Faster Tree Edit Distance via APSP Equivalence

DIMACS Workshop on Fine-Grained Complexity of String Problems

Jakob Nogler[4]

Based on joint work with:

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

[1]ETH Zurich

[2]Bocconi University

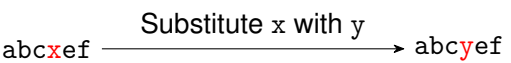
[3]UC San Diego

[4]MIT

(String) Edit Distance 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.

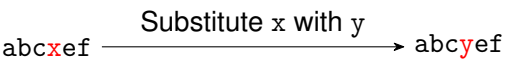
(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

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

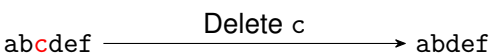


(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

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

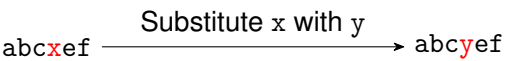


2. Delete a character c with cost $\delta(c, \varepsilon)$

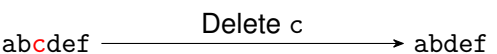


(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

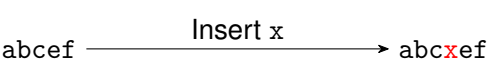
1. Substitute a character c with c' with cost $\delta(c, c')$



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

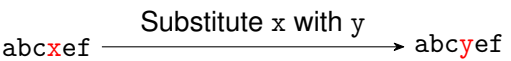


3. Insert a character c with cost $\delta(\varepsilon, c)$

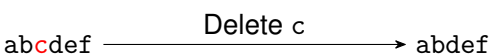


(String) Edit Distance Problem
Input: Two strings S_1, S_2 and a cost function δ.
Output: Cheapest transformation of S_1 into S_2 using deletion, insertions and substitutions.

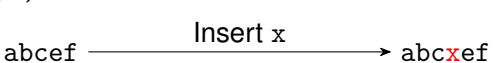
1. Substitute a character c with c' with cost $\delta(c, c')$



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



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



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

Algorithms for (String) Edit Distance

References	Time	Remarks
Vin68, NW70, Sel74, WF74	$\mathcal{O}(n^2)$	textbook algorithm
MP80	$\mathcal{O}(n^2/\log^2 n)$	small alphabets and integer weights only
BF05	$\mathcal{O}(n^2 \log\log n/\log^2 n)$	small integer weights only

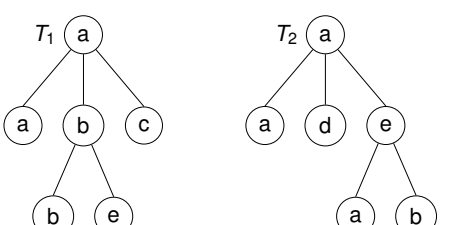
Algorithms for (String) Edit Distance

References	Time	Remarks
Vin68, NW70, Sel74, WF74	$\mathcal{O}(n^2)$	textbook algorithm
MP80	$\mathcal{O}(n^2/\log^2 n)$	small alphabets and integer weights only
BF05	$\mathcal{O}(n^2 \log\log n/\log^2 n)$	small integer weights only
BI18	$n^{2-o(1)}$	Lower bound under OVH/SETH, already for unweighted

(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

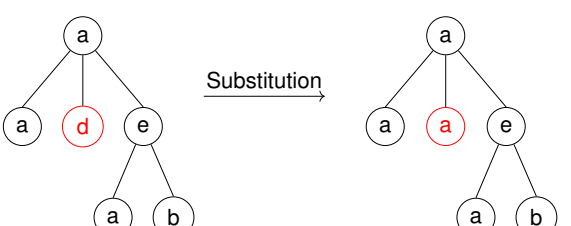
Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2 using deletion, insertions and substitutions.



(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

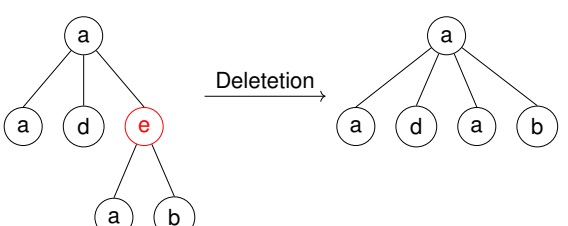
Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2 using deletion, insertions and substitutions.



(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

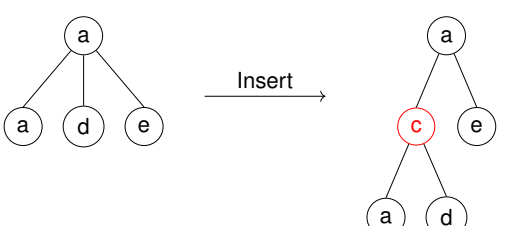
Tree Edit Distance Problem (TED)
Input: Two rooted, labeled, left-to-right-ordered trees T_1, T_2 and a cost function δ.
Output: Cheapest transformation of T_1 into T_2 using deletion, insertions and substitutions.



(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.

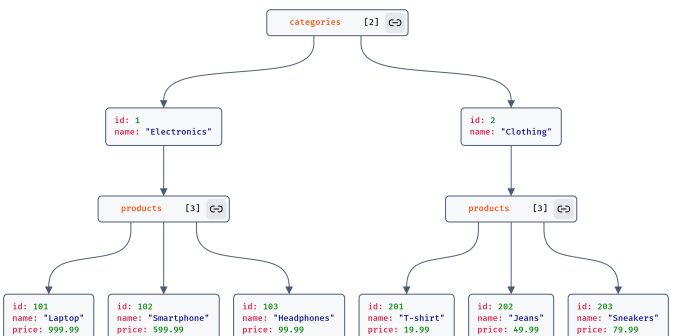


Background

- Introduced by Selkow in the late 1970s.

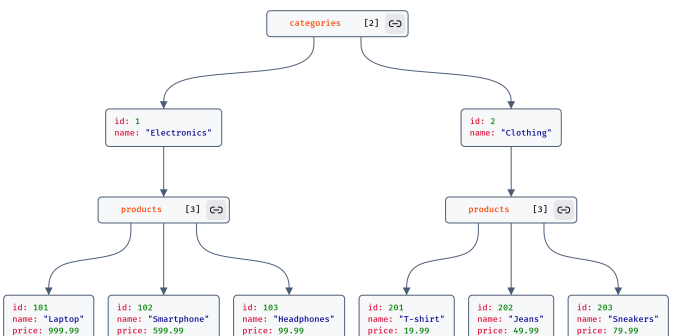
BACKGROUND

- Introduced by Selkow in the late 1970s.
- Useful to compate hyerachical data.



Background

- Introduced by Selkow in the late 1970s.
- Useful to compate hyerachical data.



- Applications in computational biology, structured data analysis, image processing, compiler optimization, and more.

Algorithms for Tree Edit Distance

Algorithms for Tree Edit Distance

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

Algorithms for Tree Edit Distance

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

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

Red rows: algorithms fall within a framework for which there is a $\Omega(n^3)$ lower bound.

Algorithms for Tree Edit Distance

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	$n^{3-o(1)}$ lower bound under the APSP conjecture

Red rows: algorithms fall within a framework for which there is a $\Omega(n^3)$ lower bound.

Algorithms for Tree Edit Distance

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	$n^{3-o(1)}$ lower bound under the APSP conjecture
2022	Mao	unweighted	$\mathcal{O}(n^{2.9546})$
2023	Dürr	unweighted	$\mathcal{O}(n^{2.9148})$

Red rows: algorithms fall within a framework for which there is a $\Omega(n^3)$ lower bound.

Algorithms for Tree Edit Distance

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	$n^{3-o(1)}$ lower bound under the APSP conjecture
2022	Mao	unweighted	$\mathcal{O}(n^{2.9546})$
2023	Dürr	unweighted	$\mathcal{O}(n^{2.9148})$

Red rows: algorithms fall within a framework for which there is a $\Omega(n^3)$ lower bound.

Algorithms for Tree Edit Distance

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	$n^{3-o(1)}$ lower bound under the APSP conjecture
2022	Mao	unweighted	$\mathcal{O}(n^{2.9546})$
2023	Dürr	unweighted	$\mathcal{O}(n^{2.9148})$

Red rows: algorithms fall within a framework for which there is a $\Omega(n^3)$ lower bound.

