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

(String) Edit Distance

(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

(String) Edit Distance

(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

1. Substitute a characters c with c' with cost $\delta(c, c')$

$$\texttt{abcxef} \xrightarrow{\text{Substitute } \texttt{x} \text{ with } \texttt{y}} \texttt{abcyef}$$

(String) Edit Distance

(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

1. Substitute a characters c with c' with cost $\delta(c, c')$

$$\texttt{abcxef} \xrightarrow{\text{Substitute } \texttt{x} \text{ with } \texttt{y}} \texttt{abcyef}$$

2. Delete a character c with cost $\delta(c, \varepsilon)$

$$\texttt{abcdef} \xrightarrow{\text{Delete } \texttt{c}} \texttt{abdef}$$

(String) Edit Distance

(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

1. Substitute a characters c with c' with cost $\delta(c, c')$

$$\texttt{abcxef} \xrightarrow{\text{Substitute } \texttt{x} \text{ with } \texttt{y}} \texttt{abcyef}$$

2. Delete a character c with cost $\delta(c, \varepsilon)$

$$\texttt{abcdef} \xrightarrow{\text{Delete c}} \texttt{abdef}$$

3. Insert a character c with cost $\delta(\varepsilon, c)$

$$\texttt{abcef} \xrightarrow{\text{Insert } \texttt{x}} \texttt{abcxef}$$

(String) Edit Distance

(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

1. Substitute a characters c with c' with cost $\delta(c, c')$

$$\texttt{abcxef} \xrightarrow{\text{Substitute } \texttt{x} \text{ with } \texttt{y}} \texttt{abcyef}$$

2. Delete a character c with cost $\delta(c, \varepsilon)$

$$\texttt{abcdef} \xrightarrow{\text{Delete } \texttt{c}} \texttt{abdef}$$

3. Insert a character c with cost $\delta(\varepsilon, c)$

$$\texttt{abcef} \xrightarrow{\text{Insert } \texttt{x}} \texttt{abcxef}$$

"Unweighted" (String) Edit Distance: all costs are one

Tree Edit Distance (TED)

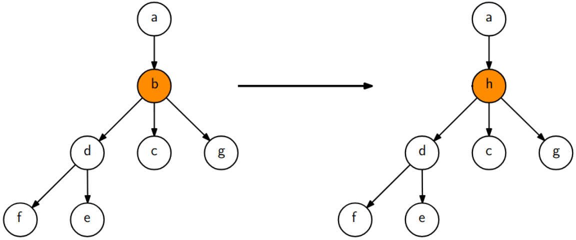
Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2.

Tree Edit Distance (TED)

Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2.

- Substitute v with v' with cost $\delta(v, v')$.

Substitute:

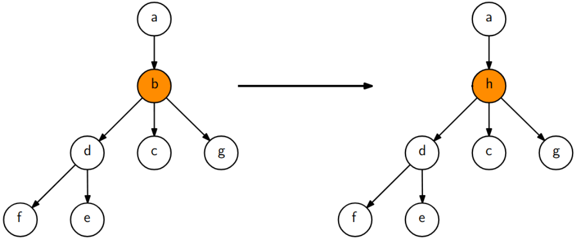


Tree Edit Distance (TED)

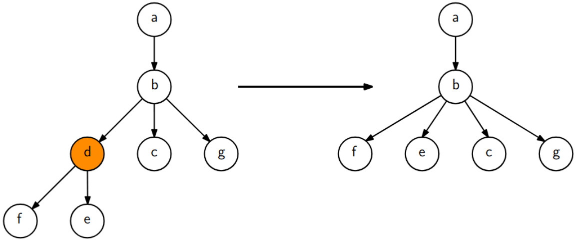
Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2.

- Substitute v with v' with cost $\delta(v, v')$.
- Delete v with cost $\delta(v, \varepsilon)$.

Substitute:



Delete:

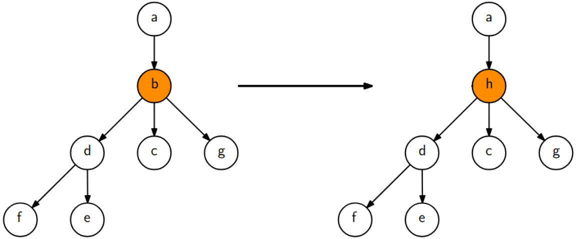


Tree Edit Distance (TED)

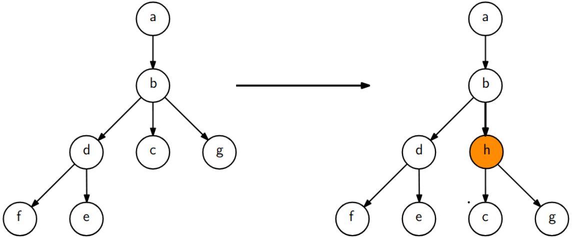
Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2.

- Substitute v with v' with cost $\delta(v, v')$.
- Delete v with cost $\delta(v, \varepsilon)$.
- Insert v' with cost $\delta(\varepsilon, v')$.

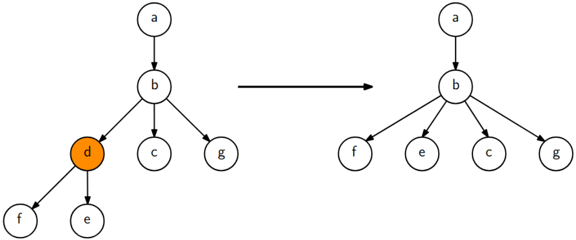
Substitute:



Insert:



Delete:



Images from Seddighin, Seddighin 2019

TED reformulated

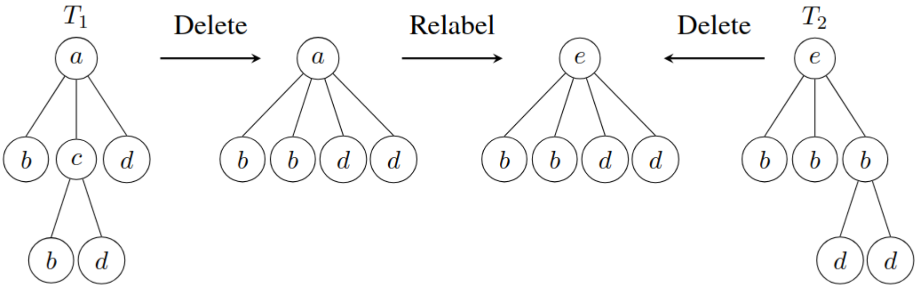


Image from Mao 2022

- Relabel v to v' with cost $\delta(v, v')$.
- Delete v from T_1 with cost $\delta(v, \varepsilon)$.
- Delete v' from T_2 with cost $\delta(\varepsilon, v')$.

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

Year	Work	Setting	Complexity
1979	Tai	weighted	$\mathcal{O}(n^6)$

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

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	$\mathcal{O}(n^3)$

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

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	$\mathcal{O}(n^3)$

Last three fall within decomposition strategy framework formalized in [Dulucq and Touzet, 2003].
For algorithms within the framework a $\Omega(n^3)$ lower bound exists.

