Hardness of Tree Edit Distance and Friends

Bingbing Hu[1] **Jakob Nogler**[2] Barna Saha[1]

[1]UC San Diego

[2]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 $\delta(c, c')$

`abcxef` ⟶ `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 $\delta(c, c')$

   `abcxef` ⟶ `abcyef`

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

   `abcdef` ⟶ `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` ⟶ `abcyef`

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

   `abcdef` ⟶ `abdef`

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

   `abcef` ⟶ `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` → `abcyef`

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

   `abcdef` → `abdef`

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

   `abcef` → `abcxef`

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

(String) Edit Distance Algorithms

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	$\Omega(n^{2-o(1)})$	under SETH, already for unweighted

Dynamic (String) Edit Distance

Strings undergo updates (insertion/deletions and substitutions), and we need to maintain their edit distance.

$$\text{ed}(\texttt{port}, \texttt{arts}) = 3$$

Dynamic (String) Edit Distance

Strings undergo updates (insertion/deletions and substitutions), and we need to maintain their edit distance.

$$\text{ed}(\ \texttt{part}\ ,\ \texttt{arts}\ ) = 2$$

Dynamic (String) Edit Distance

Strings undergo updates (insertion/deletions and substitutions), and we need to maintain their edit distance.

$$\text{ed}(\texttt{ part },\ \texttt{art }) = 1$$

Dynamic (String) Edit Distance

Strings undergo updates (insertion/deletions and substitutions), and we need to maintain their edit distance.

$$\mathrm{ed}(\,\texttt{part}\,,\ \texttt{art}\,) = 1$$

Question: Can we do better than recomputing from scratch?

Dynamic (String) Edit Distance

Strings undergo updates (insertion/deletions and substitutions), and we need to maintain their edit distance.

$$\mathrm{ed}(\texttt{part}, \texttt{art}) = 1$$

Question: Can we do better than recomputing from scratch?

References	**Update Time**	**Remarks**
CKM20	$\widetilde{\mathcal{O}}(n)$	unweighted
CKM20	$\widetilde{\mathcal{O}}(n \cdot \min(\sqrt{n}, W))$	uniform weights, W is maximum weight
GK25	$\widetilde{\mathcal{O}}(nW)$	arbitrary weights
CKW23	$\Omega(n^{1.5-o(1)})$	under APSP, large W

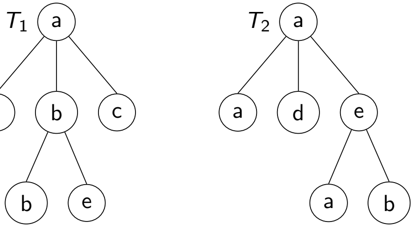
This Paper:
Generalization of (String) Edit Distance
in the Dynamic Setting

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.

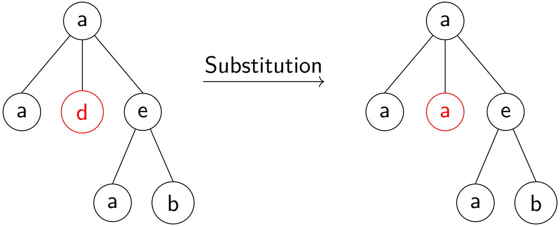


Tree Edit Distance

(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.

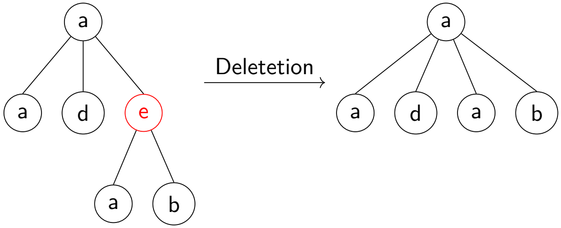


Tree Edit Distance

(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.

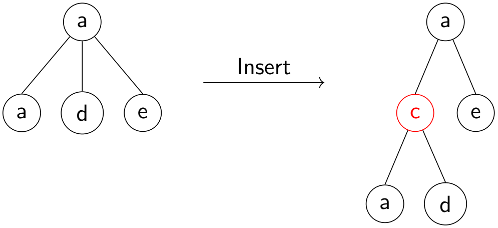


Tree Edit Distance

(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.