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 on $\mathcal{O}(\log n)$-bit weights running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

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 on $\mathcal{O}(\log n)$-bit weights running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

(under the APSP conjecture, there is no $\mathcal{O}(n^{3-\varepsilon})$ algorithm for weighted TED for any $\varepsilon > 0$.)

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 on $\mathcal{O}(\log n)$-bit weights running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

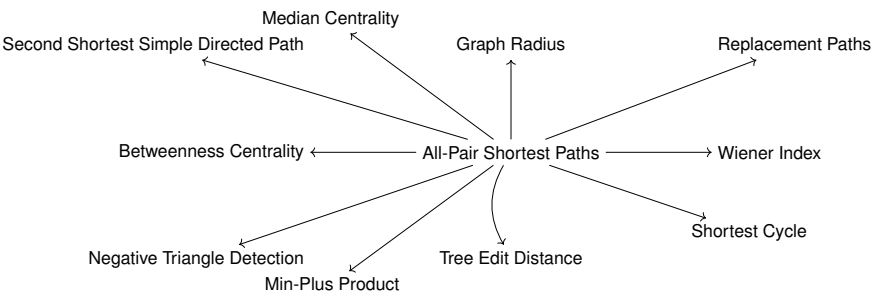
(under the APSP conjecture, there is no $\mathcal{O}(n^{3-\varepsilon})$ algorithm for weighted TED for any $\varepsilon > 0$.)

Question 1: Are subpolynomial improvements for weighted TED possible, e.g. $\mathcal{O}(n^3/\log n)$?

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 on $\mathcal{O}(\log n)$-bit weights running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

(under the APSP conjecture, there is no $\mathcal{O}(n^{3-\varepsilon})$ algorithm for weighted TED for any $\varepsilon > 0$.)

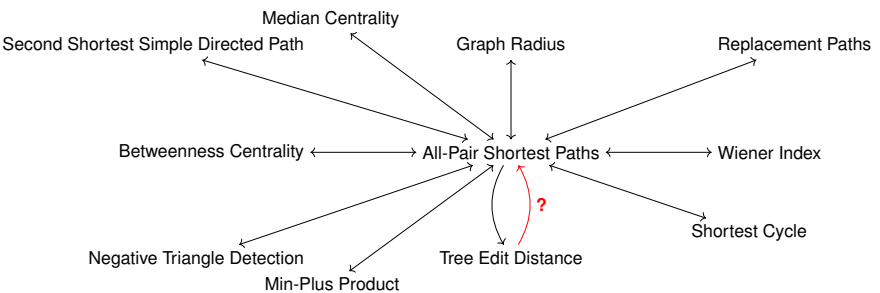


Question 1: Are subpolynomial improvements for weighted TED possible, e.g. $\mathcal{O}(n^3/\log n)$?

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 on $\mathcal{O}(\log n)$-bit weights running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

(under the APSP conjecture, there is no $\mathcal{O}(n^{3-\varepsilon})$ algorithm for weighted TED for any $\varepsilon > 0$.)

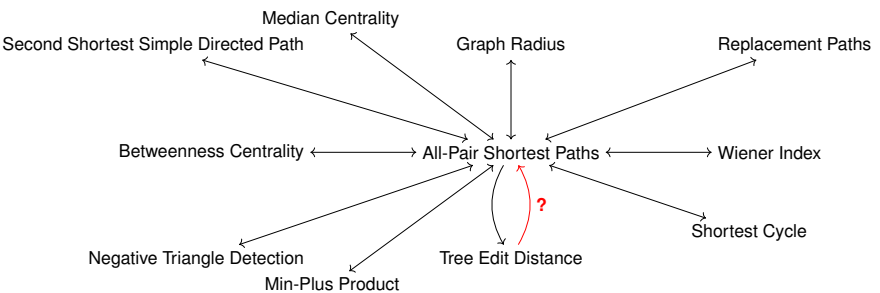


Question 1: Are subpolynomial improvements for weighted TED possible, e.g. $\mathcal{O}(n^3/\log n)$?

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 on $\mathcal{O}(\log n)$-bit weights running in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

(under the APSP conjecture, there is no $\mathcal{O}(n^{3-\varepsilon})$ algorithm for weighted TED for any $\varepsilon > 0$.)

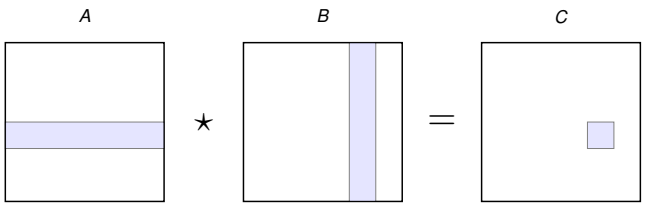


Question 1: Are subpolynomial improvements for weighted TED possible, e.g. $\mathcal{O}(n^3/\log n)$?

Question 2: is weighted TED (subcubically) equivalent to APSP?

What allows us design truly subsubic algorithms for unweighted TED?

Min-plus Product

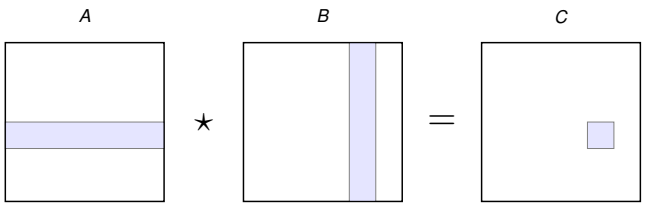


$$C_{i,j} = \min_{1 \le k \le n}\{A_{i,k} + B_{k,j}\}$$

Min-Plus Product Runtime: $\mathsf{T}_{MUL} = \Theta(\mathsf{T}_{APSP})$

What allows us design truly subsubic 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**:

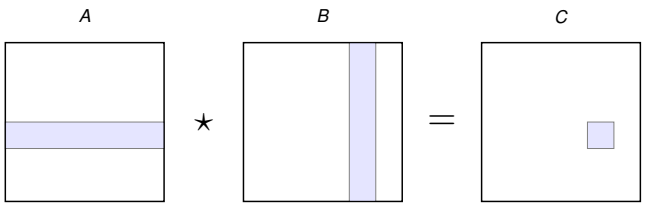
$$\forall i, j \quad A_{i,j}, B_{i,j} = \mathcal{O}(n).$$

Min-Plus Product Runtime: $\mathsf{T}_{MUL} = \Theta(\mathsf{T}_{APSP})$

Monotone Min-Plus Product Runtime: $T_{\mathsf{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ [Chi, Duan, Xie, Zhang '22]

What allows us design truly subsubic 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**:

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

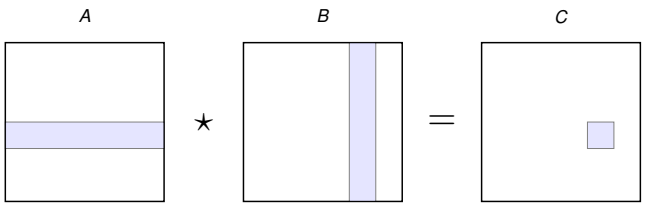
Min-Plus Product Runtime: $\mathsf{T}_{MUL} = \Theta(\mathsf{T}_{APSP})$

Monotone Min-Plus Product Runtime: $T_{\mathsf{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ [Chi, Duan, Xie, Zhang '22]

Subcubic algorithm for unweighted TED of Mao & Dürr are non-tight reductions to Monotone Min-plus Product.

What allows us design truly subsubic 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**:

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

Min-Plus Product Runtime: $\mathsf{T}_{MUL} = \Theta(\mathsf{T}_{APSP})$

Monotone Min-Plus Product Runtime: $T_{\mathsf{MonMUL}} = \mathcal{O}(n^{(\omega+3)/2}) = \mathcal{O}(n^{2.687})$ [Chi, Duan, Xie, Zhang '22]

Subcubic algorithm for unweighted TED of Mao & Dürr are non-tight reductions to Monotone Min-plus Product.

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

Theorem 1

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

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? ✔

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

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 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 $\mathcal{O}(T_{\text{MonMUL}}(n) + n^{2+o(1)}) = \mathcal{O}(n^{(\omega+3)/2})$.

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 $\mathcal{O}(T_{\text{MonMUL}}(n) + n^{2+o(1)}) = \mathcal{O}(n^{(\omega+3)/2})$.

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

Goal: Compute the TED of two trees T and T' via an APSP algorithm.