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

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, Weinmann	weighted	no $\mathcal{O}(n^{3-\varepsilon})$ algo under APSP

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

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, Weinmann	weighted	no $\mathcal{O}(n^{3-\varepsilon})$ algo under APSP
2022	Mao	unweighted	$\mathcal{O}(n^{2.9546})$
2023	Dürr	unweighted	$\mathcal{O}(n^{2.9148})$

Algorithms for Tree Edit Distance

Background: Introduced by Selkow in the late 1970s. Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

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, Weinmann	weighted	no $\mathcal{O}(n^{3-\varepsilon})$ algo under APSP
2022	Mao	unweighted	$\mathcal{O}(n^{2.9546})$
2023	Dürr	unweighted	$\mathcal{O}(n^{2.9148})$

Question 1: is there a $o(n^3)$ algorithm for (weighted) TED?

All-Pair Shortest Path Problem (APSP)
Input: A weighted and directed graph G.
Output: Shortest distance between every pair of nodes.

All-Pair Shortest Path Problem (APSP)
Input: A weighted and directed graph G.
Output: Shortest distance between every pair of nodes.

APSP Conjecture

There is no algorithm for APSP running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

The Fine-grained Complexity of Tree Edit Distance

All-Pair Shortest Path Problem (APSP)
Input: A weighted and directed graph G.
Output: Shortest distance between every pair of nodes.

APSP Conjecture

There is no algorithm for APSP running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

All-Pair Shortest Paths

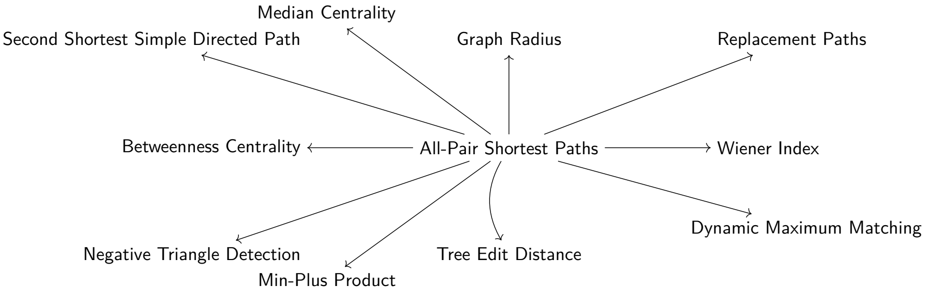
Tree Edit Distance

The Fine-grained Complexity of Tree Edit Distance

All-Pair Shortest Path Problem (APSP)
Input: A weighted and directed graph G.
Output: Shortest distance between every pair of nodes.

APSP Conjecture

There is no algorithm for APSP running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

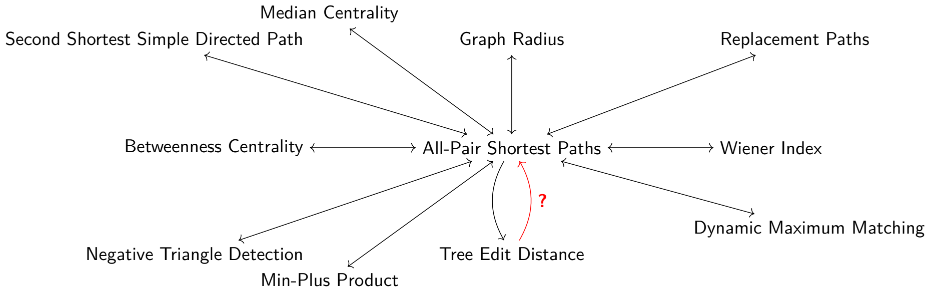


The Fine-grained Complexity of Tree Edit Distance

All-Pair Shortest Path Problem (APSP)
Input: A weighted and directed graph G.
Output: Shortest distance between every pair of nodes.

APSP Conjecture

There is no algorithm for APSP running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

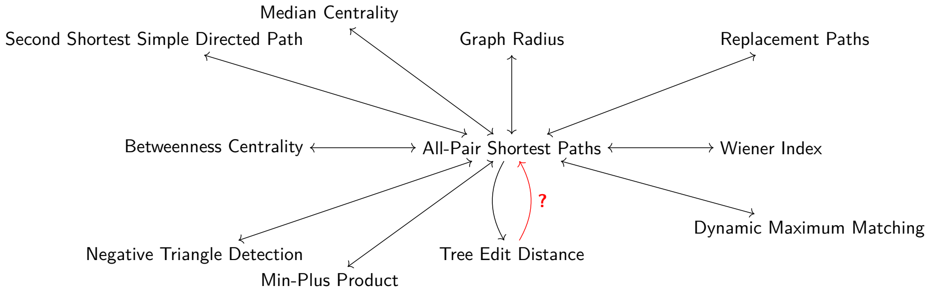


The Fine-grained Complexity of Tree Edit Distance

All-Pair Shortest Path Problem (APSP)
Input: A weighted and directed graph G.
Output: Shortest distance between every pair of nodes.

APSP Conjecture

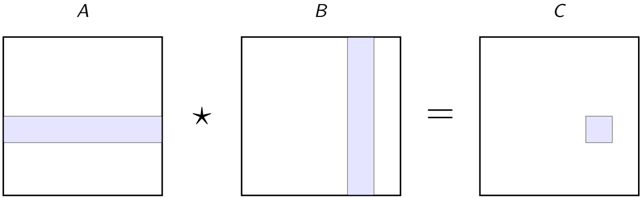
There is no algorithm for APSP running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.



Question 2: is TED equivalent to APSP?

Key component to achieve truly subcubic algorithms for unweighted TED:

Monotone Min-plus Product



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

- B **is row monotone**:

$$\forall i, j \quad B_{i,j} \le B_{i,j+1}.$$

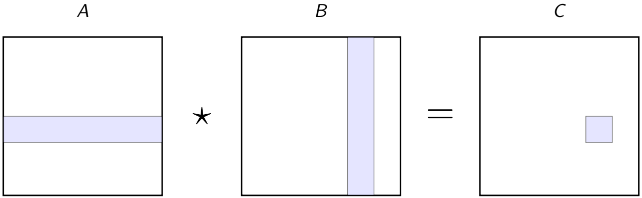
- A, B **are bounded**:

$$\forall i, j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

Unweighted Tree Edit Distance

Key component to achieve truly subcubic algorithms for unweighted TED:

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**:

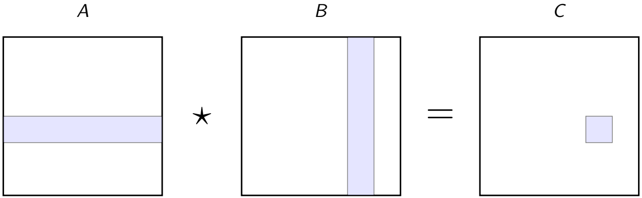
$$\forall i,j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

Chi, Duan, Xie, Zhang '22: $T_{\text{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$.

Unweighted Tree Edit Distance

Key component to achieve truly subcubic algorithms for unweighted TED:

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**:

$$\forall i, j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

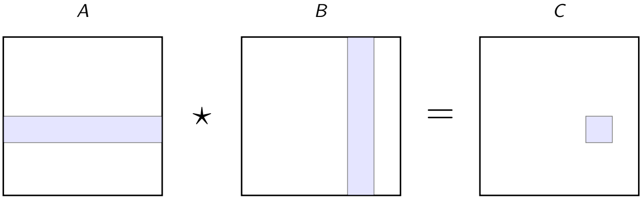
Chi, Duan, Xie, Zhang '22: $T_{\text{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$.