Tree Edit Distance (Background)

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

```
<?xml version="1.0"?>
- <Dataset xmlns="http://www.safe.com">
  - <Building id="Surrey Head Office">
      <Address>"7445 132 St."</Address>
      <City>Surrey</City>
      <Province>BC</Province>
      <Country>Canada</Country>
    - <Location>
        <Longitude>-122.860</Longitude>
        <Latitude>49.138</Latitude>
      </Location>
      <Reference>https://www.google.ca/maps/
        3m1!4b1!4m5!3m4!1s0x5485dbd520cc
        122.8574636?hl=en</Reference>
    - <Room id="Admin_100">
```

Source: www.support.safe.com

Algorithms for Tree Edit Distance

Reference	Complexity	Remarks
Tai79	$\mathcal{O}(n^6)$	weighted
SZ89	$\mathcal{O}(n^4)$	weighted
Klein98	$\mathcal{O}(n^3 \log n)$	weighted
DMRW10	$\mathcal{O}(n^3)$	weighted
BGMW20	no $\mathcal{O}(n^{3-\varepsilon})$ algo under APSP	weighted
NPSVWXY25	$n^3/2^{\Omega(\sqrt{\log n})}$	weighted
Mao22	$\mathcal{O}(n^{2.9546})$	unweighted
Dürr23	$\mathcal{O}(n^{2.9148})$	unweighted
NPSVWXY25	$\mathcal{O}(n^{2.687})$	unweighted

Algorithms for Tree Edit Distance

Reference	Complexity	Remarks
Tai79	$\mathcal{O}(n^6)$	weighted
SZ89	$\mathcal{O}(n^4)$	weighted
Klein98	$\mathcal{O}(n^3 \log n)$	weighted
DMRW10	$\mathcal{O}(n^3)$	weighted
BGMW20	no $\mathcal{O}(n^{3-\varepsilon})$ algo under APSP	weighted
NPSVWXY25	$n^3/2^{\Omega(\sqrt{\log n})}$	weighted
Mao22	$\mathcal{O}(n^{2.9546})$	unweighted
Dürr23	$\mathcal{O}(n^{2.9148})$	unweighted
NPSVWXY25	$\mathcal{O}(n^{2.687})$	unweighted

Question 1: How about *Dynamic* Tree Edit Distance?

Algorithms for Tree Edit Distance

Reference	**Complexity**	**Remarks**
Tai79	$\mathcal{O}(n^6)$	weighted
SZ89	$\mathcal{O}(n^4)$	weighted
Klein98	$\mathcal{O}(n^3 \log n)$	weighted
DMRW10	$\mathcal{O}(n^3)$	weighted
BGMW20	no $\mathcal{O}(n^{3-\varepsilon})$ algo under APSP	weighted
NPSVWXY25	$n^3/2^{\Omega(\sqrt{\log n})}$	weighted
Mao22	$\mathcal{O}(n^{2.9546})$	unweighted
Dürr23	$\mathcal{O}(n^{2.9148})$	unweighted
NPSVWXY25	$\mathcal{O}(n^{2.687})$	unweighted

Question 1: How about *Dynamic* Tree Edit Distance?

Question 2: Is Unweighted Edit Distance on trees harder than on strings?

Dyck 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.

$\Downarrow$ About Balanced Sequence of Parenthesis

Dyck Edit Distance Problem
Input: A string S over a set of parenthesis
Output: Closest S' to S w.r.t. edit distance such that S' is balanced

Dyck 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.

$\Downarrow$ About Balanced Sequence of Parenthesis

Dyck Edit Distance Problem
Input: A string S over a set of parenthesis
Output: Closest S' to S w.r.t. edit distance such that S' is balanced

$$S = \mathbf{(\,(\,\{\,)\,]} \quad \longrightarrow \quad S' = \mathbf{(\,\{\,\}\,)}$$

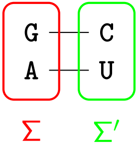
distance = 3

RNA Folding

RNA Folding
Input: A string S over alphabets $\Sigma \cup \Sigma'$, where each $\sigma \in \Sigma$ has a matching symbol $\sigma' \in \Sigma'$.
Output: Max. number of intersection-free connections between matching symbols σ, σ'.