Sketch of the Reduction

Goal: Compute the TED of two trees T and T' via a Min-Plus Product algorithm.

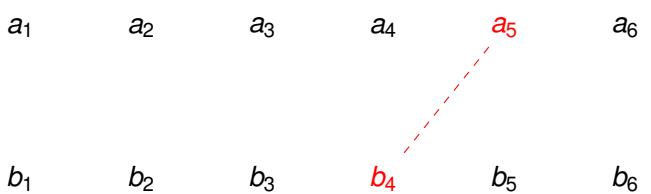
Sketch of the Reduction

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Sketch of the Reduction

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

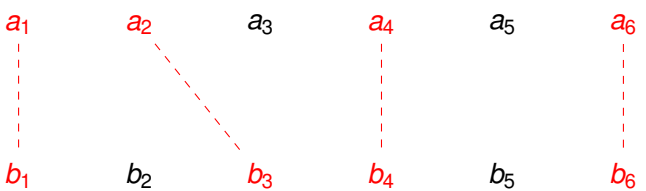
Given two strings $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_n$, we define their similarity as:



1. $\eta(a_i, b_j) := \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ *How much I save by substituting a_i with b_j*

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two strings $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_n$, we define their similarity as:

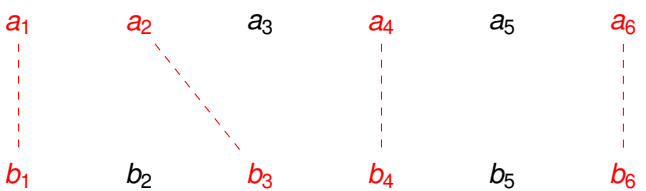


1. $\eta(a_i, b_j) := \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ *How much I save by substituting a_i with b_j*

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

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two strings $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_n$, we define their similarity as:



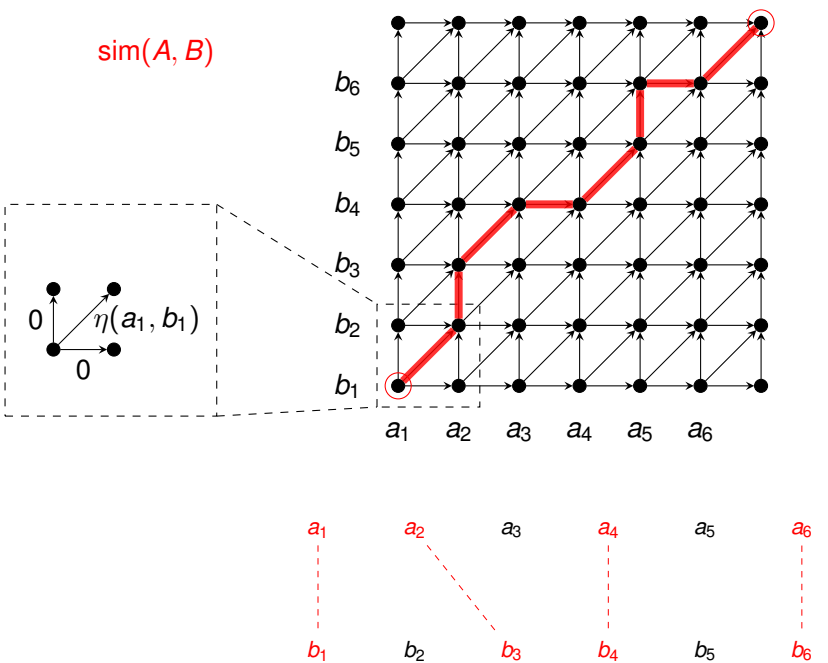
1. $\eta(a_i, b_j) := \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ *How much I save by substituting a_i with b_j*

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

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

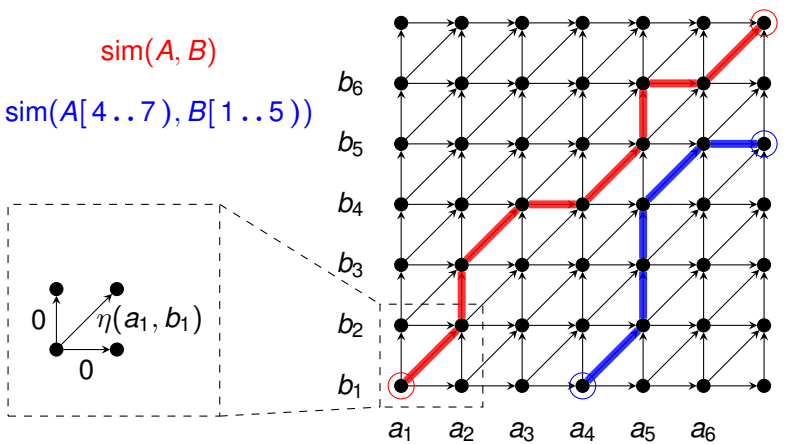
Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

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



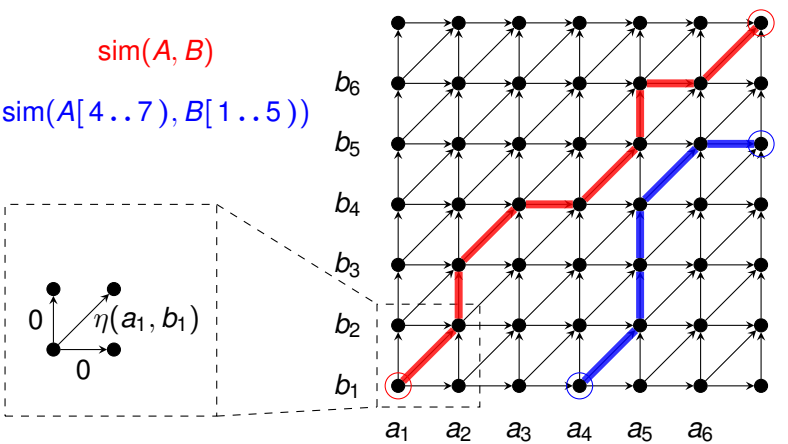
Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

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



Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

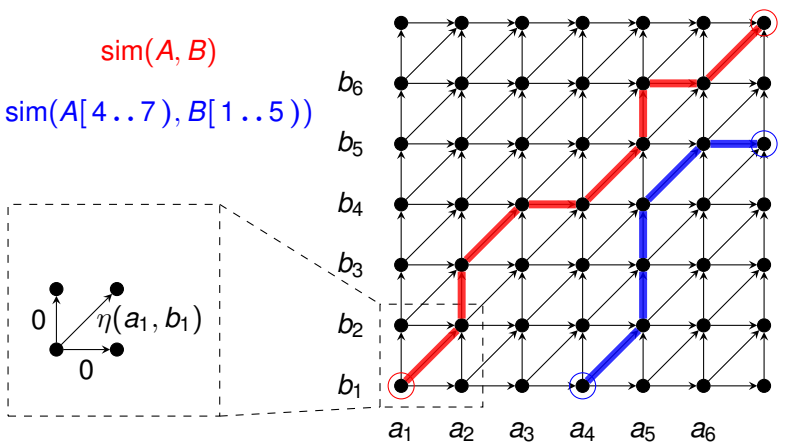
The *string alignment graph* summarizes the DP scheme computing the similarity.



lower-to-right border:
suffix vs prefix

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

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



lower-to-right border:
suffix vs prefix

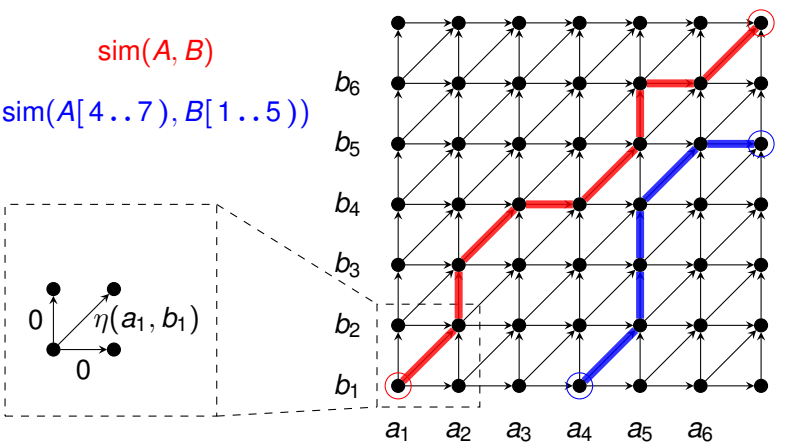
lower-to-upper border:
infix vs whole

left-to-right border:
whole vs infix

lower-to-right border:
prefix vs suffix

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

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

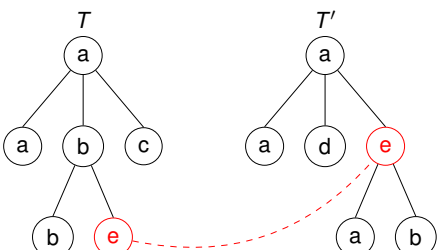
left-to-right border:
whole vs infix

lower-to-right border:
prefix vs suffix

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

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

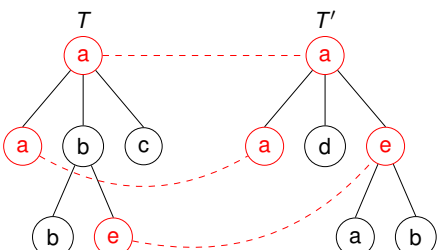
Given two trees T and T', we define their similarity as:



