Faster Weighted and Unweighted Tree Edit Distance and APSP Equivalence

Jakob Nogler[1] Adam Polak[2] Barna Saha[3] Virginia Vassilevska Williams[4] Yinzhan Xu[3] Christopher Ye[3]

[1]ETH Zurich [2]Bocconi University [3]UC San Diego [4]MIT

Problem Definition

(String) Edit Distance

Input: Strings S, S'.
Output: Cheapest transformation of S into S'.

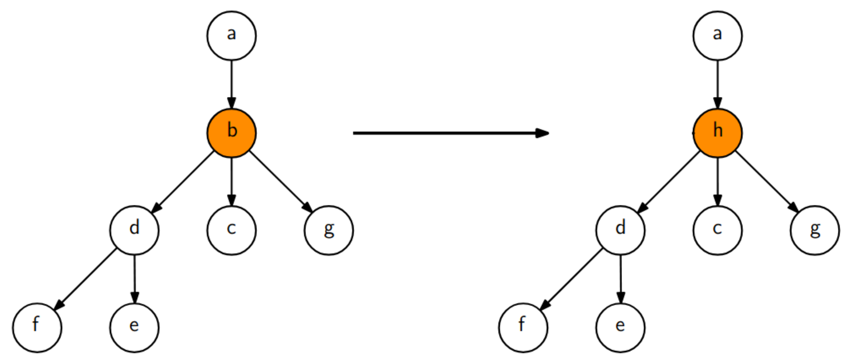
Tree Edit Distance (TED)

Input: Labeled, left-to-right ordered trees T, T'.
Output: Cheapest transformation of T into T'.

Allowed Operations:

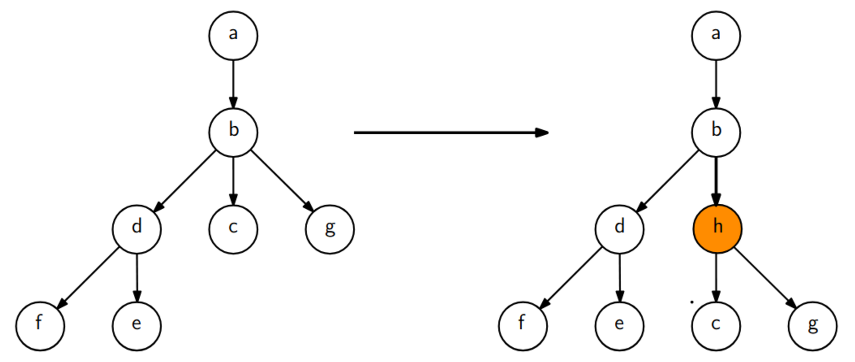
1. Substitution

`abcxef` ⟶ `abcyef`



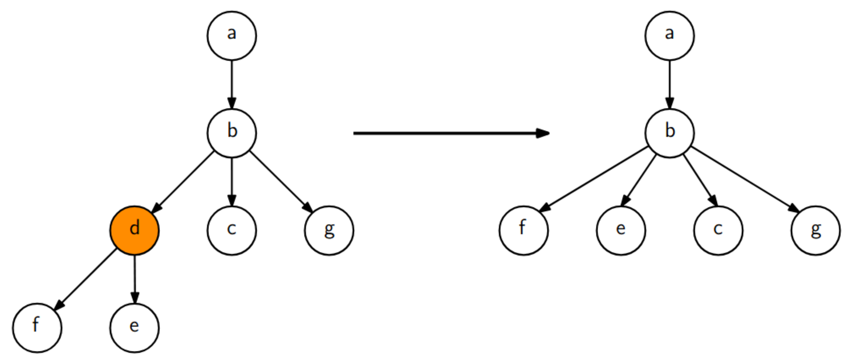
2. Insertion

`abcef` ⟶ `abcxef`



3. Deletion

`abcdef` ⟶ `abdef`



Right hand side images taken from **[SS19]**.

Unweighted: All operations have cost 1.
Weighted: Cost of operations is specifced by a cost function δ, i.e. $\delta(c, c')$, $\delta(c, \varepsilon)$ and $\delta(\varepsilon, c)$ specify substitution, deletion and insertion cost of character c.