RNA Folding

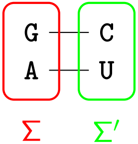
RNA Folding
Input: A string S over alphabets $\Sigma \cup \Sigma'$, where each $\sigma \in \Sigma$ has a matching symbol $\sigma' \in \Sigma'$.
Output: Max. number of intersection-free connections between matching symbols σ, σ'.

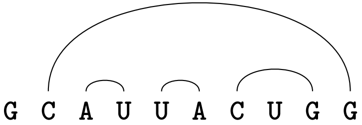


G C A U U A C U G G

RNA Folding

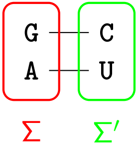
RNA Folding
Input: A string S over alphabets $\Sigma \cup \Sigma'$, where each $\sigma \in \Sigma$ has a matching symbol $\sigma' \in \Sigma'$.
Output: Max. number of intersection-free connections between matching symbols σ, σ'.

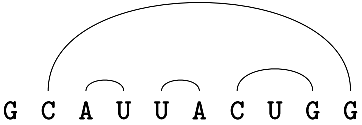




RNA Folding

RNA Folding
Input: A string S over alphabets $\Sigma \cup \Sigma'$, where each $\sigma \in \Sigma$ has a matching symbol $\sigma' \in \Sigma'$.
Output: Max. number of intersection-free connections between matching symbols σ, σ'.





Relation with edit distance: we can embed edit distance in RNA folding.

Algorithms for Dyck Edit Distance and RNA Folding

References	Time	Remarks
AP72, NJ80	$\mathcal{O}(n^3)$	
VGF14	$\mathcal{O}(n^3 / \log n)$	
BGSVW17	$\mathcal{O}(n^{2.8603})$	
CDXW22	$\mathcal{O}(n^{2.687})$	
ABVW18	$\Omega(n^{3-o(1)})$	under k-Clique Detection, combinatorial

Algorithms for Dyck Edit Distance and RNA Folding

References	Time	Remarks
AP72, NJ80	$\mathcal{O}(n^3)$	
VGF14	$\mathcal{O}(n^3/\log n)$	
BGSVW17	$\mathcal{O}(n^{2.8603})$	
CDXW22	$\mathcal{O}(n^{2.687})$	
ABVW18	$\Omega(n^{3-o(1)})$	under k-Clique Detection, combinatorial

Question 3: How about *Dynamic* Dyck Edit Distance and RNA Folding?

Our Results: Bad News...

Our Results

Question 1: How about *Dynamic* Tree Edit Distance?

Our Results

Question 1: How about *Dynamic* Tree Edit Distance?

Theorem

For any $\varepsilon > 0$, there are no dynamic algorithms:

- for weighted TED with $\mathcal{O}(n^{3-\varepsilon})$ update time, unless Weighted 4-Clique Conj. fails;
- for unweighted TED with $\mathcal{O}(n^{2-\varepsilon})$ (comb.) updated time, unless k-Clique Detection Conj. fails.

Our Results

Question 1: How about *Dynamic* Tree Edit Distance?

Theorem

For any $\varepsilon > 0$, there are no dynamic algorithms:

- for weighted TED with $\mathcal{O}(n^{3-\varepsilon})$ update time, unless Weighted 4-Clique Conj. fails;
- for unweighted TED with $\mathcal{O}(n^{2-\varepsilon})$ (comb.) updated time, unless k-Clique Detection Conj. fails.

Question 2: Is Unweighted Edit Distance on trees harder than on strings?

Dynamic Unweighted Edit Distance $\neq$ Dynamic Unweighted Tree Edit Distance

Our Results

Question 1: How about *Dynamic* Tree Edit Distance?

Theorem

For any $\varepsilon > 0$, there are no dynamic algorithms:

- for weighted TED with $\mathcal{O}(n^{3-\varepsilon})$ update time, unless Weighted 4-Clique Conj. fails;
- for unweighted TED with $\mathcal{O}(n^{2-\varepsilon})$ (comb.) updated time, unless k-Clique Detection Conj. fails.