1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v'*

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

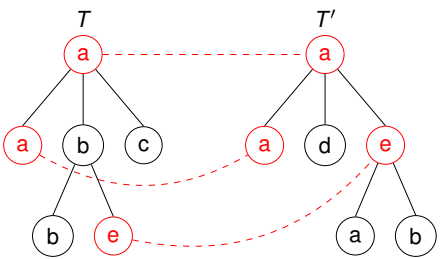
Given two trees T and T', we define their similarity as:



1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v′*
2. $\mathrm{sim}(T, T') =$"maximum weight of similarity matching" *Maximum I can save*

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two trees T and T', we define their similarity as:



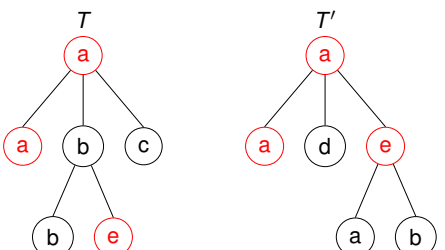
1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v'*

2. $\mathrm{sim}(T, T') =$"maximum weight of similarity matching" *Maximum I can save*

3. 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'.

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two trees T and T', we define their similarity as:



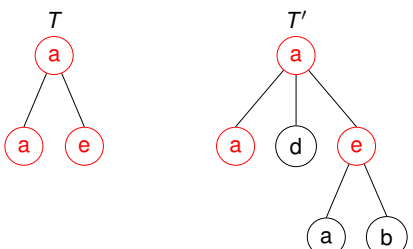
1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v'*

2. $\mathrm{sim}(T, T') =$"maximum weight of similarity matching" *Maximum I can save*

3. 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'.

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two trees T and T', we define their similarity as:



1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v'*

2. $\mathrm{sim}(T, T') =$"maximum weight of similarity matching" *Maximum I can save*

3. 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'.

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two trees T and T', we define their similarity as:

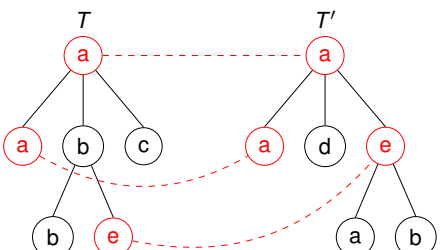
1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v'*

2. $\mathrm{sim}(T, T') =$"maximum weight of similarity matching" *Maximum I can save*

3. 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'.

Goal: Compute the similarity of two trees T and T' via a Min-Plus Product algorithm.

Given two trees T and T', we define their similarity as:



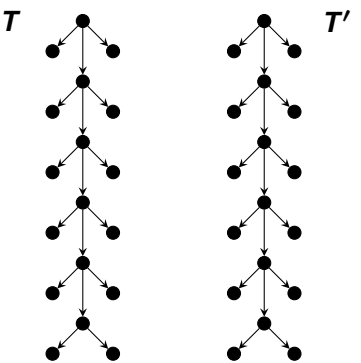
1. $\eta(v, v') := \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$ *How much I save by substituting v with v′*

2. $\mathrm{sim}(T, T')$ =“maximum weight of similarity matching” *Maximum I can save*

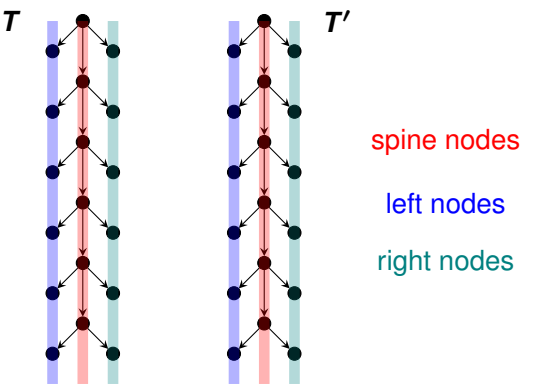
3. Condition on similarity matching: for any two matched vertices (v, v') and (u, u')
   - v is an ancestor of u in T if and only if v' is an ancestor of u' in T',
   - v comes before u in the pre-order of T if and only if v' comes before u' in the pre-order of T'.

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

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm.



Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm.



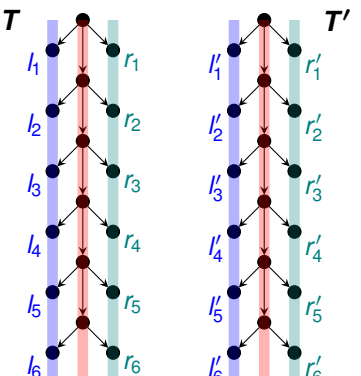
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm.

$L = l_1 l_2 l_3 l_4 l_5 l_6$

$R = r_1 r_2 r_3 r_4 r_5 r_6$

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

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



spine nodes

left nodes

right nodes

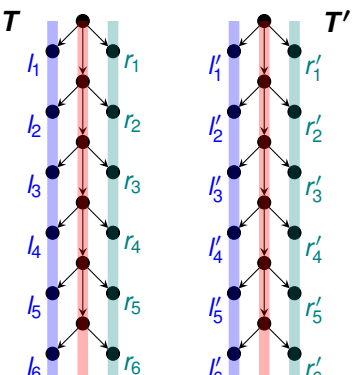
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

$L = l_1 l_2 l_3 l_4 l_5 l_6$

$R = r_1 r_2 r_3 r_4 r_5 r_6$

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

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



spine nodes

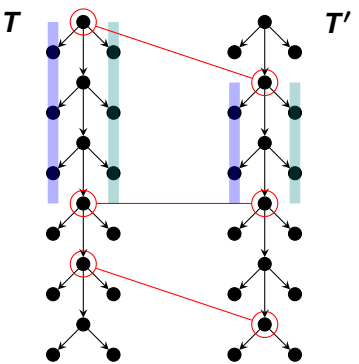
left nodes

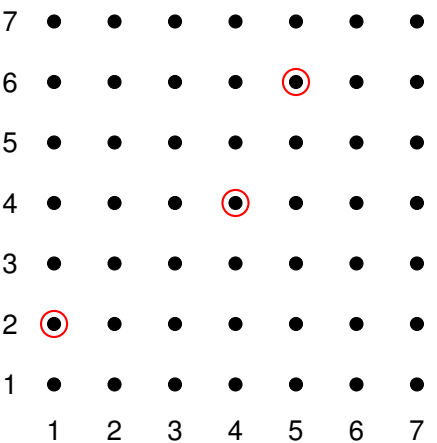
right nodes

Left-to-Left/Spine-to-spine/Right-to-Right restriction:

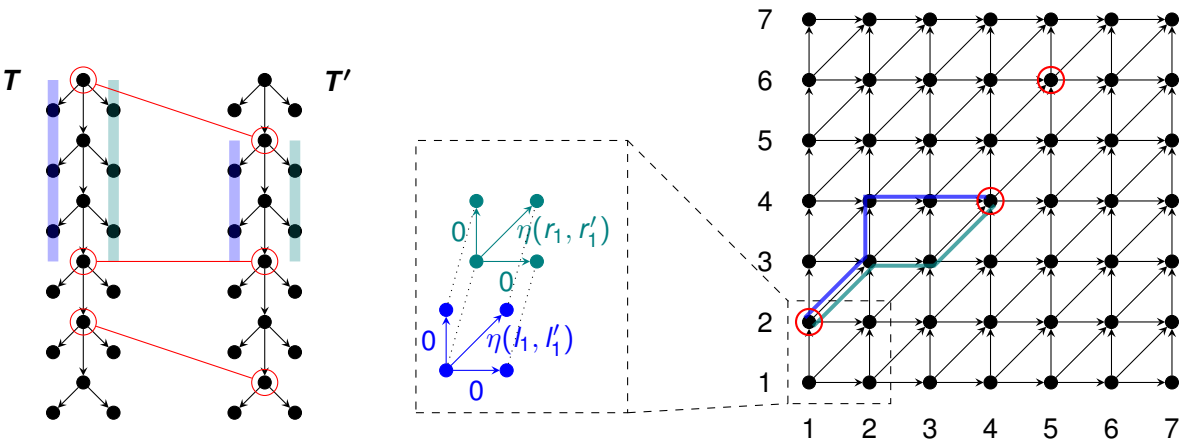
spine, left and right nodes of T only match with nodes of their same type (color) in T'.

