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 character c with c' with cost $\delta(c, c')$

$$\texttt{abcxef} \longrightarrow \texttt{abcyef}$$

(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 character c with c' with cost $\delta(c, c')$

$$\texttt{abcxef} \longrightarrow \texttt{abcyef}$$

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

$$\texttt{abcdef} \longrightarrow \texttt{abdef}$$

(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 character c with c' with cost $\delta(c, c')$

   `abcxef` $\longrightarrow$ `abcyef`

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

   `abcdef` $\longrightarrow$ `abdef`

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

   `abcef` $\longrightarrow$ `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 character c with c' with cost $\delta(c, c')$

   `abcxef` $\longrightarrow$ `abcyef`

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

   `abcdef` $\longrightarrow$ `abdef`

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

   `abcef` $\longrightarrow$ `abcxef`

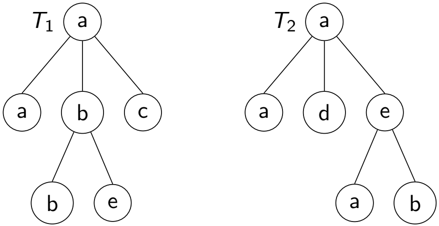
"Unweighted" (String) Edit Distance: all costs are one

Tree 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.

$\Downarrow$ Generalization on Trees

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 using deletion, insertions and substitutions.

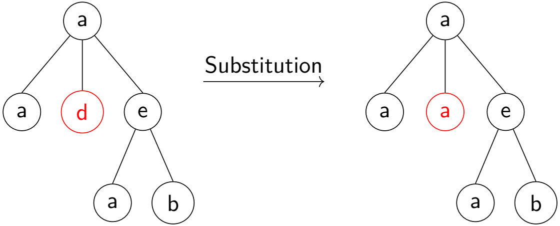


Tree Edit Distance

(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.

$\Downarrow$ Generalization on Trees

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 using deletion, insertions and substitutions.

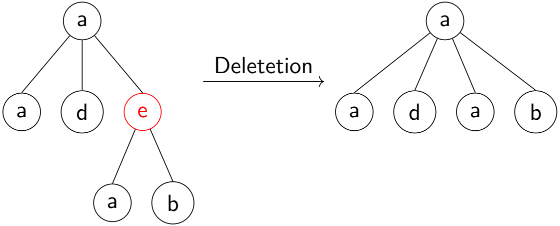


Tree Edit Distance

(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.

$\Downarrow$ Generalization on Trees

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 using deletion, insertions and substitutions.

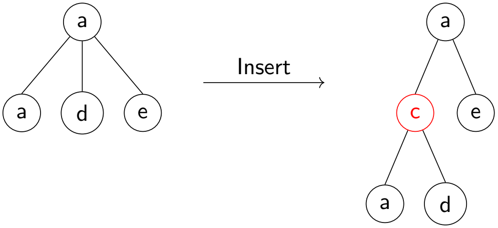


Tree Edit Distance

(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.

$\Downarrow$ Generalization on Trees

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 using deletion, insertions and substitutions.



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

Question 1: is there a $o(n^3)$, e.g. $\mathcal{O}(n^3/\log n)$, algorithm for (weighted) TED?

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 $o(n^3)$, e.g. $\mathcal{O}(n^3/\log n)$, 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

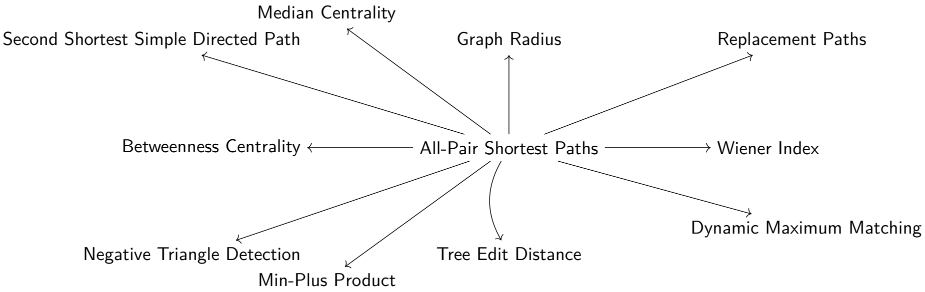
Tree Edit Distance

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 $\varepsilon > 0$.