Algorithms for unweighted TED of Mao & Dürr lose factors in the exponent and use other techniques that only apply to the unweighted case.

Unweighted Tree Edit Distance

Key component to achieve truly subcubic algorithms for unweighted TED:

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**:

$$\forall i,j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

Chi, Duan, Xie, Zhang '22: $T_{\text{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$.

Algorithms for unweighted TED of Mao & Dürr lose factors in the exponent and use other techniques that only apply to the unweighted case.

Question 3: is there a $\mathcal{O}(n^{(\omega+3)/2})$ algorithm for unweighted TED?

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\mathrm{APSP}}(n) + n^{2+o(1)})$.

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\mathrm{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\mathrm{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

Williams '18: $T_{\mathrm{APSP}}(n) = n^3/2^{\Omega(\sqrt{\log n})}$.

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\text{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

Williams '18: $T_{\text{APSP}}(n) = n^3/2^{\Omega(\sqrt{\log n})}$.

Theorem 2

There is an algorithm for TED running in time $n^3/2^{\Omega(\sqrt{\log n})}$.

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\mathrm{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

Williams '18: $T_{\mathrm{APSP}}(n) = n^3/2^{\Omega(\sqrt{\log n})}$.

Theorem 2

There is an algorithm for TED running in time $n^3/2^{\Omega(\sqrt{\log n})}$.

Question 1: is there a $o(n^3)$ algorithm for (weighted) TED? ✔

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\mathrm{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

Williams '18: $T_{\mathrm{APSP}}(n) = n^3/2^{\Omega(\sqrt{\log n})}$.

Theorem 2

There is an algorithm for TED running in time $n^3/2^{\Omega(\sqrt{\log n})}$.

Question 1: is there a $o(n^3)$ algorithm for (weighted) TED? ✔

Theorem 3

There is an algorithm for unweighted TED running in time $\mathcal{O}(T_{\mathrm{MonMUL}}(n) + n^{2+o(1)})$.

Chi, Duan, Xie, Zhang '22: $T_{\mathrm{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2})$.

Results

Theorem 1

There is an algorithm for TED running in time $\mathcal{O}(T_{\text{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

Williams '18: $T_{\text{APSP}}(n) = n^3/2^{\Omega(\sqrt{\log n})}$.

Theorem 2

There is an algorithm for TED running in time $n^3/2^{\Omega(\sqrt{\log n})}$.

Question 1: is there a $o(n^3)$ algorithm for (weighted) TED? ✔

Theorem 3

There is an algorithm for unweighted TED running in time $\mathcal{O}(T_{\text{MonMUL}}(n) + n^{2+o(1)})$.

Chi, Duan, Xie, Zhang '22: $T_{\text{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2})$.

Question 3: is there a $\mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ algorithm for unweighted TED? ✔

Algorithms for Tree Edit Distance (Updated)

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, Weinmann	weighted	no $\mathcal{O}(n^{3-\varepsilon})$ 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**	$\boldsymbol{\mathcal{O}(n^{2.687})}$

Sketch of the Reduction

Similarity of Strings

Instead of computing the edit distance between two strings $A = a_1 \cdots a_n$, $B = b_1 \cdots b_n$, we compute the similarity between A, B.

m i t a g s e m i n a r

c h r i s t m a s

$\eta(a_i, b_j) := \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ "how much I save by substituting a_i with b_j"

Similarity of Strings

Instead of computing the edit distance between two strings $A = a_1 \cdots a_n$, $B = b_1 \cdots b_n$, we compute the similarity between A, B.

`m i t a g s e m i n a r`

`c h r i s t m a s`

$\eta(a_i, b_j) := \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ "how much I save by substituting a_i with b_j"

$\mathrm{sim}(A, B) := \max_{\substack{i_1 < \cdots < i_k \in [1\,..\,n] \\ j_1 < \cdots < j_k \in [1\,..\,n]}} \left\{ \eta(a_{i_1}, b_{j_1}) + \eta(a_{i_2}, b_{j_2}) + \cdots + \eta(a_{i_k}, b_{j_k}) \right\}$. "max I can save"

Similarity of Strings

Instead of computing the edit distance between two strings $A = a_1 \cdots a_n$, $B = b_1 \cdots b_n$, we compute the similarity between A, B.

m i t a g s e m i n a r

c h r i s t m a s

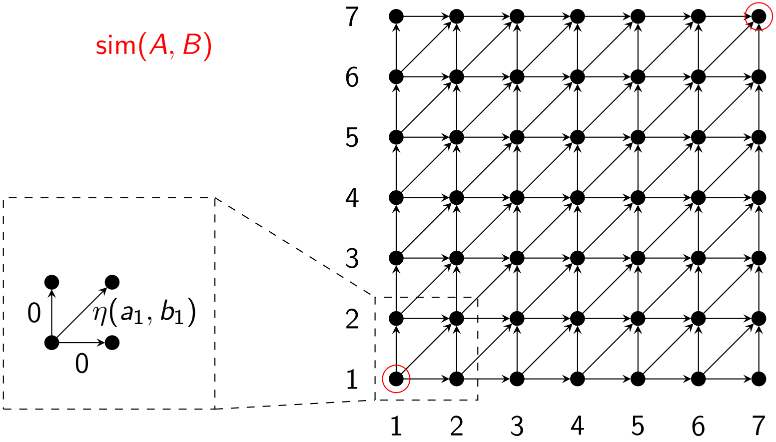
$\eta(a_i, b_j) := \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ "how much I save by substituting a_i with b_j"

$\mathrm{sim}(A, B) := \max_{\substack{i_1 < \cdots < i_k \in [1\,..\,n] \\ j_1 < \cdots < j_k \in [1\,..\,n]}} \left\{ \eta(a_{i_1}, b_{j_1}) + \eta(a_{i_2}, b_{j_2}) + \cdots + \eta(a_{i_k}, b_{j_k}) \right\}$. "max I can save"

$\mathrm{sim}(A, B) = \sum_i \delta(a_i, \varepsilon) + \sum_j \delta(\varepsilon, b_j) - \mathrm{ed}(A, B)$.

String Alignment Graphs

The *string alignment graph* summarizes the DP scheme computing the similarity.

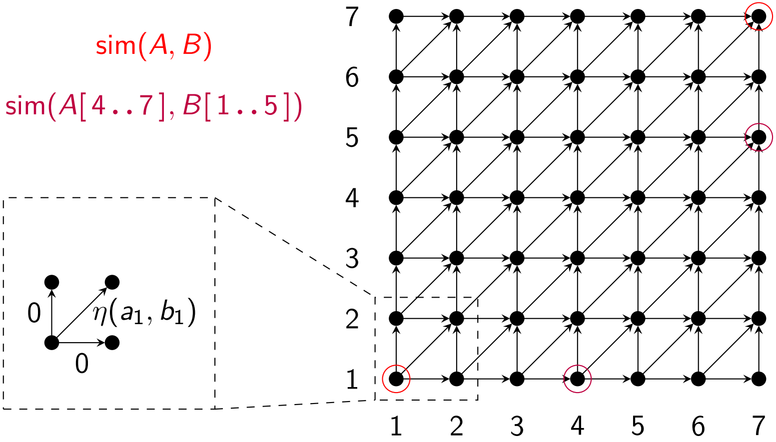


String Alignment Graphs

The *string alignment graph* summarizes the DP scheme computing the similarity.