Question 2: Is Unweighted Edit Distance on trees harder than on strings?

Dynamic Unweighted Edit Distance $\neq$ Dynamic Unweighted Tree Edit Distance

Question 3: How about *Dynamic* Dyck Edit Distance and RNA Folding?

Our Results

Question 1: How about *Dynamic* Tree Edit Distance?

Theorem

For any $\varepsilon > 0$, there are no dynamic algorithms:

- for weighted TED with $\mathcal{O}(n^{3-\varepsilon})$ update time, unless Weighted 4-Clique Conj. fails;
- for unweighted TED with $\mathcal{O}(n^{2-\varepsilon})$ (comb.) updated time, unless k-Clique Detection Conj. fails.

Question 2: Is Unweighted Edit Distance on trees harder than on strings?

Dynamic Unweighted Edit Distance $\neq$ Dynamic Unweighted Tree Edit Distance

Question 3: How about *Dynamic* Dyck Edit Distance and RNA Folding?

Theorem

For any $\varepsilon > 0$, there are no dynamic algorithms for Dyck Edit Distance and RNA Folding with $\mathcal{O}(n^{3-\varepsilon})$ (comb.) updated time, unless k-Clique Detection Conj. fails

The Two Conjectures

k-**Clique Detection**
Input: A unweighted graph $G = (V, E)$ on n nodes.
Output: YES if there are $v_1, \ldots, v_k \in V$ such that $v_1, \ldots, v_k$ is a k-clique, and NO otherwise.

The Two Conjectures

k-Clique Detection
Input: A unweighted graph $G = (V, E)$ on n nodes.
Output: YES if there are $v_1, \ldots, v_k \in V$ such that $v_1, \ldots, v_k$ is a k-clique, and NO otherwise.

k-Clique Detection Conjecture

For any $\varepsilon > 0$, k-Clique Detection cannot be solved in $\mathcal{O}(n^{k-\varepsilon})$ time by any combinatorial algorithm.

The Two Conjectures

k-Clique Detection
Input: A unweighted graph $G = (V, E)$ on n nodes.
Output: $\textsc{Yes}$ if there are $v_1, \ldots, v_k \in V$ such that $v_1, \ldots, v_k$ is a k-clique, and $\textsc{No}$ otherwise.

k-Clique Detection Conjecture

For any $\varepsilon > 0$, k-Clique Detection cannot be solved in $\mathcal{O}(n^{k-\varepsilon})$ time by any combinatorial algorithm.

Weighted k-Clique
Input: A weighted graph $G = (V, E, w)$.
Output: $\min_{v_1,\ldots,v_k \in V} \sum_{i<j} w(v_i, v_j)$ such that $v_1, \ldots, v_k$ is a k-clique.

The Two Conjectures

k-Clique Detection
Input: A unweighted graph $G = (V, E)$ on n nodes.
Output: YES if there are $v_1, \ldots, v_k \in V$ such that $v_1, \ldots, v_k$ is a k-clique, and NO otherwise.

k-Clique Detection Conjecture

For any $\varepsilon > 0$, k-Clique Detection cannot be solved in $\mathcal{O}(n^{k-\varepsilon})$ time by any combinatorial algorithm.

Weighted k-Clique
Input: A weighted graph $G = (V, E, w)$.
Output: $\min_{v_1,\ldots,v_k \in V} \sum_{i<j} w(v_i, v_j)$ such that $v_1, \ldots, v_k$ is a k-clique.

Weighted k-Clique Conjecture

For any $\varepsilon > 0$, there is $c > 0$ such that for any $k \geqslant 3$, the Weighted k-Clique with edge weights in $\{1, \ldots, n^{ck}\}$ cannot be solved in $\mathcal{O}(n^{k(1-\varepsilon)})$ time.

Prook Sketch:
Lower Bound for Dynamic Unweighted Tree Edit Distance

Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;

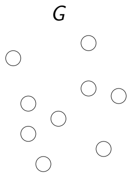
Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;



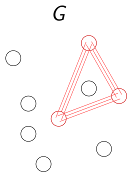
Proof Sketch

3k-Clique Detection Conjecture

For any $\varepsilon > 0$, 3k-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \dots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;



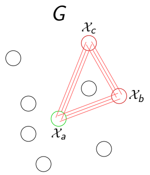
Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique.