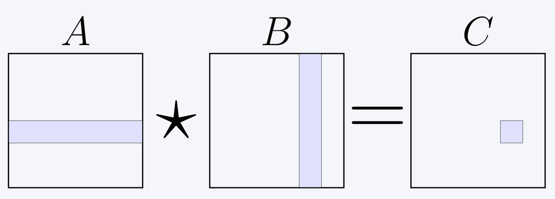
Previous Works and Our Results

Year	Work	Setting	Complexity
1979	Tai	weighted	$\mathcal{O}(n^6)$
1989	Shasha, Zhang	weighted	$\mathcal{O}(n^4)$
1998	Klein	weighted	$\mathcal{O}(n^3 \log n)$
2007	Demaine, Mozes, Rossman, Weimann	weighted	$\mathcal{O}(n^3)$
2020	Bringmann, Gawrychowski, Mozes, Weimann	weighted	no $\mathcal{O}(n^{3-\epsilon})$ algo under APSP
2024	**This work**	**weighted**	$\boldsymbol{n^3/2^{\Omega(\sqrt{\log n})}}$
2022	Mao	unweighted	$\mathcal{O}(n^{2.9546})$
2023	Dürr	unweighted	$\mathcal{O}(n^{2.9148})$
2024	**This work**	**unweighted**	$\boldsymbol{\mathcal{O}(n^{2.687})}$

More details on our results:

- We show how to reduce TED to APSP. Combined with the APSP lower bound, this gives equivalence between TED and APSP.
- Via the state-of-the-art APSP Algorithm **[Williams'18]** we get faster weighted TED.
- Previous truly subcubic algorithm by Mao and Dürr, reduce (unweighted) TED to monotone min-plus product, but lose some polynomial factors on the way. We show how to get a tight reduction. By doing so, via the state-of-the-art algorithm for this product **[CDXZ'22]**, we obtain a $\mathcal{O}(n^{(\omega+3)/2})$-time algorithm.

Monotone Min-plus Product



$C_{i,j} = \min_{1 \le k \le n} \{A_{i,k} + B_{k,j}\}$

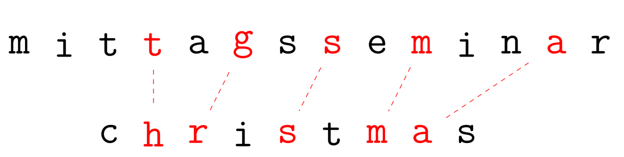
- B is row monotone:
  $\forall i, j \quad B_{i,j} \le B_{i,j+1}.$
- A, B are bounded:
  $\forall i, j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$

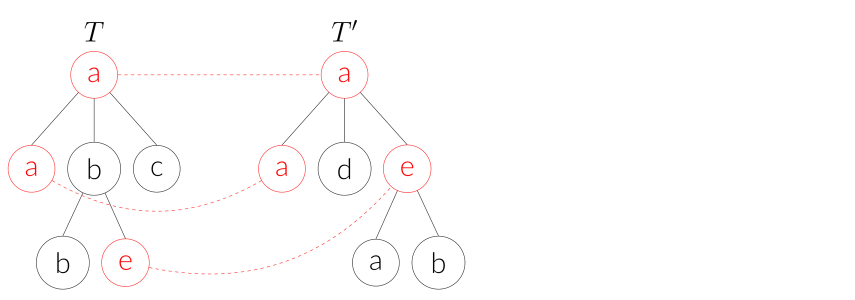
Problem Re-Definition

Rather than finding we finding a least-cost transformation, we aim to a maximize a matching.

We define the cost $\eta(c, c')$ of matching two characters:

$$\eta(c, c') := \delta(c, \varepsilon) + \delta(\varepsilon, c') - \delta(c, c').$$





String Similarity: maximum total weight matching, under the condition that matching do not *cross*.