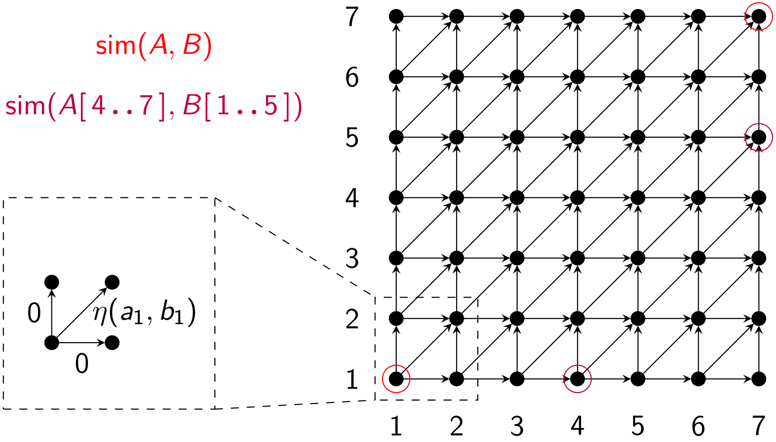
$\mathrm{sim}(A, B)$

$\mathrm{sim}(A[4\,..\,7], B[1\,..\,5])$



String Alignment Graphs

The *string alignment graph* summarizes the DP scheme computing the similarity.

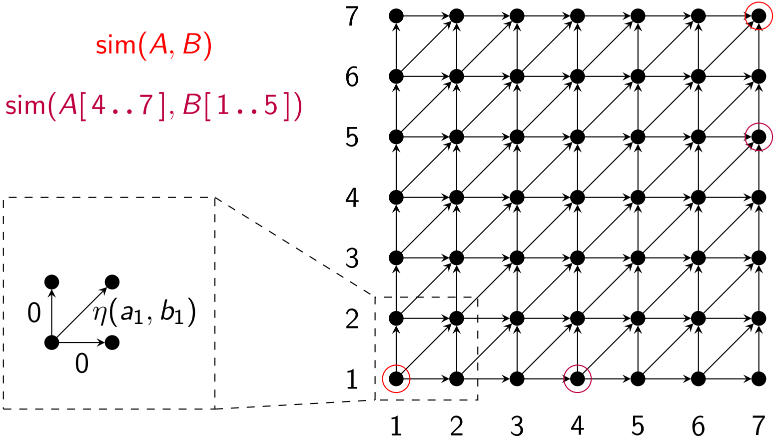


lower-to-right border:

suffix vs prefix

String Alignment Graphs

The *string alignment graph* summarizes the DP scheme computing the similarity.



lower-to-right border:
suffix vs prefix

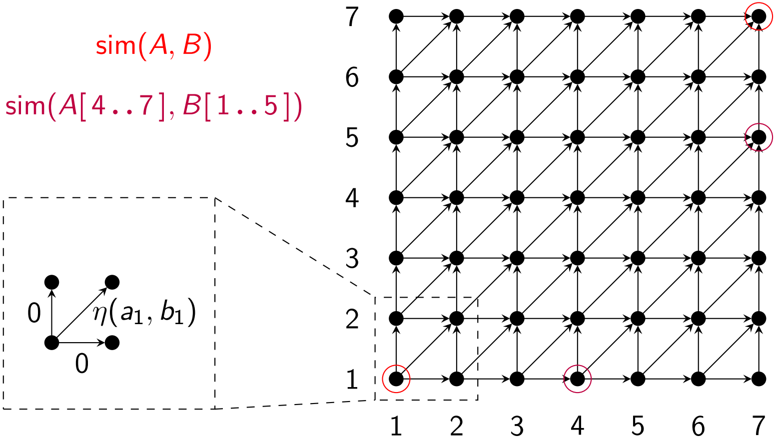
lower-to-upper border:
infix vs whole

left-to-right border:
whole vs infix

lower-to-right border:
prefix vs suffix

String Alignment Graphs

The *string alignment graph* summarizes the DP scheme computing the similarity.



lower-to-right border:
suffix vs prefix

lower-to-upper border:
infix vs whole

left-to-right border:
whole vs infix

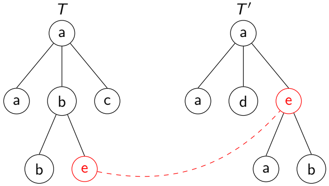
lower-to-right border:
prefix vs suffix

$\mathcal{O}(n^2)$ to compute all

Bedtime reading: "Semi-local string comparison: algorithmic techniques and applications" by Alexander Tiskin

Similarity of Trees

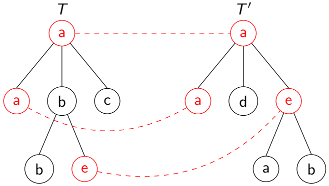
We compute the similarity between T and T'.



$\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ "how much I save by substituting v with v'"

Similarity of Trees

We compute the similarity between T and T'.



$\eta(v, v') \coloneqq \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ "how much I save by substituting v with v'"

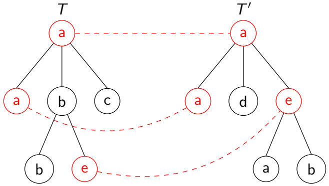
$\mathrm{sim}(T, T') =$"maximum weight of similarity matching"

Condition on similarity matching: for any two matched vertices (v, v') and (u, u')

- v is an ancestor of u in T if and only if v' is an ancestor of u' in T',
- v comes before u in the pre-order of T if and only if v' comes before u' in the pre-order of T'.

Similarity of Trees

We compute the similarity between T and T'.



$\eta(v, v') \coloneqq \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ "how much I save by substituting v with v'"

$\mathrm{sim}(T, T') =$"maximum weight of similarity matching"

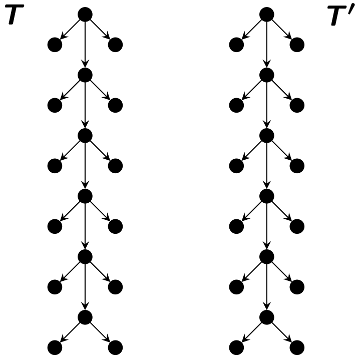
Condition on similarity matching: for any two matched vertices (v, v') and (u, u')

- v is an ancestor of u in T if and only if v' is an ancestor of u' in T',
- v comes before u in the pre-order of T if and only if v' comes before u' in the pre-order of T'.

$\mathrm{sim}(T, T') = \sum_{v \in T} \delta(v, \varepsilon) + \sum_{v' \in T'} \delta(\varepsilon, v') - \mathrm{ted}(T, T')$.

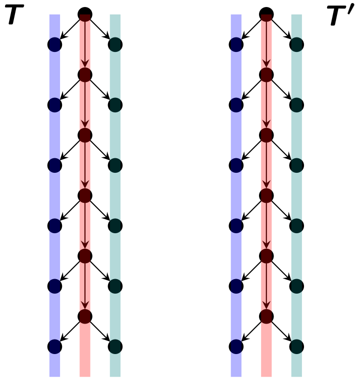
TED on Caterpillar Trees I

Let us start by computing the similarity between two caterpillar trees...



