1 \cup \cdots \cup R_d$;
      $w \leftarrow w_{v_1}(R_1) + \cdots + w_{v_d}(R_d)$;
      $W_v(R) \leftarrow \min(\ w,\ W_v(R)\ )$;                         (#1)
      $w_v(R) \leftarrow \min(\ w_v(R),\ w + \delta(-, \ell(v)))$;      (#2)
    end for
    if $W_v(chd(u)) < +\infty$ then                                    (#3)
      $w_v(\{u\}) \leftarrow W_v(chd(u)) + \delta(\ell(u), \ell(v))$;
    for all $v_i$ such that $w_{v_i}(\{u\}) < +\infty$ do             (#4)
      $w_v(\{u\}) \leftarrow \min(\ w_v(\{u\}),\ w_{v_i}(\{u\}) + \delta(-, \ell(v))\ )$.
  end for
end for
```

Fig. 1 illustrates the core part of the algorithm. It corresponds to the case of $R_1 = \{u_1, u_3\}$, $R_2 = \{u_2\}$, and $R_3 = \{\}$, which means that $T_1(u_1)$ and $T_1(u_3)$ are embedded into $T_2(v_1)$, $T_1(u_2)$ is embedded into $T_2(v_2)$, and no subtree is embedded into $T_2(v_3)$. In this case, the cost of embedding $T_1(u)$ into $T_2(v)$ is given by $w_{v_1}(R_1) + w_{v_2}(R_2) + w_{v_3}(R_3) + \delta(\ell(u), \ell(v))$, where u corresponds to v, and $\delta(l_1, l_2)$ denotes the cost of substituting the label l_1 by l_2. It is to be noted that $\delta(\ell(u), \ell(v))$ is computed in "then" part of (#3). We examine all partitions (i.e., all $(R_1, \ldots, R_d)$s) of the children of u and take the minimum cost one (this minimum is computed at (#1)). (#2) takes care of the case in which children of u correspond to descendants of v but u does not correspond to v. (#4) takes care of the case in which u corresponds to a descendant of v. We note that the "**for all** $R_1, \ldots, R_d$"-loop can be implemented efficiently by applying dynamic programming (DP) from the leftmost child to the rightmost child.

Theorem 1. $D_{\prec}(T_1, T_2)$ can be computed in $O(2^{2D}mn)$ time using $O(2^D mn)$ space.

IV. Experimental Results

All experiments were performed on a PC cluster with Intel(R) Xeon(R) CPU E5-2690 2.90GHz and 35.87 GB memory, running on a Linux operating system. Our algorithms were implemented using the C++ language and each execution was performed as a single process (i.e., no parallel processes), where very minor simplifications were done in the implemented versions. In this section, m, n, d, D and ℓ denote the size of the pattern tree ($|V(T_1)|$), the size of the text tree ($|V(T_2)|$), the average outdegree, the maximum outdegree, and the alphabet size ($|\Sigma|$), respectively.

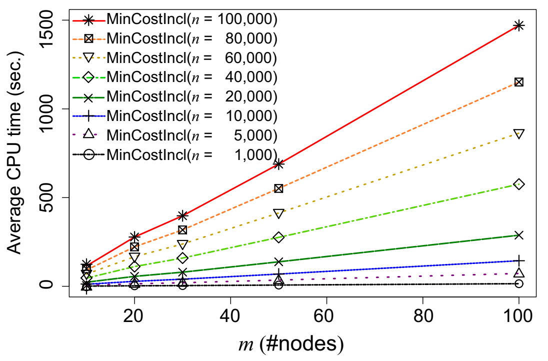

Fig. 2. Execution times of MinCostIncl for varying sizes of the text trees. Note that $m \leq n$ must hold from the definition of MinCostIncl.

To evaluate how the running time of MinCostIncl depends on the sizes of the input trees, we randomly generated 100 pairs of pattern and text trees and measured the average CPU time for each pair. The parameters m and n were varied and the other parameters were fixed as $(d, D, \ell) = (3, 5, 10)$ for both the pattern and text trees. The results are shown in Fig. 2. We can observe from this figure that the computation time increases linearly with m. In the same way, we observed linear processing time for n, which matches the theoretical bound of $O(2^{2D}mn)$ on the running time. Note that MinCostIncl is fast for large text trees (e.g., $n = 100,000$). Although it is not fast enough for real-time applications, the performance is allowable for batch processing. We also conducted experiments using real datasets and observed that MinCostIncl is fast and scalable.

V. Conclusion

In this paper, we have extended the concept of unordered tree inclusion to take the costs of insertions and substitutions into account. The resulting algorithm, MinCostIncl, has the same time complexity as the original algorithm of [4] for unordered tree inclusion ($O(2^{2D}mn)$). Computational experiments on a large synthetic dataset as well as real datasets showed that our proposed algorithm is fast and scalable. Source codes of the implemented algorithms are available upon request.

References

[1] M. Pawlik and N. Augsten, "RTED: A robust algorithm for the tree edit distance," *Proc. the VLDB Endowment*, 5(4), pp. 334-345, 2012.

[2] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, "An optimal decomposition algorithm for tree edit distance," *ACM Trans. Algorithms*, vol. 6, no. 1, article 2, 2009.

[3] K. Zhang, R. Statman, and D. Shasha, "On the editing distance between unordered labeled trees," *Inf. Proc. Lett.*, vol. 42, pp. 133–139, 1992.

[4] P. Kilpeläinen and H. Mannila, "Ordered and unordered tree inclusion," *SIAM J. Computing*, vol. 24, pp. 340–356, 1995.