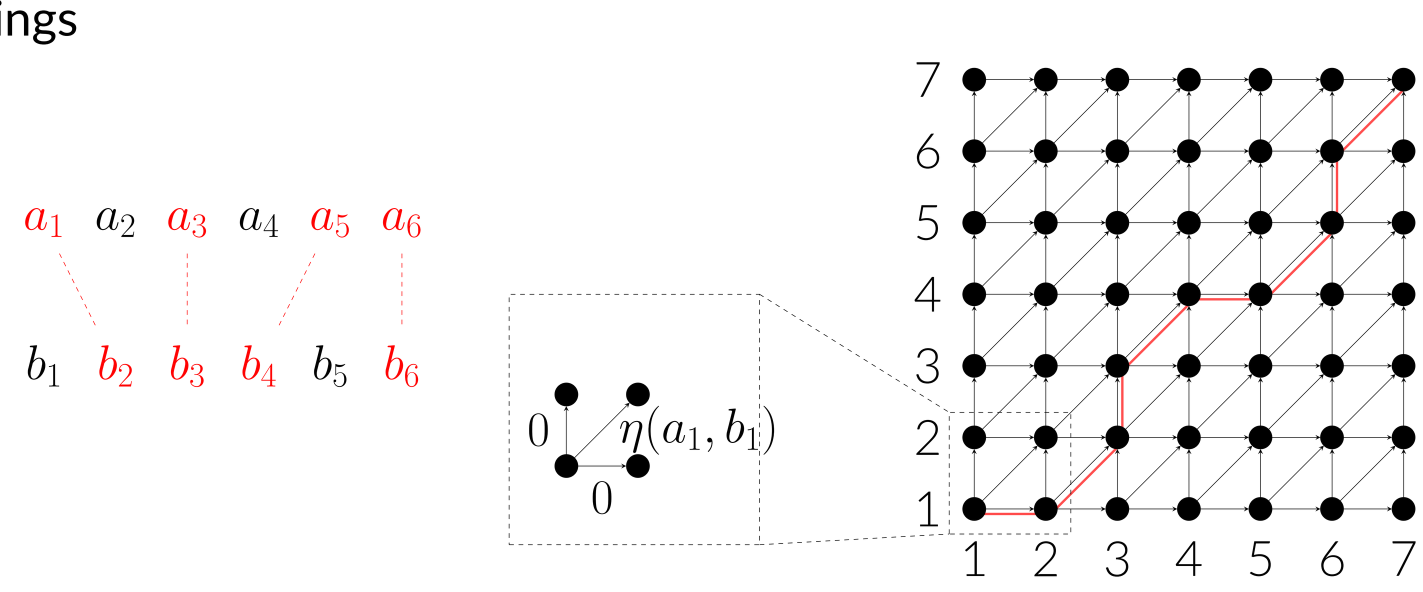
Tree Similarity: maximum total weight matching, under the condition that matching respect same ancestry and pre-order orderings of the matched nodes.

The following relation holds for string/tree distance and similarity:

$$\mathsf{sim}(A, B) = \sum_{a \in A} \delta(a, \varepsilon) + \sum_{b \in B} \delta(\varepsilon, b) - \mathsf{ed}(A, B).$$