TED on Caterpillar Trees I

Let us start by computing the similarity between two caterpillar trees...



spine nodes

left nodes

right nodes

TED on Caterpillar Trees I

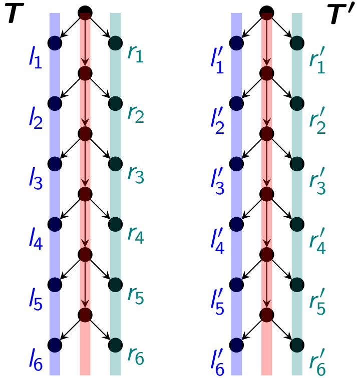
Let us start by computing the similarity between two caterpillar trees...

$L = l_1 l_2 l_3 l_4 l_5 l_6$

$R = r_1 r_2 r_3 r_4 r_5 r_6$

$L' = l'_1 l'_2 l'_3 l'_4 l'_5 l'_6$

$R' = r'_1 r'_2 r'_3 r'_4 r'_5 r'_6$



spine nodes

left nodes

right nodes

TED on Caterpillar Trees I

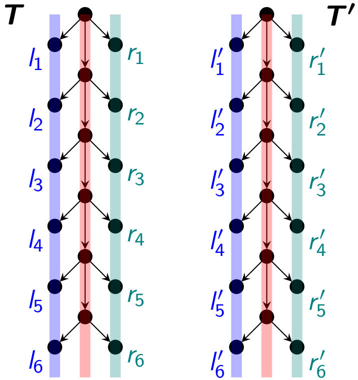
Let us start by computing the similarity between two caterpillar trees...

$L = l_1 l_2 l_3 l_4 l_5 l_6$

$R = r_1 r_2 r_3 r_4 r_5 r_6$

$L' = l'_1 l'_2 l'_3 l'_4 l'_5 l'_6$

$R' = r'_1 r'_2 r'_3 r'_4 r'_5 r'_6$



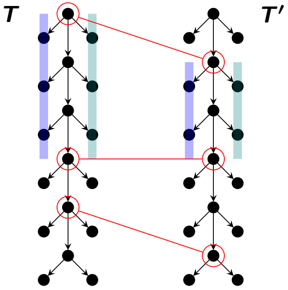
spine nodes

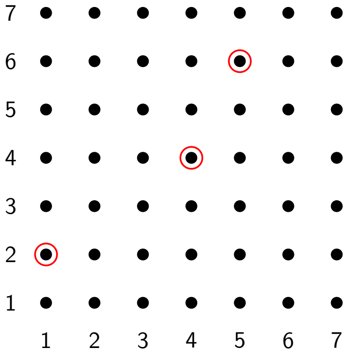
left nodes

right nodes

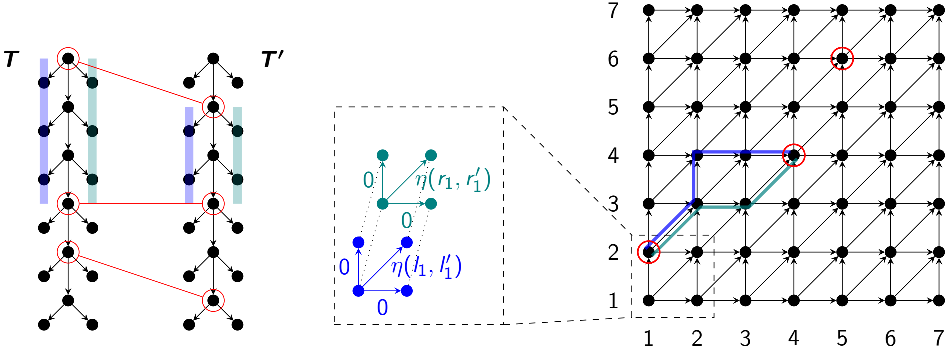
...with the assumption that spine, left and right nodes of T only match with nodes of their same type (color) in T', respectively.

TED on Caterpillar Trees II

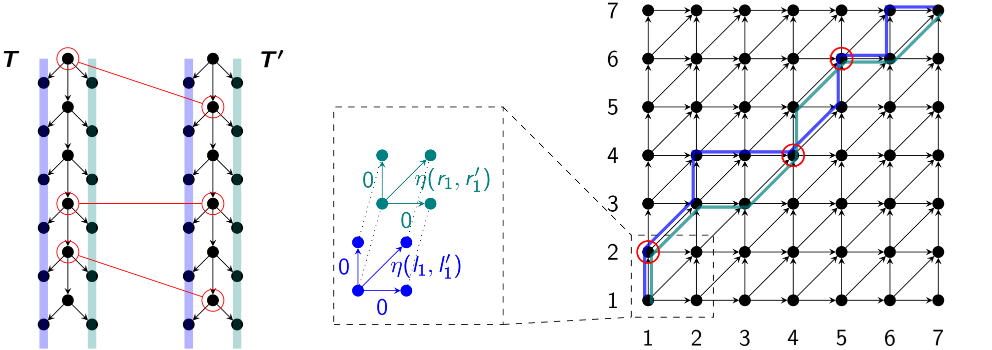




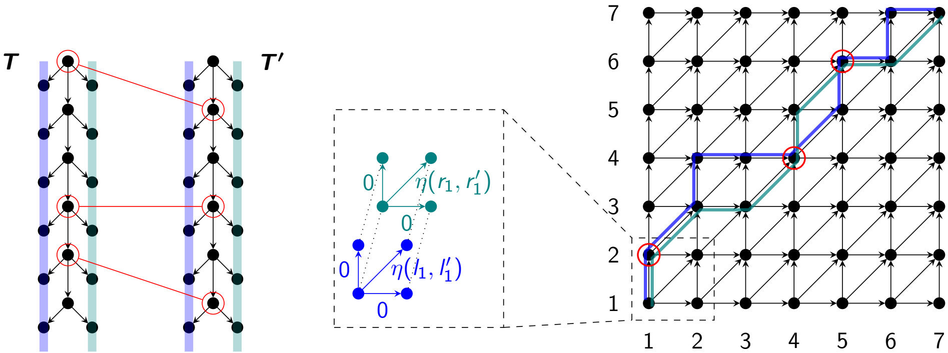
TED on Caterpillar Trees II



TED on Caterpillar Trees II



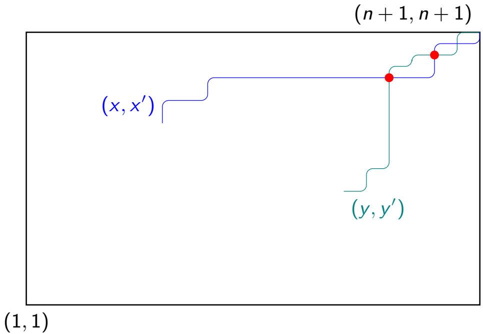
TED on Caterpillar Trees II



sim(T, T') equals to the maximum achievable sum of:

1. the weight of a path from $(1, 1)$ to $(n+1, n+1)$ in the alignment graph of sim(L, L');
2. the weight of a path from $(1, 1)$ to $(n+1, n+1)$ in the alignment graph of 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).