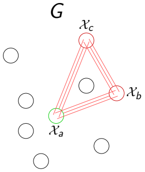
Proof Sketch

3k-Clique Detection Conjecture

For any $\varepsilon > 0$, 3k-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. ted$(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is 3k-clique.
  Size of $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ is $n^{k+\mathcal{O}(1)}$ and dependence on $\mathcal{X}_a$ is $n^{\mathcal{O}(1)}$.



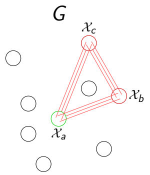
Proof Sketch

3k-Clique Detection Conjecture

For any $\varepsilon > 0$, 3k-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is 3k-clique.
  Size of $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ is $n^{k+\mathcal{O}(1)}$ and dependence on $\mathcal{X}_a$ is $n^{\mathcal{O}(1)}$.
- Proceed in N rounds. In round i:



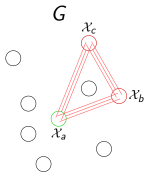
Proof Sketch

3k-Clique Detection Conjecture

For any $\varepsilon > 0$, 3k-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\mathrm{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is 3k-clique.
  Size of $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ is $n^{k+\mathcal{O}(1)}$ and dependence on $\mathcal{X}_a$ is $n^{\mathcal{O}(1)}$.
- Proceed in N rounds. In round i:
  - Verify whether there are $\mathcal{X}_b, \mathcal{X}_c$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is 3k-clique,



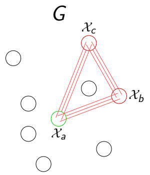
Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\mathrm{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique.
  Size of $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ is $n^{k+\mathcal{O}(1)}$ and dependence on $\mathcal{X}_a$ is $n^{\mathcal{O}(1)}$.
- Proceed in N rounds. In round i:
  - Verify whether there are $\mathcal{X}_b, \mathcal{X}_c$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique,
  - Transform $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ into $\mathbf{T}(\mathcal{X}_{a+1}), \mathbf{T}'(\mathcal{X}_{a+1})$ via $n^{\mathcal{O}(1)}$ updates.



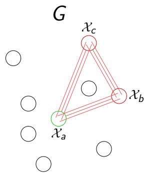
Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique.
  Size of $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ is $n^{k+\mathcal{O}(1)}$ and dependence on $\mathcal{X}_a$ is $n^{\mathcal{O}(1)}$.
- Proceed in N rounds. In round i:
  - Verify whether there are $\mathcal{X}_b, \mathcal{X}_c$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique,
  - Transform $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ into $\mathbf{T}(\mathcal{X}_{a+1}), \mathbf{T}'(\mathcal{X}_{a+1})$ via $n^{\mathcal{O}(1)}$ updates.



Runtime: $\underbrace{n^k}_{\text{rounds}} \times \underbrace{n^{\mathcal{O}(1)}}_{\text{updates per rounds}} \times \underbrace{(n^{k+O(1)})^{2-\varepsilon'}}_{\text{time per update}}$

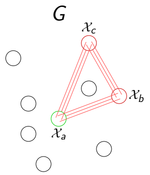
Proof Sketch

$3k$-Clique Detection Conjecture

For any $\varepsilon > 0$, $3k$-Clique Detection cannot be solved in $\mathcal{O}(n^{3k-\varepsilon})$ time by any combinatorial algorithm.