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

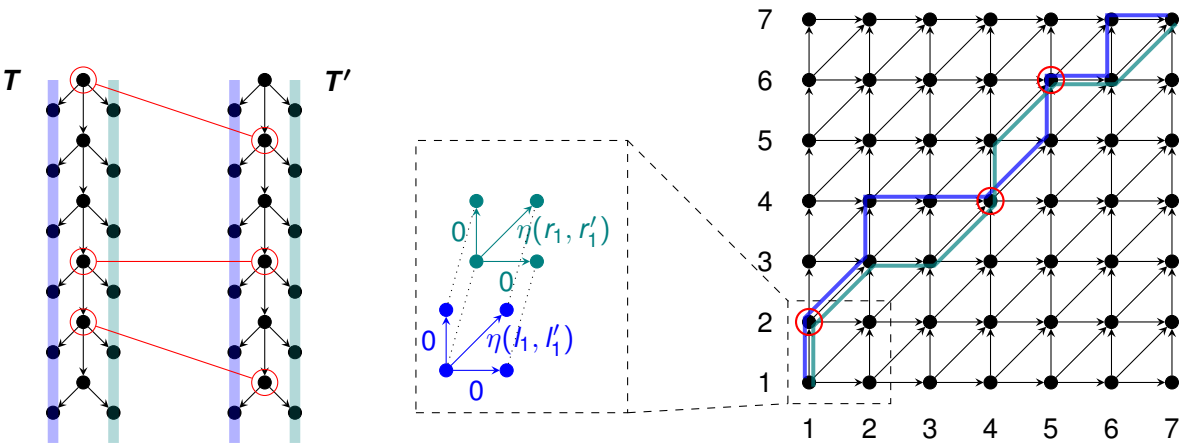




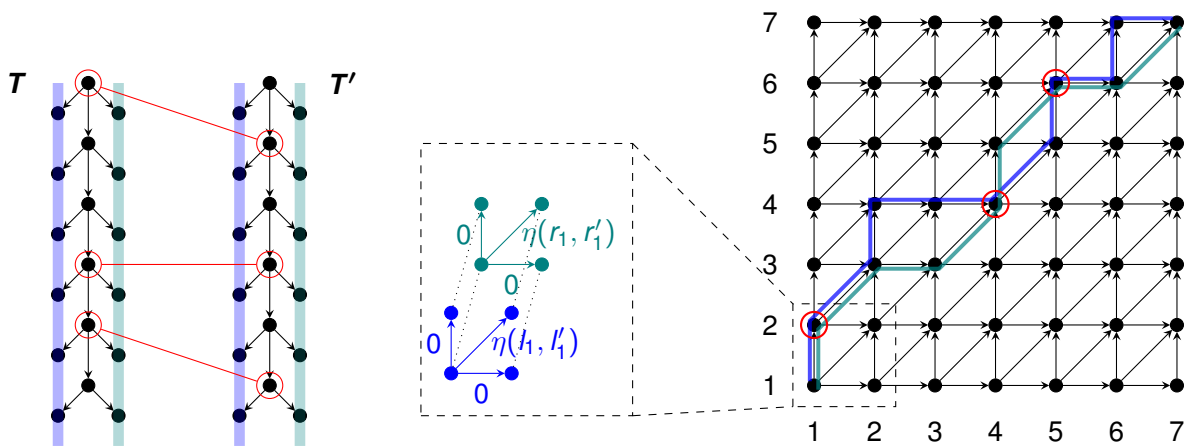
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



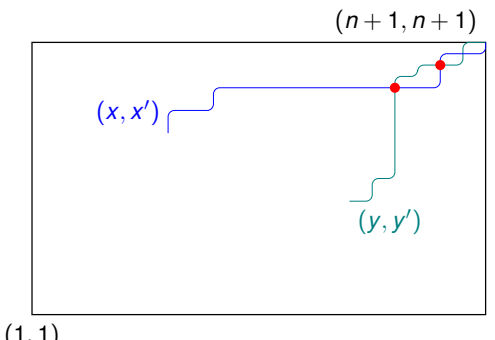
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



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).

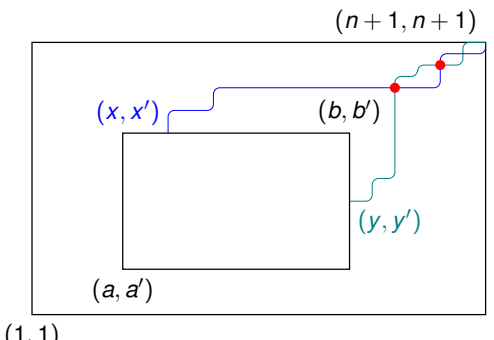
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



$\mathrm{sim}((x, x'), (y, y'))$ 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).

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Divide et Conquer Scheme

Input:

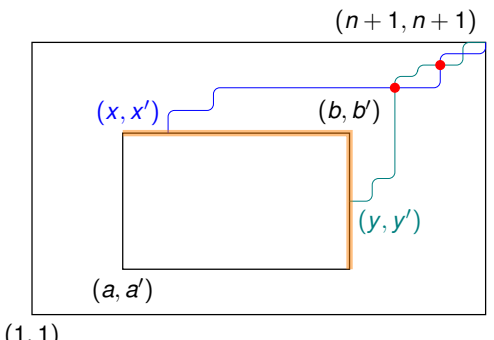
- The lower-left corner (a, a') and upper-right corner (b, b') of a rectangle.

Output:

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

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Divide et Conquer Scheme

Input:

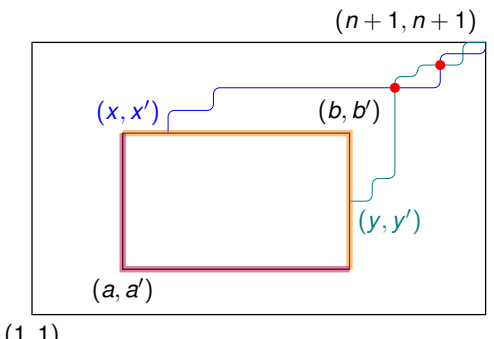
- The lower-left corner (a, a') and upper-right corner (b, b') of a rectangle.
- $\mathrm{sim}((x, x'), (y, y'))$
  $\forall (x, x'), (y, y') \in ([a\,..\,b] \times \{b'\}) \cup (\{b\} \times [a'\,..\,b'])$.

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).