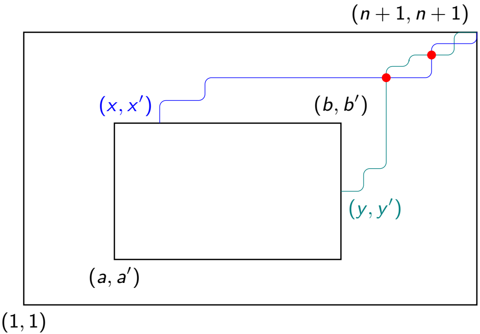
A Divide and Conquer Scheme for TED on Caterpillar Trees I



$\mathrm{sim}((x, x'), (y, y'))$ 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 $\mathrm{sim}(L, L')$;
2. the weight of a path from (y, y') to $(n+1, n+1)$ in the alignment graph of $\mathrm{sim}(R, R')$; 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).

A Divide and Conquer Scheme for TED on Caterpillar Trees I



Divide et Conquer Scheme

Input:

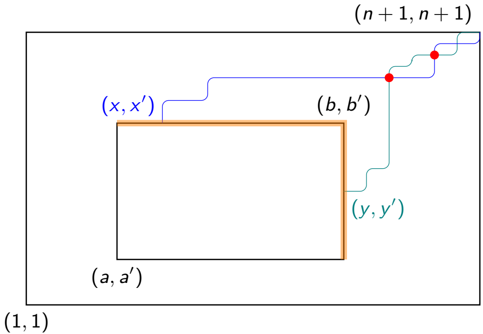
- The lower-left corner (a, a') and upper-right corner (b, b') of a rectangle.

Output:

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 sim(L, L');
2. the weight of a path from (y, y') to $(n+1, n+1)$ in the alignment graph of 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).

A Divide and Conquer Scheme for TED on Caterpillar Trees I



Divide et Conquer Scheme

Input:

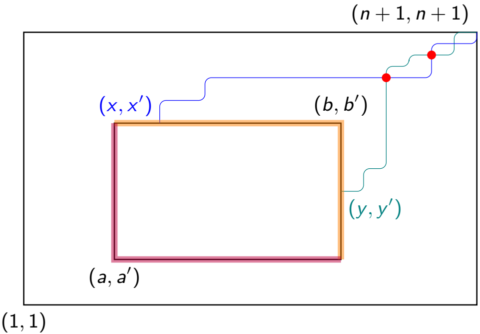
- The lower-left corner (a, a') and upper-right corner (b, b') of a rectangle.
- $\mathrm{sim}((x, x'), (y, y'))$ $\forall (x, x'), (y, y') \in ([\,a \,..\, b\,] \times \{b'\}) \cup (\{b\} \times [\,a' \,..\, b'\,])$.

Output:

$\mathrm{sim}((x, x'), (y, y'))$ 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 $\mathrm{sim}(L, L')$;
2. the weight of a path from (y, y') to $(n+1, n+1)$ in the alignment graph of $\mathrm{sim}(R, R')$; 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).

A Divide and Conquer Scheme for TED on Caterpillar Trees I



Divide et Conquer Scheme

Input:

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

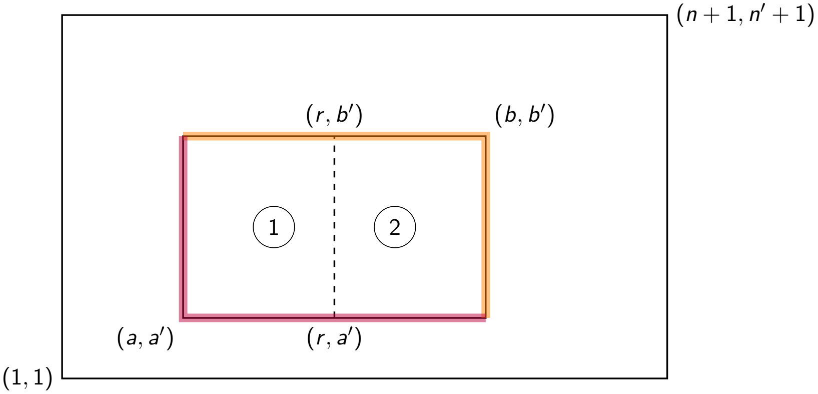
Output:

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

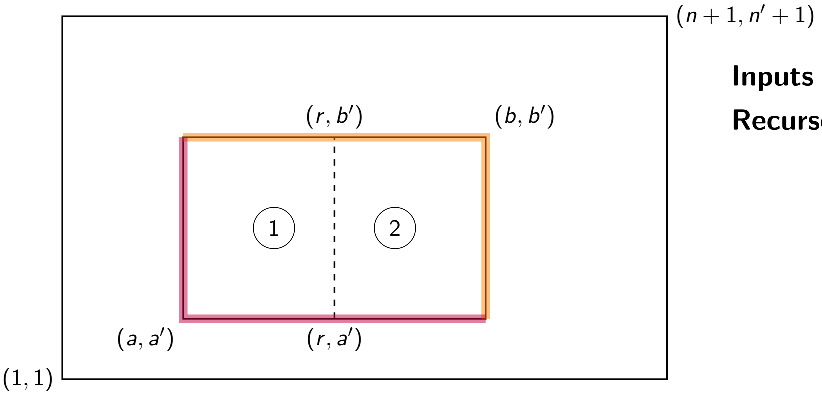
$\mathrm{sim}((x, x'), (y, y'))$ equals to the maximum achievable sum of:

1. the weight of a path from (x, x') to $(n+1, n+1)$ in the alignment graph of $\mathrm{sim}(L, L')$;
2. the weight of a path from (y, y') to $(n+1, n+1)$ in the alignment graph of $\mathrm{sim}(R, R')$; 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).

A Divide and Conquer Scheme for TED on Caterpillar Trees II



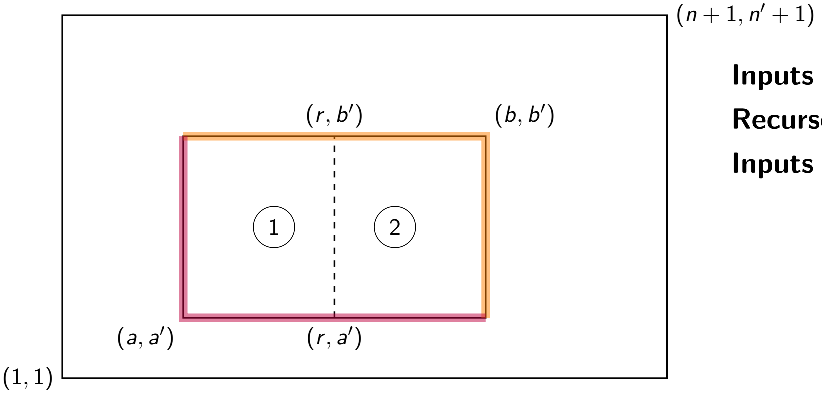
A Divide and Conquer Scheme for TED on Caterpillar Trees II



Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

A Divide and Conquer Scheme for TED on Caterpillar Trees II

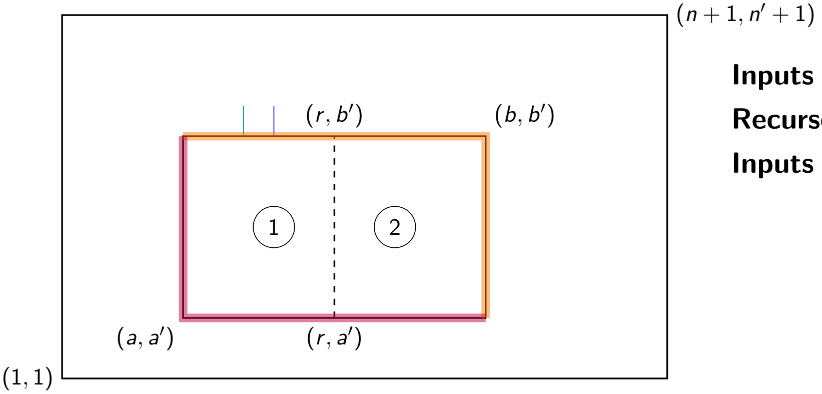


Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.