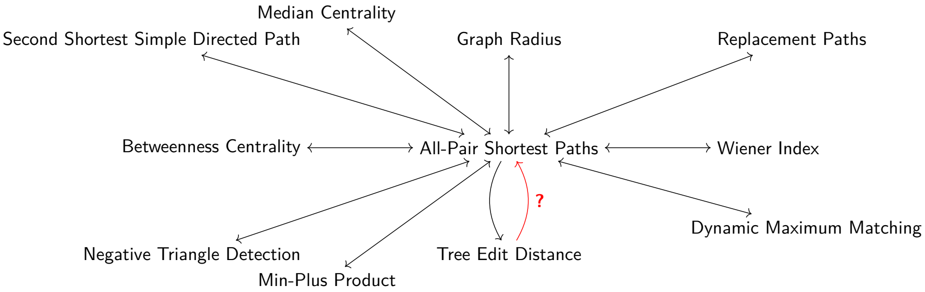


The Fine-grained Complexity of Tree Edit Distance

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 $\varepsilon > 0$.

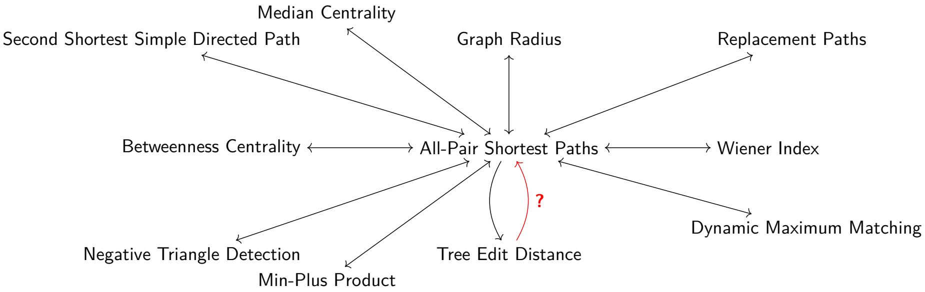


The Fine-grained Complexity of Tree Edit Distance

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 $\varepsilon > 0$.

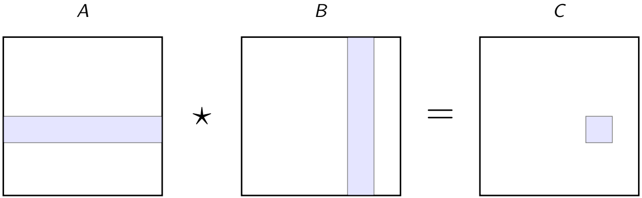


Question 2: is TED equivalent to APSP?

Reducing to Min-Plus Product

Can we reduce TED to computing min-plus product?

Min-plus Product (APSP equivalent)

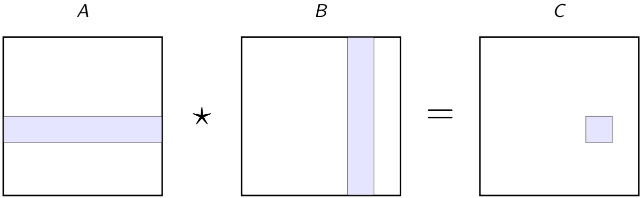


$$C_{i,j} = \min_{1 \le k \le n}\{A_{i,k} + B_{k,j}\}$$

Reducing to Min-Plus Product

Can we reduce TED to computing min-plus product?

Monotone Min-plus Product (Solvable in $\mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ time)



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

- **B is row monotone**:

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

- **A, B are bounded**:

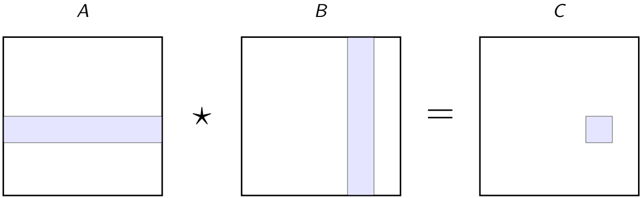
$$\forall i,j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

Reducing to Min-Plus Product

Can we reduce unweighted TED to computing monotone min-plus product?