Steps:

- List all k-cliques $\mathcal{X}_1, \ldots, \mathcal{X}_N$ in time $\mathcal{O}(n^k)$;
- For each k-clique $\mathcal{X}_a$, we construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique.
  Size of $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ is $n^{k+\mathcal{O}(1)}$ and dependence on $\mathcal{X}_a$ is $n^{\mathcal{O}(1)}$.
- Proceed in N rounds. In round i:
  - Verify whether there are $\mathcal{X}_b, \mathcal{X}_c$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is $3k$-clique,
  - Transform $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ into $\mathbf{T}(\mathcal{X}_{a+1}), \mathbf{T}'(\mathcal{X}_{a+1})$ via $n^{\mathcal{O}(1)}$ updates.



$$\text{Runtime: } \underbrace{n^k}_{\text{rounds}} \times \underbrace{n^{\mathcal{O}(1)}}_{\text{updates per rounds}} \times \underbrace{(n^{k+O(1)})^{2-\varepsilon'}}_{\text{time per update}} \overset{k \text{ large enough}}{\leq} n^{3k-\varepsilon}$$

Clique Gadgets

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\mathrm{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

Core Gadget: Given G, there exist two string embeddings $\mathrm{CLG} : V^k \to \Sigma^{\lambda_1}$ and $\mathrm{CNG} : V^k \to \Sigma^{\lambda_2}$ of lengths $\lambda_1, \lambda_2 = n^{O(1)}$ and a constant C such that for any k-cliques $\mathcal{X}, \mathcal{Y} \in V^k$:

$$\begin{aligned} \mathrm{ed}(\mathrm{CLG}(\mathcal{X}), \mathrm{CNG}(\mathcal{Y})) &= C && \text{if } \mathcal{X} \text{ is fully connected with } \mathcal{Y}, \\ \mathrm{ed}(\mathrm{CLG}(\mathcal{X}), \mathrm{CNG}(\mathcal{Y})) &> C && \text{otherwise.} \end{aligned}$$

Clique Gadgets

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

Core Gadget: Given G, there exist two string embeddings $\text{CLG} : V^k \to \Sigma^{\lambda_1}$ and $\text{CNG} : V^k \to \Sigma^{\lambda_2}$ of lengths $\lambda_1, \lambda_2 = n^{O(1)}$ and a constant C such that for any k-cliques $\mathcal{X}, \mathcal{Y} \in V^k$:

$$\begin{aligned} \text{ed}(\text{CLG}(\mathcal{X}), \text{CNG}(\mathcal{Y})) &= C && \text{if } \mathcal{X} \text{ is fully connected with } \mathcal{Y}, \\ \text{ed}(\text{CLG}(\mathcal{X}), \text{CNG}(\mathcal{Y})) &> C && \text{otherwise.} \end{aligned}$$

Remark: Similar gadgets already appear in ABVW17, but for different string similarity notions.

Proof Sketch II

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

$\mathbf{T}(\mathcal{X}_a)$

$$\begin{array}{ccc} \text{CNG}(\mathcal{X}_1) & + & \text{CLG}(\mathcal{X}_1) \\ \text{CNG}(\mathcal{X}_2) & + & \text{CLG}(\mathcal{X}_2) \\ & \vdots & \\ \text{CNG}(\mathcal{X}_b) & + & \text{CLG}(\mathcal{X}_b) \\ & \vdots & \\ \text{CNG}(\mathcal{X}_N) & + & \text{CLG}(\mathcal{X}_N) \end{array}$$

$$\text{CNG}(\mathcal{X}_a)$$

$\mathbf{T}'(\mathcal{X}_a)$

$$\begin{array}{ccc} \text{CNG}(\mathcal{X}_1) & + & \text{CLG}(\mathcal{X}_1) \\ \text{CNG}(\mathcal{X}_2) & + & \text{CLG}(\mathcal{X}_2) \\ & \vdots & \\ \text{CNG}(\mathcal{X}_c) & + & \text{CLG}(\mathcal{X}_c) \\ & \vdots & \\ \text{CNG}(\mathcal{X}_N) & + & \text{CLG}(\mathcal{X}_N) \end{array}$$

$$\text{CLG}(\mathcal{X}_a)$$

Proof Sketch II

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a)$, $\mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

$\mathbf{T}(\mathcal{X}_a)$

$\text{CNG}(\mathcal{X}_1) + \text{CLG}(\mathcal{X}_1)$
$\text{CNG}(\mathcal{X}_2) + \text{CLG}(\mathcal{X}_2)$
$\vdots$
$\text{CNG}(\mathcal{X}_b) + \text{CLG}(\mathcal{X}_b)$
$\vdots$
$\text{CNG}(\mathcal{X}_N) + \text{CLG}(\mathcal{X}_N)$