A Divide and Conquer Scheme for TED on Caterpillar Trees II

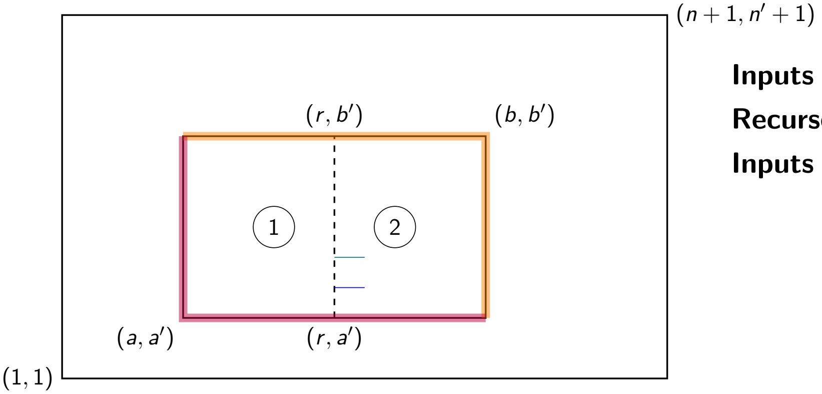


Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.

A Divide and Conquer Scheme for TED on Caterpillar Trees II

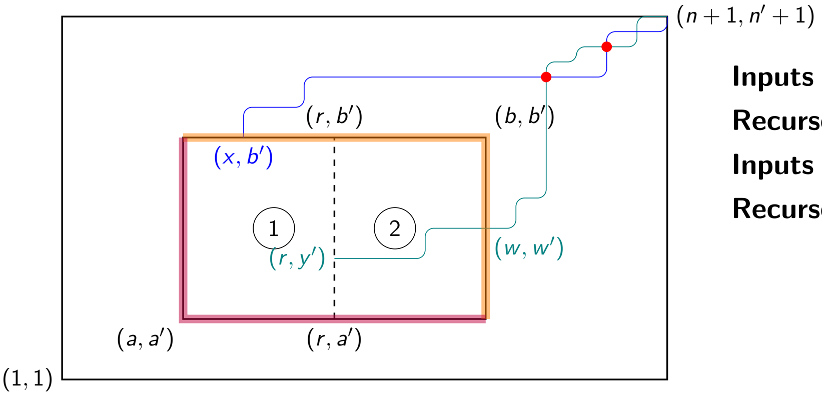


Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.

A Divide and Conquer Scheme for TED on Caterpillar Trees II



Inputs of subrectangle 2. ✔

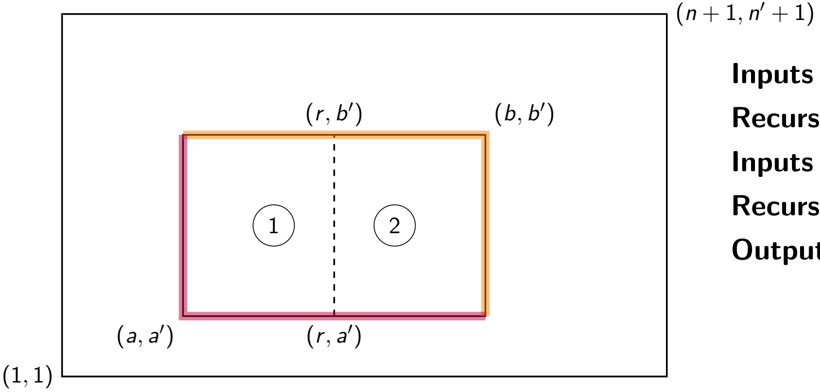
Recurse on subrectangle 2. ✔

Inputs of subrectangle 1. ✔

Recurse on subrectangle 1. ✔

$$\mathrm{sim}((x, b'), (r, y')) = \max_{(w,w')} \left\{ \mathrm{sim}(R[\, r \mathinner{..} w \,], R'[\, y' \mathinner{..} w' \,]) + \mathrm{sim}((x, b'), (w, w')) \right\}.$$

A Divide and Conquer Scheme for TED on Caterpillar Trees II



Inputs of subrectangle 2. ✔

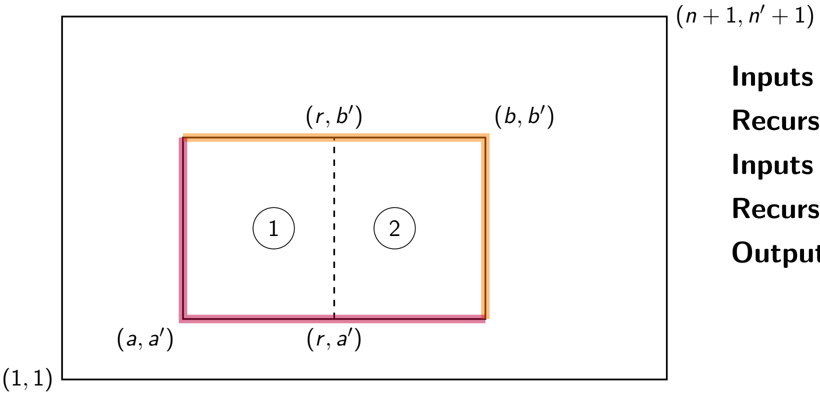
Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.✔

Recurse on subrectangle 1. ✔

Outputs of big rectangle.

A Divide and Conquer Scheme for TED on Caterpillar Trees II



Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

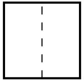
Inputs of subrectangle 1.✔

Recurse on subrectangle 1. ✔

Outputs of big rectangle. ✔

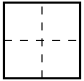
A Divide and Conquer Scheme for TED on Caterpillar Trees III

Since we can handle vertical “cuts”, we can also handle horizontal “cuts”.



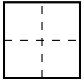
A Divide and Conquer Scheme for TED on Caterpillar Trees III

Since we can handle vertical “cuts”, we can also handle horizontal “cuts”.



A Divide and Conquer Scheme for TED on Caterpillar Trees III

Since we can handle vertical "cuts", we can also handle horizontal "cuts".

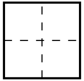


We obtain the recurrence

$$T(n) = 4T(n/2) + \mathcal{O}(T_{\text{APSP}}).$$

A Divide and Conquer Scheme for TED on Caterpillar Trees III

Since we can handle vertical “cuts”, we can also handle horizontal “cuts”.