Monotone Min-plus Product

(Solvable in $\mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ time)



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

- **B is row monotone**:

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

- **A, B are bounded**:

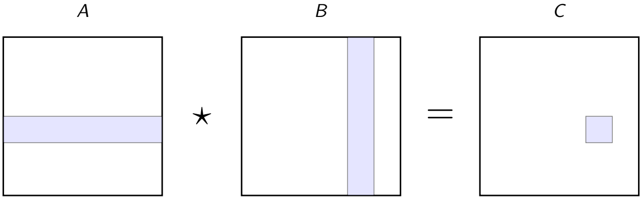
$$\forall i, j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

Reducing to Min-Plus Product

Can we reduce unweighted TED to computing monotone min-plus product?

Yes! ✔

Monotone Min-plus Product (Solvable in $\mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ time)



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

- ***B* is row monotone**:

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

- ***A*, *B* are bounded**:

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

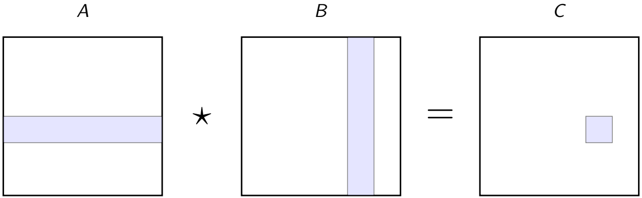
Reducing to Min-Plus Product

Can we reduce unweighted TED to computing monotone min-plus product?

Yes! ✔ But... existing reductions by Mao and Dürr:

- use observations that only apply to the unweighted case
- are not tight, yielding a $\mathcal{O}(n^{2.9148})$-time algorithm

Monotone Min-plus Product (Solvable in $\mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ time)



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

- **B is row monotone**:

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

- **A, B are bounded**:

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

Reducing to Min-Plus Product

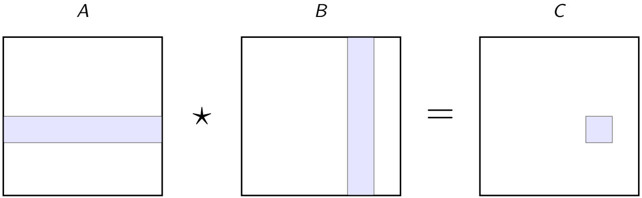
Can we reduce unweighted TED to computing monotone min-plus product?

Yes! ✔ But... existing reductions by Mao and Dürr:

- use observations that only apply to the unweighted case
- are not tight, yielding a $\mathcal{O}(n^{2.9148})$-time algorithm

Question 3: is there tight reduction from unweighted TED to monotone min-plus product?

Monotone Min-plus Product (Solvable in $\mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ time)



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

- ***B* is row monotone**:

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

- ***A*, *B* are bounded**:

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

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 $\mathcal{O}(T_{\mathsf{APSP}}(n) + n^{2+o(1)})$.

Question 2: is TED equivalent to APSP? ✔

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

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 $n^3/2^{\Omega(\sqrt{\log n})}$.

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? ✔

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: $T_{\text{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2})$.

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: $T_{\mathrm{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2})$.

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**	$\boldsymbol{\mathcal{O}(n^{2.687})}$

How to visualize TED to come up with the reduction

Visualization I: Edit Distance as a Matching

We can visualize the least-cost transformation as a matching.

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

`o` is substituted with `r`

`c h r i s t m a s`

Visualization I: Edit Distance as a Matching

We can visualize the least-cost transformation as a matching.

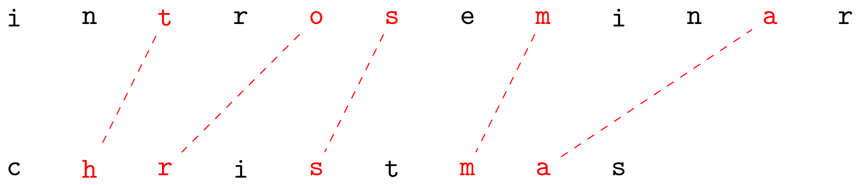
`i n t r o s e m i n a r`

`m` is left untouched

`c h r i s t m a s`

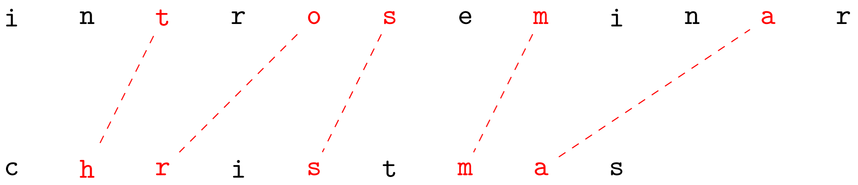
Visualization I: Edit Distance as a Matching

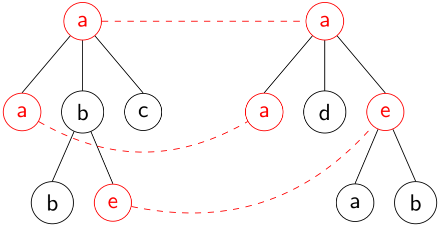
We can visualize the least-cost transformation as a matching.