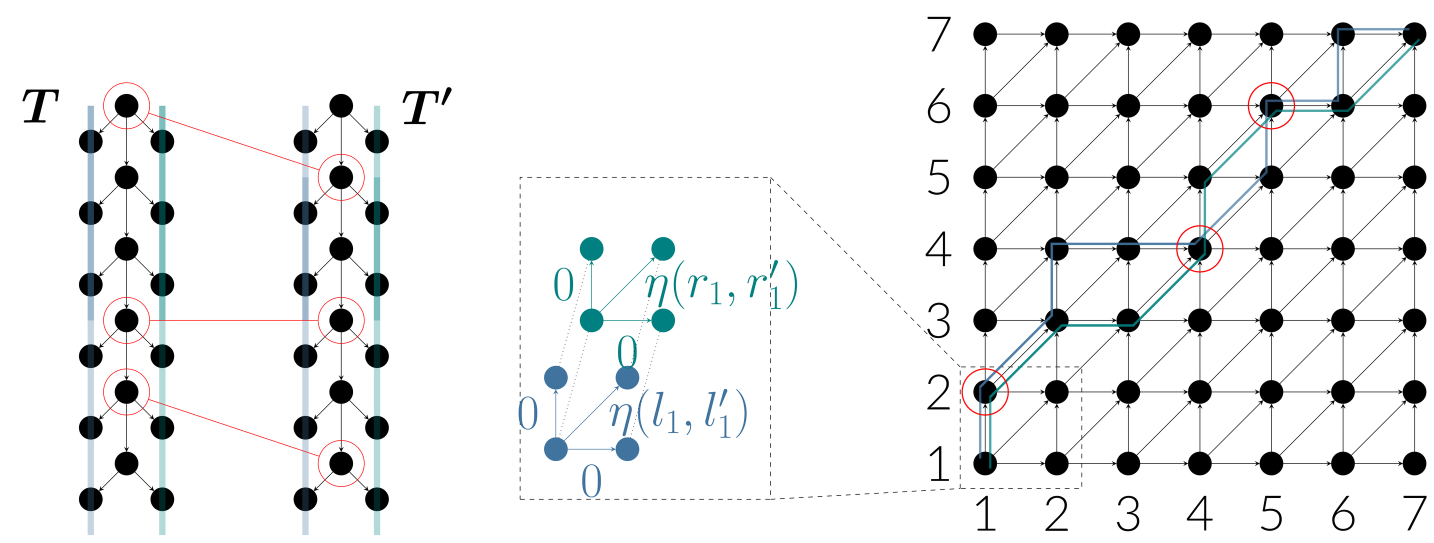
Similarity Computation on a Grid: Strings vs Caterpillar Trees

1. Strings



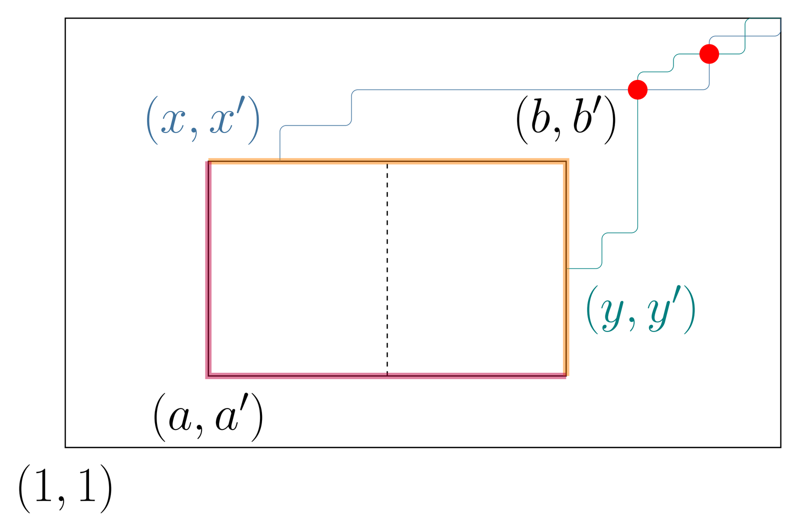
N.B. Border-to-border distances in such graphs can be computed in $\mathcal{O}(n^2)$ time.

2. Caterpillar Trees



N.B. Here we assume the left, right and spine nodes $l_1, l_2, \ldots, r_1, r_2, \ldots$, and $c_1, c_2, \ldots$ of $\mathbf{T}$ only match with the left, right and spine nodes $l'_1, l'_2, \ldots, r'_1, r'_2, \ldots$, and $c'_1, c'_2, \ldots$ of $\mathbf{T'}$.

A Divide and Conquer Scheme for Caterpillar Tree Similarity



Input:

- The lower-left corner (a, a') and upper-right corner (b, b') of a rectangle.
- $\mathsf{sim}((x, x'), (y, y'))$ $\forall (x, x'), (y, y') \in ([a\,..\,b] \times \{b'\}) \cup (\{b\} \times [a'\,..\,b'])$.

Output:

- $\mathsf{sim}((x, x'), (y, y'))$ $\forall (x, x'), (y, y') \in ([a\,..\,b] \times \{a'\}) \cup (\{a\} \times [a'\,..\,b'])$.