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Divide et Conquer Scheme

Input:

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

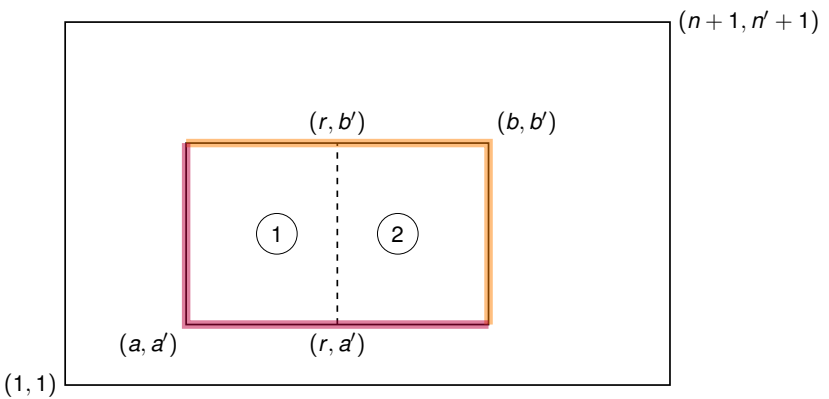
Output:

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

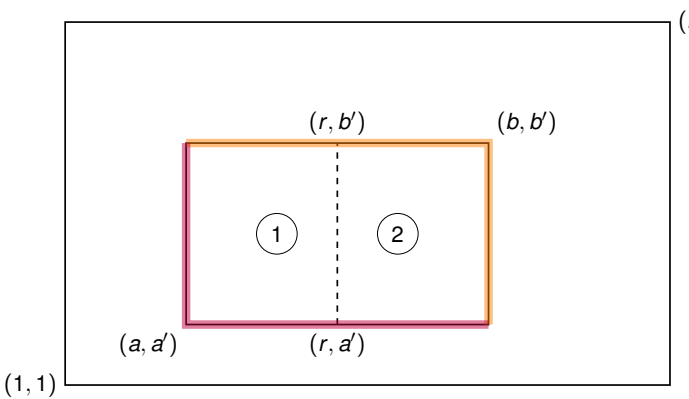
$\mathrm{sim}((x, x'), (y, y'))$ equals to the maximum achievable sum of:

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).

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



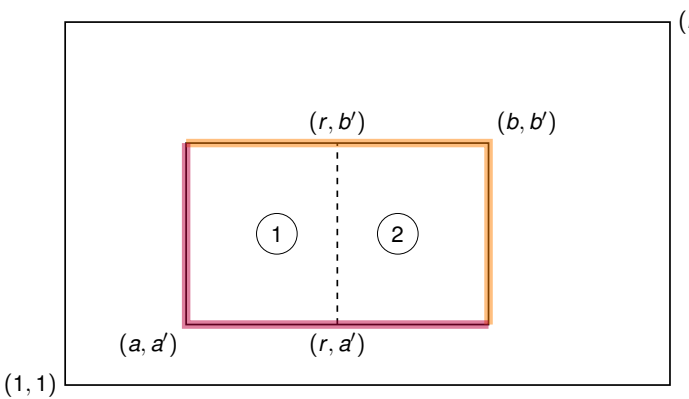
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

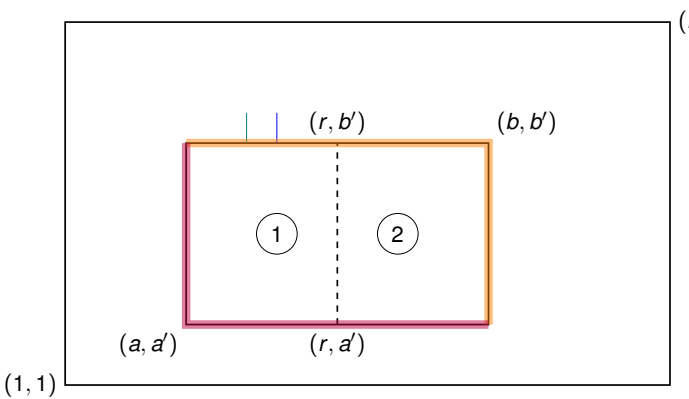


Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

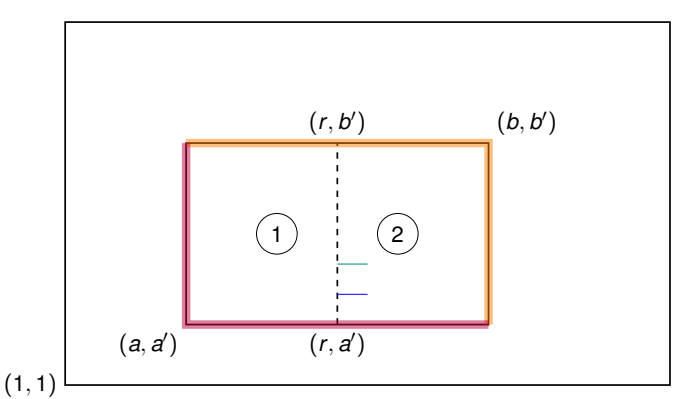


Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

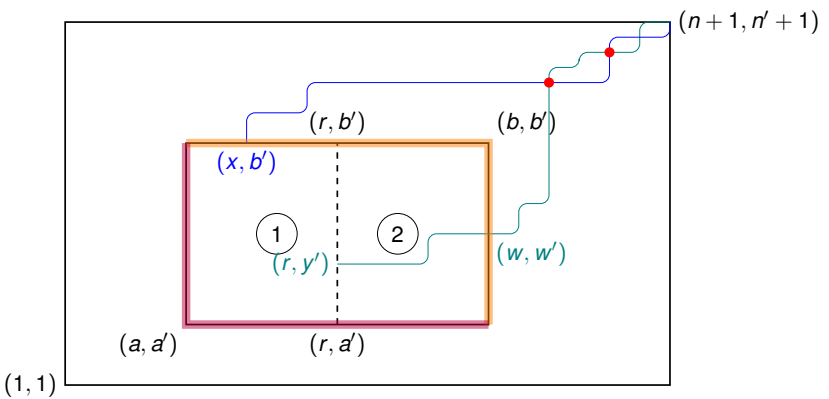


Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1.

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Inputs of subrectangle 2. ✔

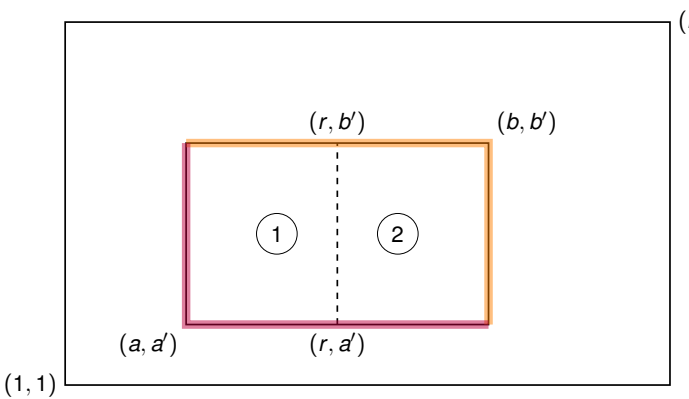
Recurse on subrectangle 2. ✔

Inputs of subrectangle 1. ✔

Recurse on subrectangle 1. ✔

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

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Inputs of subrectangle 2. ✔

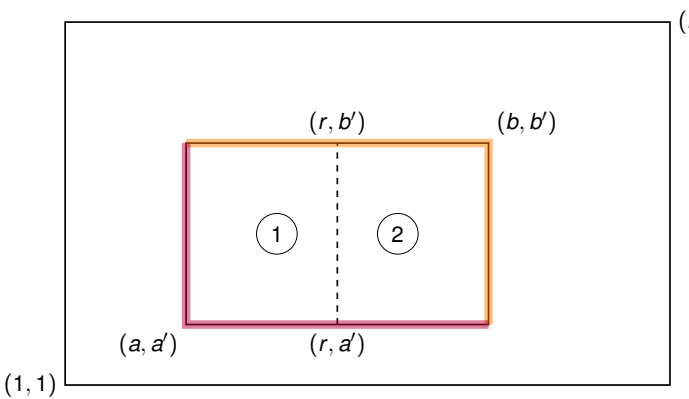
Recurse on subrectangle 2. ✔

Inputs of subrectangle 1. ✔

Recurse on subrectangle 1. ✔

Outputs of big rectangle.

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.



Inputs of subrectangle 2. ✔

Recurse on subrectangle 2. ✔

Inputs of subrectangle 1. ✔

Recurse on subrectangle 1. ✔

Outputs of big rectangle. ✔

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

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

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

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

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

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

- We obtain the recurrence

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

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

- 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 $\varepsilon > 0$.

Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm under the Left-to-Left/Spine-to-spine/Right-to-Right restriction.