Visualization I: Edit Distance as a Matching

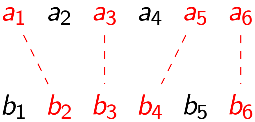
We can visualize the least-cost transformation as a matching.

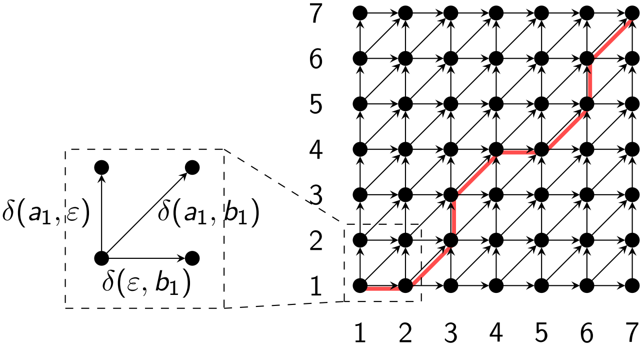




Visualization II: Edit Distance as a Path

The *string alignment graph* summarizes the DP scheme computing the edit distance.

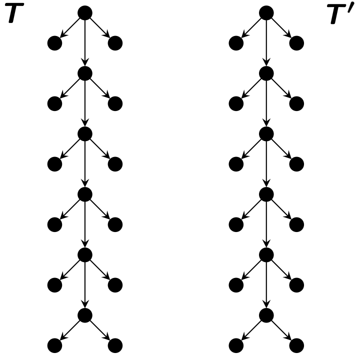




Note: All border-to-border distances can be computed in $\mathcal{O}(n^2)$ time

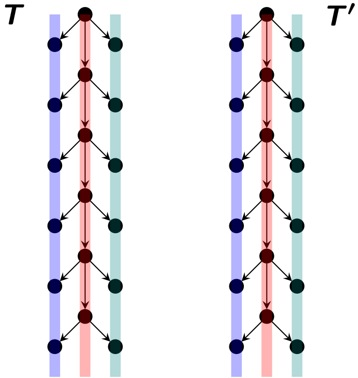
TED on Caterpillar Trees I

Let us start by visualizing the tree edit distance between two caterpillar trees...



TED on Caterpillar Trees I

Let us start by visualizing the tree edit distance between two caterpillar trees...



spine nodes

left nodes

right nodes

TED on Caterpillar Trees I

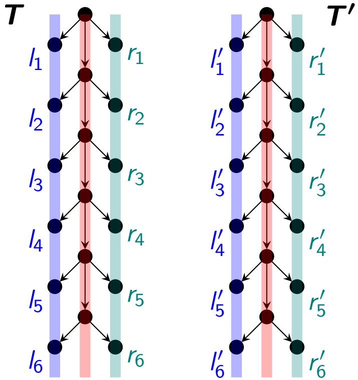
Let us start by visualizing the tree edit distance between two caterpillar trees...

$L = l_1 l_2 l_3 l_4 l_5 l_6$

$R = r_1 r_2 r_3 r_4 r_5 r_6$

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

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



spine nodes

left nodes

right nodes

TED on Caterpillar Trees I

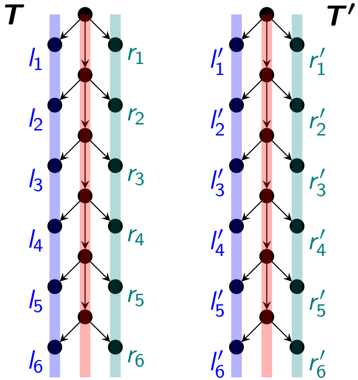
Let us start by visualizing the tree edit distance between two caterpillar trees...