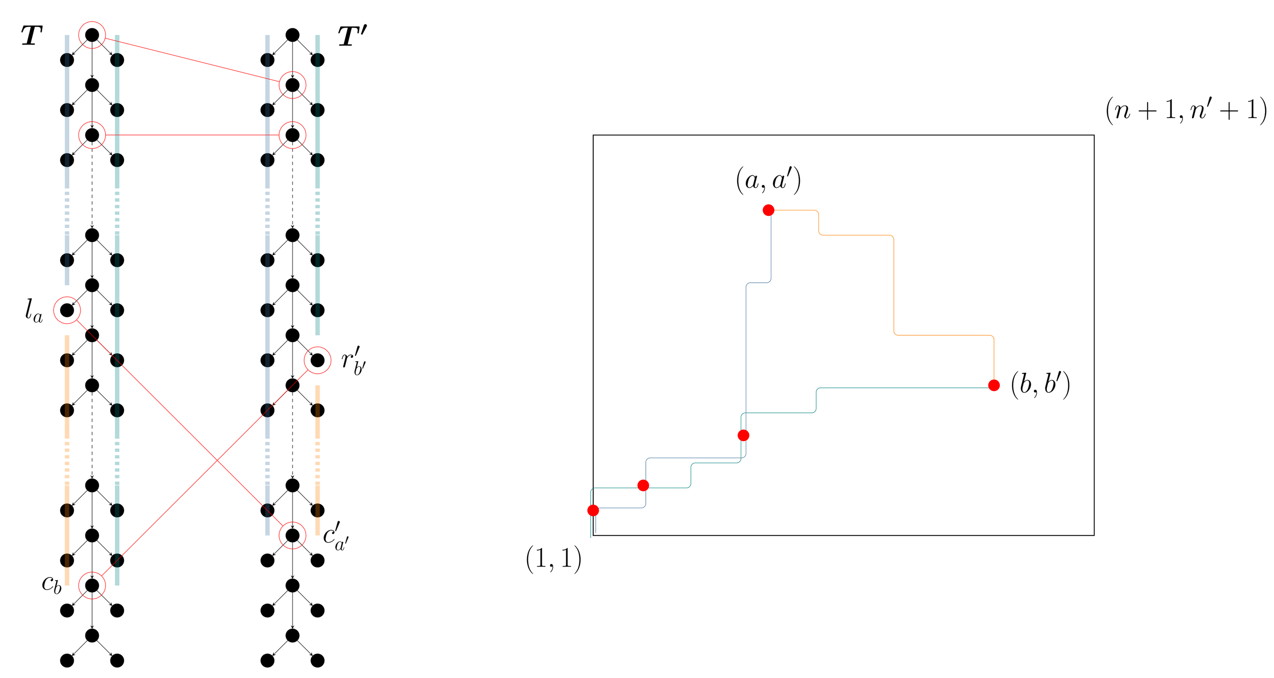
$\mathsf{sim}((x, x'), (y, y'))$ equals to the maximum achievable sum of:

1. the weight of a path from (x, x') to $(n+1, n+1)$ in the alignment graph of $\mathsf{sim}(L, L')$;
2. the weight of a path from (y, y') to $(n+1, n+1)$ in the alignment graph of $\mathsf{sim}(R, R')$; and
3. values $\eta(c_i, c'_{i'})$ for (i, i') where the two paths intersect (each c_i and $c'_{i'}$ appears at most once).

We obtain the recurrence $T(n) = 4T(n/2) + \mathcal{O}(T_{\mathsf{APSP}}(n))$. Thus, $T(n) = \mathcal{O}(T_{\mathsf{APSP}}(n))$.

How to Extend to General Caterpillars

Dropping the assumption on the mapping adds one more path in a string alignment graph.



How to Extend to General Trees

We generalize TED on caterpillar trees to Spine Edit Distance.

Input: trees T, T', root-to-leaf paths $S \subseteq T, S' \subseteq T'$, and $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ $\forall (v, v') \in (T \times T') \setminus (S \times S')$.

Images taken from **[BGHS19]**

Output: $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ $\forall (v, v') \in S \times S'$.

Why?

- There are still spine, left, and right nodes.
- The underlying paths are not in string alignment graphs anymore but in *forest alignment graphs*.
- TED on arbitrary trees is fine-grained equivalent to Spine Edit Distance.

STOC 2025 — June 23–27, 2025 — Prague, Czechia

https://arxiv.org/abs/2411.06502