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

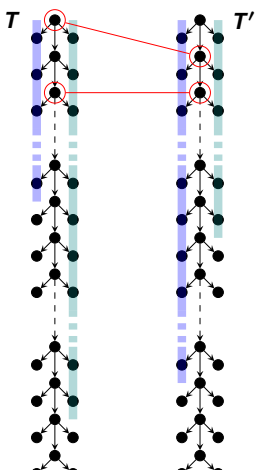
- We obtain the recurrence

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

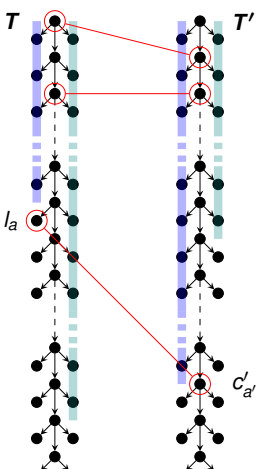
  Thus, $T(n) = \mathcal{O}(T_{\text{APSP}})$ assuming $T_{\text{APSP}} = \mathcal{O}(n^{2+\varepsilon})$ for some $\varepsilon > 0$.

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

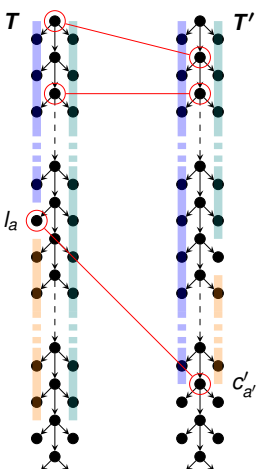
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm ~~under the Left-to-Left/Spine-to-spine/Right-to-Right restriction~~.



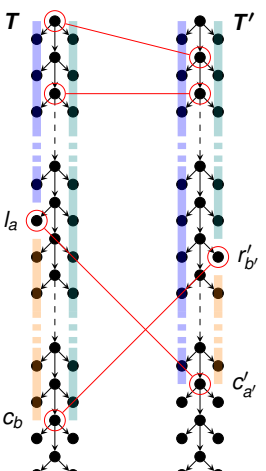
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm ~~under the Left-to-Left/Spine-to-spine/Right-to-Right restriction~~.



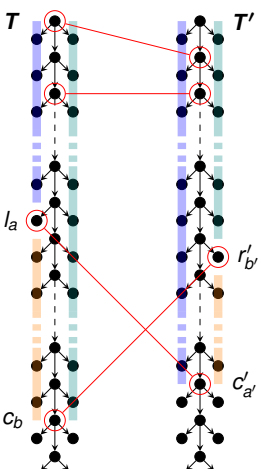
Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm ~~under the Left-to-Left/Spine-to-spine/Right-to-Right restriction~~.

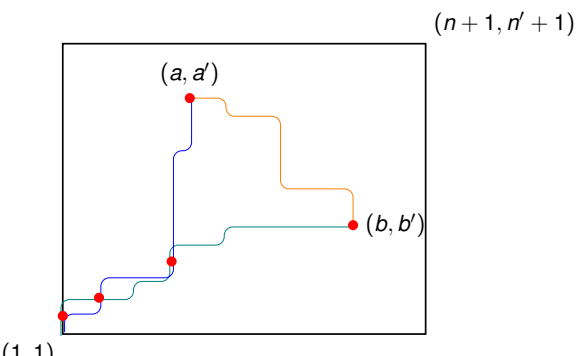


Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm ~~under the Left-to-Left/Spine-to-spine/Right-to-Right restriction~~.



Goal: Compute the similarity of two caterpillar trees T and T' via a Min-Plus Product algorithm ~~under the Left-to-Left/Spine-to-spine/Right-to-Right restriction~~.





Goal: Compute the similarity of two ~~caterpillar~~ trees T and T' via a Min-Plus Product algorithm.

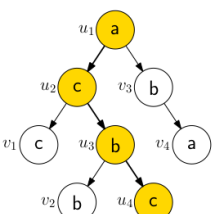
We generalize similarity computation of caterpillar trees to Spine Edit Distance.

Goal: Compute the similarity of two ~~caterpillar~~ trees T and T' via a Min-Plus Product algorithm.

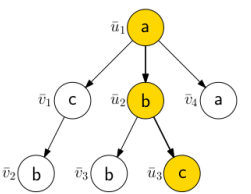
We generalize similarity computation of caterpillar trees to Spine Edit Distance.

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

T :



$\bar{T}$:

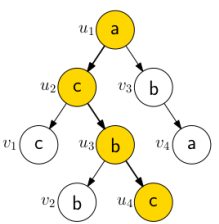


Goal: Compute the similarity of two ~~caterpillar~~ trees T and T' via a Min-Plus Product algorithm.

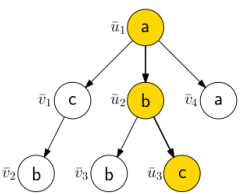
We generalize similarity computation of caterpillar trees to Spine Edit Distance.

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

T :



$\bar{T}$:



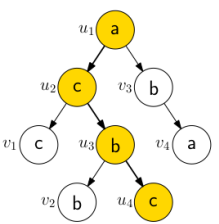
Output: $\text{sim}(\text{sub}(v), \text{sub}(v'))$ $\forall (v, v') \in S \times S'$.

Goal: Compute the similarity of two ~~caterpillar~~ trees T and T' via a Min-Plus Product algorithm.

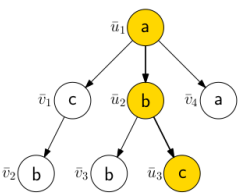
We generalize similarity computation of caterpillar trees to Spine Edit Distance.

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

T :



$\bar{T}$:



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

Why?

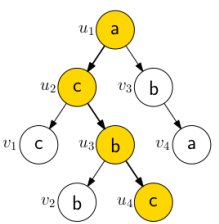
- There are still spine, left, and right nodes.
- The underlying paths are not in string alignment graphs anymore but in *forest alignment graphs*.

Goal: Compute the similarity of two ~~caterpillar~~ trees T and T' via a Min-Plus Product algorithm.

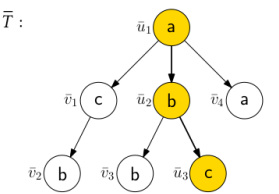
We generalize similarity computation of caterpillar trees to Spine Edit Distance.

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

T :



$\overline{T}$:

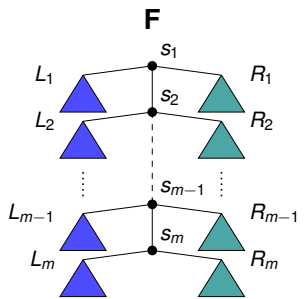


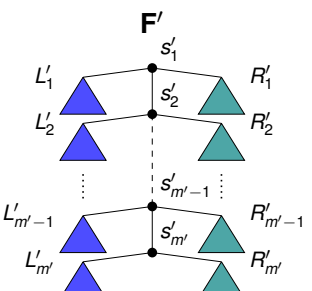
Output: $\text{sim}(\text{sub}(v), \text{sub}(v'))$ $\forall (v, v') \in S \times S'$.

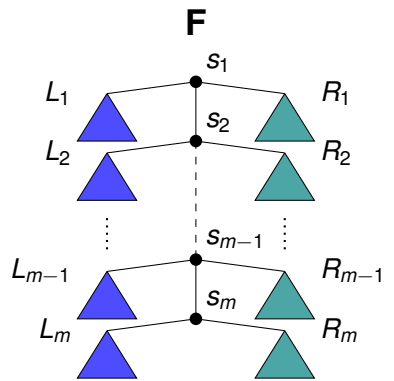
Why?

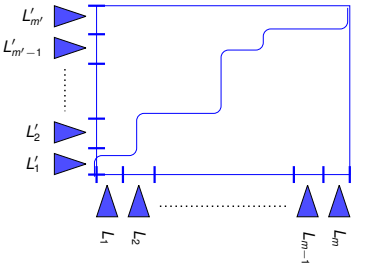
- There are still spine, left, and right nodes.
- The underlying paths are not in string alignment graphs anymore but in *forest alignment graphs*.

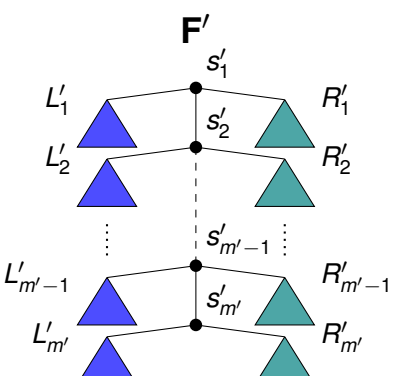
We also prove that similarity computation on arbitrary trees is fine-grained equivalent to Spine Edit Distance.