$L = l_1 l_2 l_3 l_4 l_5 l_6$

$R = r_1 r_2 r_3 r_4 r_5 r_6$

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

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



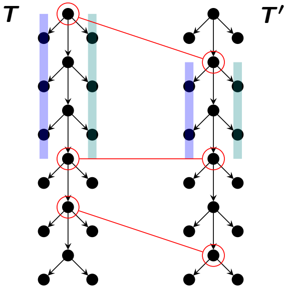
spine nodes

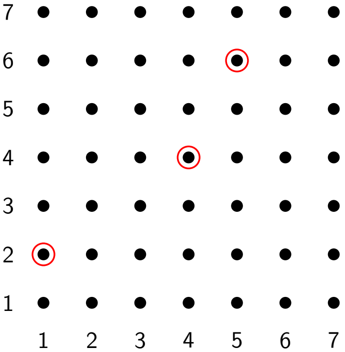
left nodes

right nodes

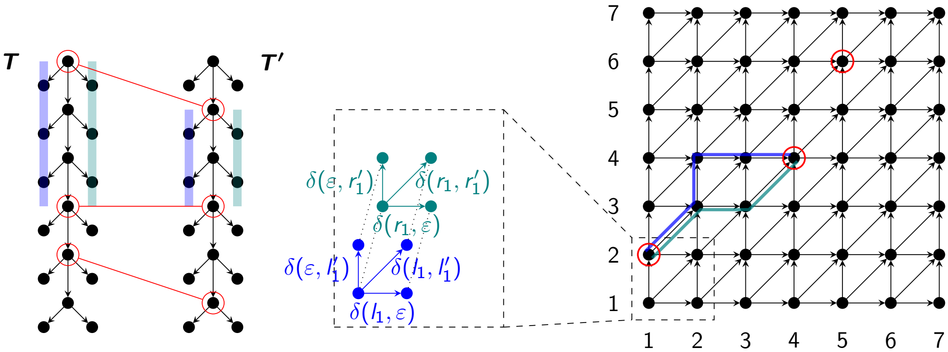
...with the assumption that spine, left and right nodes of T only match with nodes of their same type (color) in T', respectively.

TED on Caterpillar Trees II

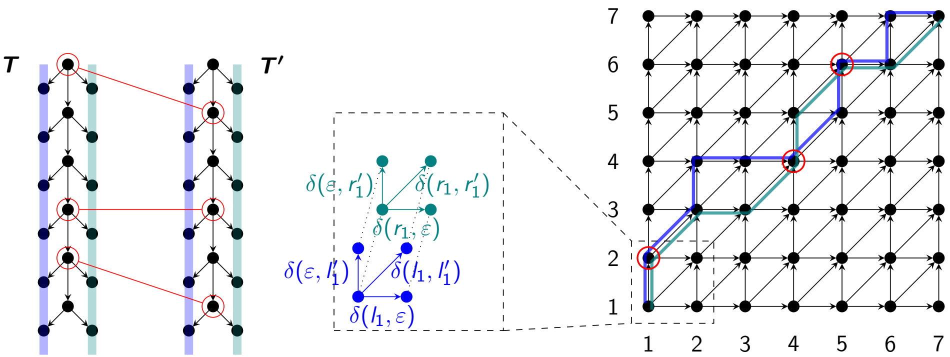




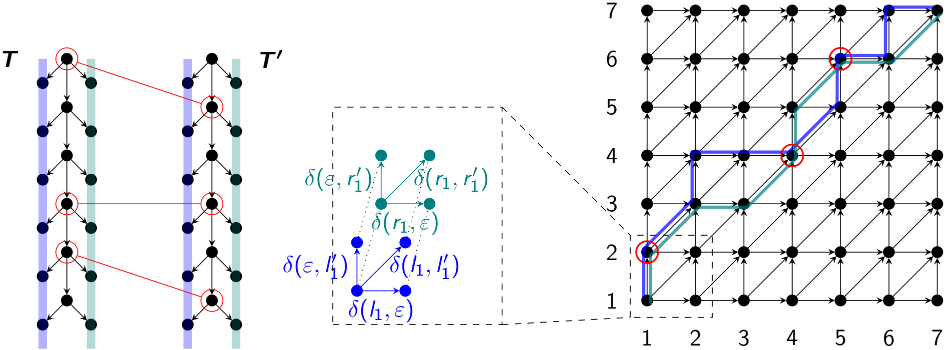
TED on Caterpillar Trees II



TED on Caterpillar Trees II



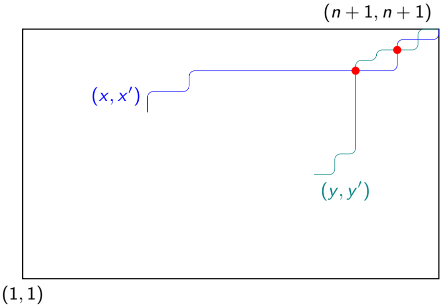
TED on Caterpillar Trees II



ted(T, T') is a pair of **interlaced paths** that minimize three terms:

1. corner-to-corner path in the alignment graph of ed(L, L');
2. corner-to-corner path in the alignment graph of ed(R, R'); and
3. a set of spine-to-spine matchings where the two paths intersect.

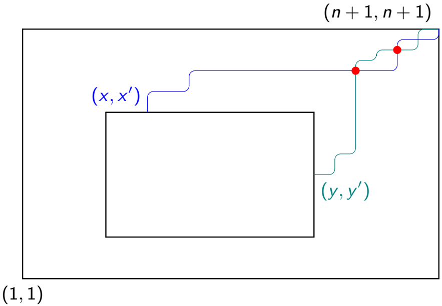
A Divide and Conquer Scheme for TED on Caterpillar Trees



val$((x, x'), (y, y'))$ is a pair of **interlaced paths** that minimize three terms:

1. (x, x')-to-corner path in the alignment graph of ed(L, L');
2. (y, y')-to-corner in the alignment graph of ed(R, R'); and
3. a set of spine-to-spine matchings where the two paths intersect.

A Divide and Conquer Scheme for TED on Caterpillar Trees



Divide and Conquer Scheme

Input:

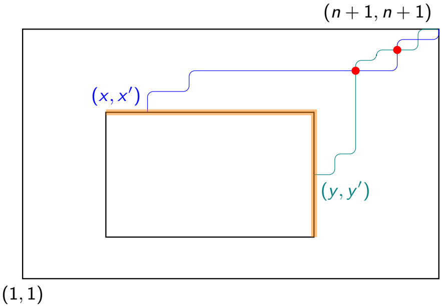
- A rectangle in the grid.

Output:

val$((x, x'), (y, y'))$ is a pair of **interlaced paths** that minimize three terms:

1. (x, x')-to-corner path in the alignment graph of ed(L, L');
2. (y, y')-to-corner in the alignment graph of ed(R, R'); and
3. a set of spine-to-spine matchings where the two paths intersect.

A Divide and Conquer Scheme for TED on Caterpillar Trees



Divide and Conquer Scheme

Input:

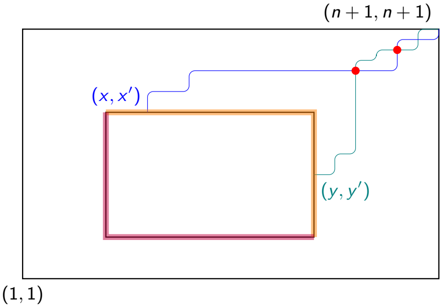
- A rectangle in the grid.
- $\mathrm{val}((x, x'), (y, y'))$ w/ $(x, x'), (y, y')$ on upper-right border.

Output:

$\mathrm{val}((x, x'), (y, y'))$ is a pair of **interlaced paths** that minimize three terms:

1. (x, x')-to-corner path in the alignment graph of $\mathrm{ed}(L, L')$;
2. (y, y')-to-corner in the alignment graph of $\mathrm{ed}(R, R')$; and
3. a set of spine-to-spine matchings where the two paths intersect.

A Divide and Conquer Scheme for TED on Caterpillar Trees



Divide and Conquer Scheme

Input:

- A rectangle in the grid.
- $\mathrm{val}((x, x'), (y, y'))$ w/ $(x, x'), (y, y')$ on upper-right border.

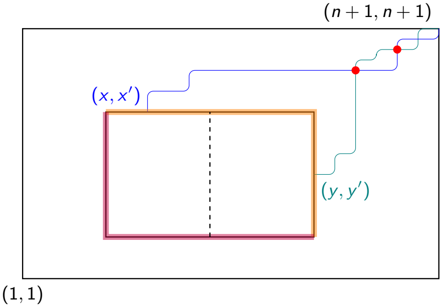
Output:

- $\mathrm{val}((x, x'), (y, y'))$ w/ $(x, x'), (y, y')$ on lower-left border.