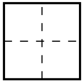
We obtain the recurrence

$$T(n) = 4T(n/2) + \mathcal{O}(T_{\text{APSP}}).$$

Thus, $T(n) = \mathcal{O}(T_{\text{APSP}})$ assuming $T_{\text{APSP}} = \mathcal{O}(n^{2+\varepsilon})$ for some $\varepsilon > 0$.

A Divide and Conquer Scheme for TED on Caterpillar Trees III

Since we can handle vertical “cuts”, we can also handle horizontal “cuts”.



We obtain the recurrence

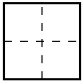
$$T(n) = 4T(n/2) + \mathcal{O}(T_{\text{APSP}}).$$

Thus, $T(n) = \mathcal{O}(T_{\text{APSP}})$ assuming $T_{\text{APSP}} = \mathcal{O}(n^{2+\varepsilon})$ for some $\varepsilon > 0$.

I cheated a bit... in the scheme I need to remember whether spines nodes are already mapped.

A Divide and Conquer Scheme for TED on Caterpillar Trees III

Since we can handle vertical “cuts”, we can also handle horizontal “cuts”.



We obtain the recurrence

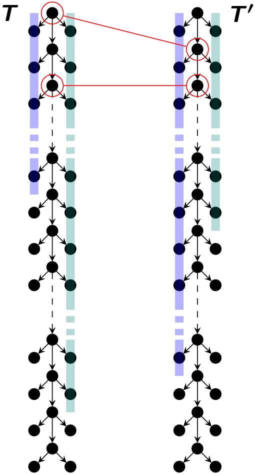
$$T(n) = 4T(n/2) + \mathcal{O}(T_{\text{APSP}}).$$

Thus, $T(n) = \mathcal{O}(T_{\text{APSP}})$ assuming $T_{\text{APSP}} = \mathcal{O}(n^{2+\varepsilon})$ for some $\varepsilon > 0$.

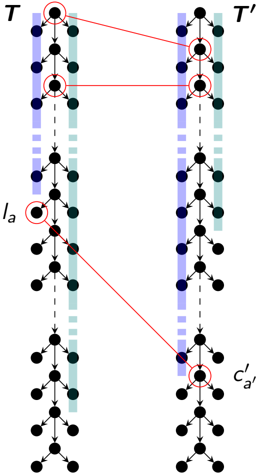
I cheated a bit... in the scheme I need to remember whether spines nodes are already mapped.

Q: How to drop the assumption that spines, left and right nodes map only nodes of the same type?

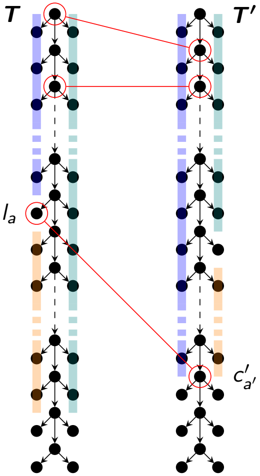
TED on Caterpillar Trees: Dropping the Assumption



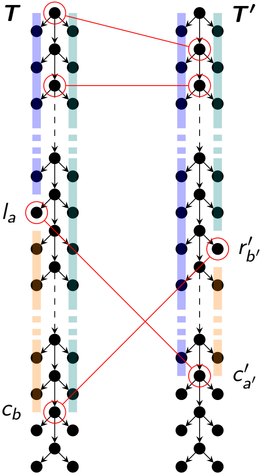
TED on Caterpillar Trees: Dropping the Assumption



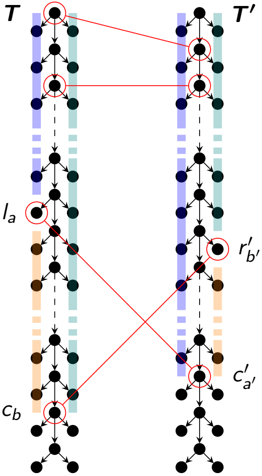
TED on Caterpillar Trees: Dropping the Assumption

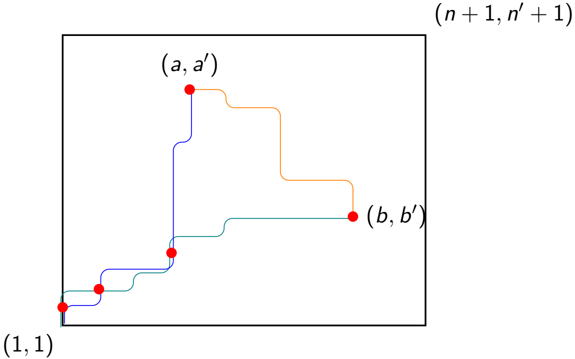


TED on Caterpillar Trees: Dropping the Assumption



TED on Caterpillar Trees: Dropping the Assumption





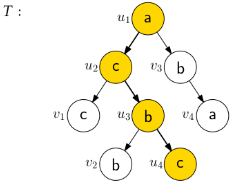
From Caterpillar Trees to Arbitrary Trees

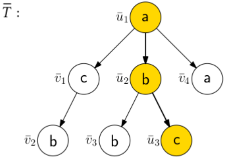
We generalize TED on caterpillar trees to Spine Edit Distance.

From Caterpillar Trees to Arbitrary 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 $\mathrm{sim}(\mathrm{sub}(v), \mathrm{sub}(v')) \;\; \forall (v, v') \in (T \times T') \setminus (S \times S')$.

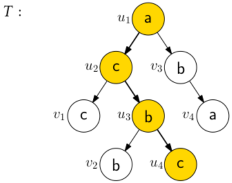


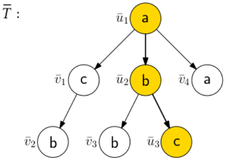


From Caterpillar Trees to Arbitrary Trees

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 $\text{sim}(\text{sub}(v), \text{sub}(v'))\ \ \forall (v, v') \in (T \times T') \setminus (S \times S')$.



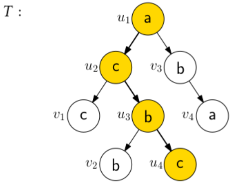


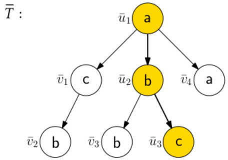
Output: $\text{sim}(\text{sub}(v), \text{sub}(v'))\ \ \forall (v, v') \in S \times S'$.

From Caterpillar Trees to Arbitrary 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 $\mathrm{sim}(\mathrm{sub}(v), \mathrm{sub}(v'))\ \ \forall (v, v') \in (T \times T') \setminus (S \times S')$.





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

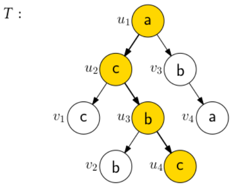
Why?

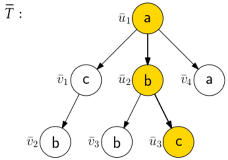
- There are still spine, left, and right nodes.
- The underlying paths are not in string alignment graphs anymore but in *forest alignment graphs*.

From Caterpillar Trees to Arbitrary 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 $\mathrm{sim}(\mathrm{sub}(v), \mathrm{sub}(v')) \;\; \forall (v, v') \in (T \times T') \setminus (S \times S')$.





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

Why?

- There are still spine, left, and right nodes.
- The underlying paths are not in string alignment graphs anymore but in *forest alignment graphs*.

We also prove that TED on arbitrary trees is fine-grained equivalent to Spine Edit Distance.

Thanks!