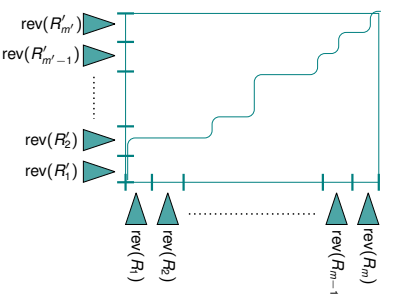

F
s_1
L_1
R_1
s_2
L_2
R_2
s_{m-1}
L_{m-1}
R_{m-1}
s_m
L_m
R_m

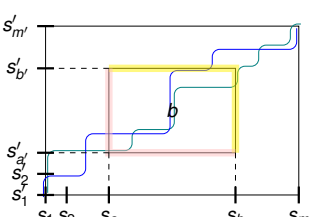

F′
s'_1
L'_1
R'_1
s'_2
L'_2
R'_2
$s'_{m'-1}$
$L'_{m'-1}$
$R'_{m'-1}$
$s'_{m'}$
$L'_{m'}$
$R'_{m'}$

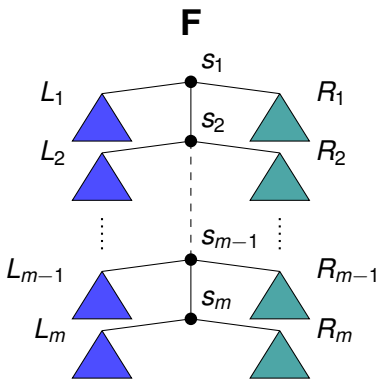

F
s_1
L_1
R_1
s_2
L_2
R_2
s_{m-1}
L_{m-1}
R_{m-1}
s_m
L_m
R_m

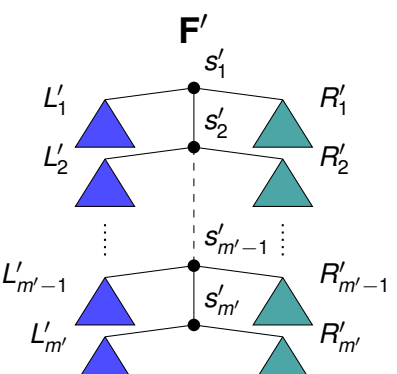

$L'_{m'}$
$L'_{m'-1}$
L'_2
L'_1
L_1
L_2
L_{m-1}
L_m

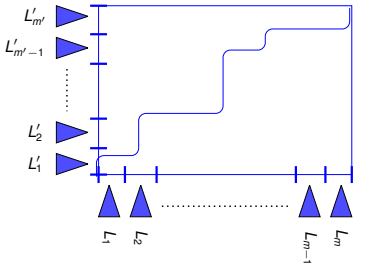

F′
s'_1
L'_1
R'_1
s'_2
L'_2
R'_2
$s'_{m'-1}$
$L'_{m'-1}$
$R'_{m'-1}$
$s'_{m'}$
$L'_{m'}$
$R'_{m'}$

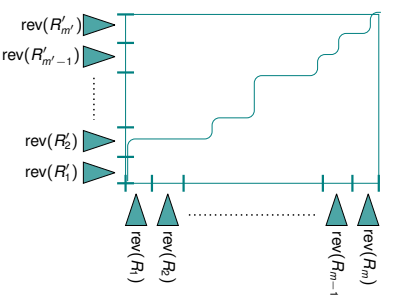

rev($R'_{m'}$)
rev($R'_{m'-1}$)
rev(R'_2)
rev(R'_1)
rev(R_1)
rev(R_2)
rev(R_{m-1})
rev(R_m)

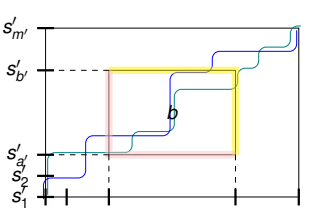

$s'_{m'}$
$s'_{b'}$
b
$s'_{a'}$
s'_2
s'_1
s_1 s_2 s_a s_b s_m

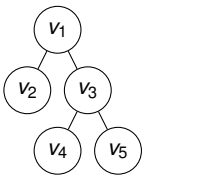

F
s_1
L_1
R_1
s_2
L_2
R_2
s_{m-1}
L_{m-1}
R_{m-1}
s_m
L_m
R_m

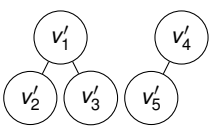

F′
s'_1
L'_1
R'_1
s'_2
L'_2
R'_2
$s'_{m'-1}$
$L'_{m'-1}$
$R'_{m'-1}$
$s'_{m'}$
$L'_{m'}$
$R'_{m'}$

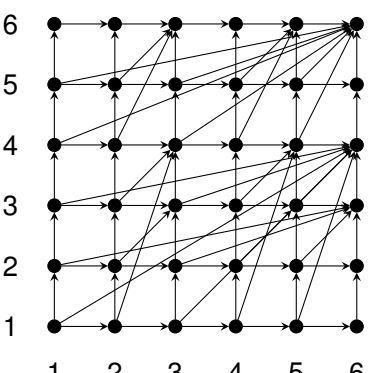

$L'_{m'}$
$L'_{m'-1}$
L'_2
L'_1
L_1
L_2
L_{m-1}
L_m


rev($R'_{m'}$)
rev($R'_{m'-1}$)
rev(R'_2)
rev(R'_1)
rev(R_1)
rev(R_2)
rev(R_{m-1})
rev(R_m)


$s'_{m'}$
$s'_{b'}$
b
$s'_{a'}$
s'_2
s'_1
s_1 s_2
s_a
s_b
s_m


v_1
v_2
v_3
v_4
v_5


v'_1
v'_4
v'_2
v'_3
v'_5


6
5
4
3
2
1
1 2 3 4 5 6

Two Open Problems

- Find any evidence that matrix multiplication techniques are needed for unweighted TED.

Two Open Problems

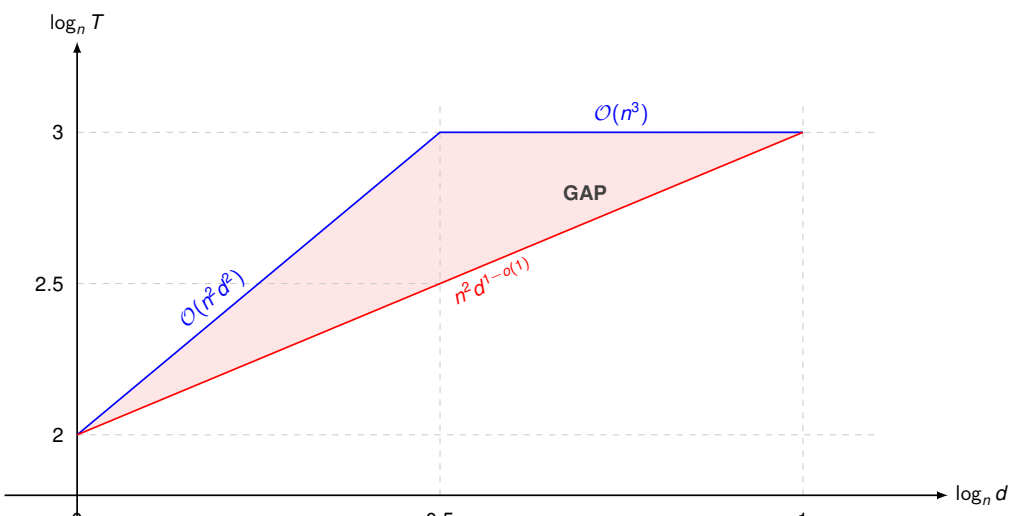
- Find any evidence that matrix multiplication techniques are needed for unweighted TED.

- (Weighted) TED parametrized by depth: $\mathcal{O}(\min(n^3, n^2 d^2))$-time algorithm versus $n^2 d^{1-o(1)}$ lower bound
  [Charalampopoulos, Gawrychowski, Mozes and Weimann, SPIRE 2022]

Two Open Problems

- Find any evidence that matrix multiplication techniques are needed for unweighted TED.

- (Weighted) TED parametrized by depth: $\mathcal{O}(\min(n^3, n^2d^2))$-time algorithm versus $n^2d^{1-o(1)}$ lower bound
  [Charalampopoulos, Gawrychowski, Mozes and Weimann, SPIRE 2022]



Thanks!