$\mathrm{val}((x, x'), (y, y'))$ is a pair of **interlaced paths** that minimize three terms:

1. (x, x')-to-corner path in the alignment graph of $\mathrm{ed}(L, L')$;
2. (y, y')-to-corner in the alignment graph of $\mathrm{ed}(R, R')$; and
3. a set of spine-to-spine matchings where the two paths intersect.

A Divide and Conquer Scheme for TED on Caterpillar Trees



Divide and Conquer Scheme

Input:

- A rectangle in the grid.
- $\mathrm{val}((x,x'),(y,y'))$ w/ $(x,x'),(y,y')$ on upper-right border.

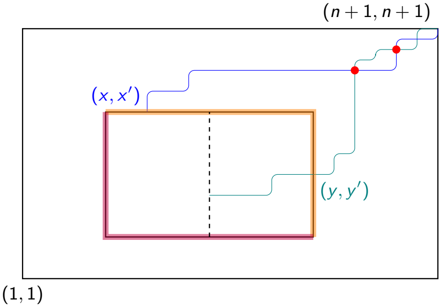
Output:

- $\mathrm{val}((x,x'),(y,y'))$ w/ $(x,x'),(y,y')$ on lower-left border.

$\mathrm{val}((x,x'),(y,y'))$ is a pair of **interlaced paths** that minimize three terms:

1. (x,x')-to-corner path in the alignment graph of $\mathrm{ed}(L,L')$;
2. (y,y')-to-corner in the alignment graph of $\mathrm{ed}(R,R')$; and
3. a set of spine-to-spine matchings where the two paths intersect.

A Divide and Conquer Scheme for TED on Caterpillar Trees



Divide and Conquer Scheme

Input:

- A rectangle in the grid.
- $\mathrm{val}((x, x'), (y, y'))$ w/ $(x, x'), (y, y')$ on upper-right border.

Output:

- $\mathrm{val}((x, x'), (y, y'))$ w/ $(x, x'), (y, y')$ on lower-left border.

$\mathrm{val}((x, x'), (y, y'))$ is a pair of **interlaced paths** that minimize three terms:

1. (x, x')-to-corner path in the alignment graph of $\mathrm{ed}(L, L')$;
2. (y, y')-to-corner in the alignment graph of $\mathrm{ed}(R, R')$; and
3. a set of spine-to-spine matchings where the two paths intersect.

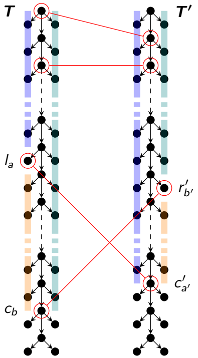
Extending to General Trees

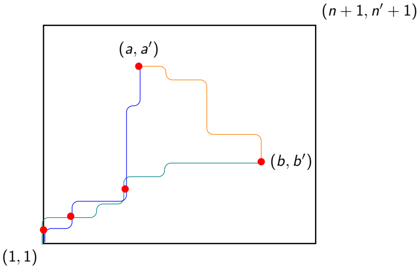
Steps needed to extend to general trees:

Extending to General Trees

Steps needed to extend to general trees:

1. Drop the assumption on caterpillars that left matches w/ left, right w/ right and spine w/ spine.





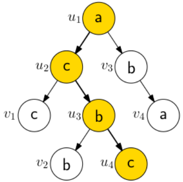
Extending to General Trees

Steps needed to extend to general trees:

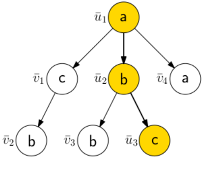
1. Drop the assumption on caterpillars that left matches w/ left, right w/ right and spine w/ spine.
2. Generalize TED on caterpillar to spine edit distance on general trees.

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

T :



$\bar{T}$:



Images from **[BGHS19]**

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

Extending to General Trees

Steps needed to extend to general trees:

1. Drop the assumption on caterpillars that left matches w/ left, right w/ right and spine w/ spine.
2. Generalize TED on caterpillar to spine edit distance on general trees.
3. Devise algorithm computing border-to-border distances in forest alignment graphs in APSP time.

Thanks!