$\text{CNG}(\mathcal{X}_a)$

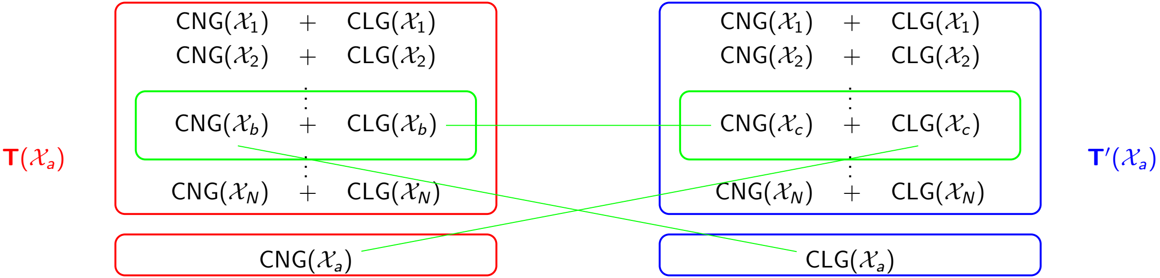
$\mathbf{T}'(\mathcal{X}_a)$

$\text{CNG}(\mathcal{X}_1) + \text{CLG}(\mathcal{X}_1)$
$\text{CNG}(\mathcal{X}_2) + \text{CLG}(\mathcal{X}_2)$
$\vdots$
$\text{CNG}(\mathcal{X}_c) + \text{CLG}(\mathcal{X}_c)$
$\vdots$
$\text{CNG}(\mathcal{X}_N) + \text{CLG}(\mathcal{X}_N)$

$\text{CLG}(\mathcal{X}_a)$

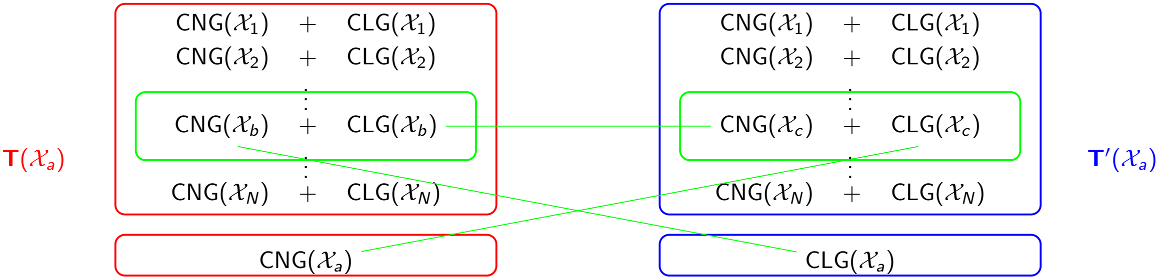
Proof Sketch II

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.



Proof Sketch II

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.



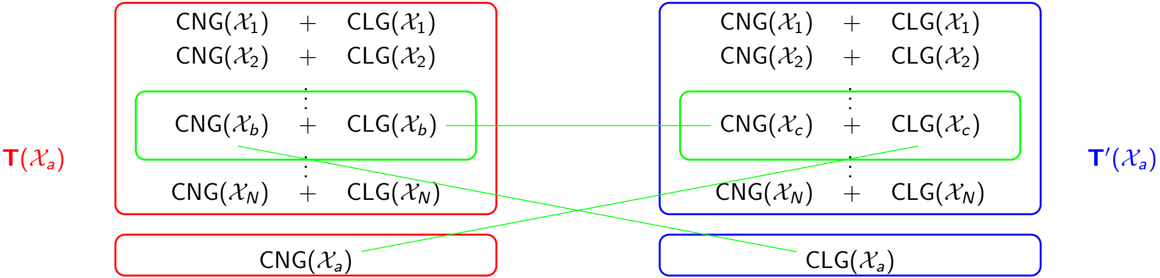
$$\text{ted}(\ \mathbf{T}(\mathcal{X}_a)\ ,\ \mathbf{T}'(\mathcal{X}_a)\ ) =$$

$$\min_{b,c} \text{ed}(\text{CNG}(\mathcal{X}_a), \text{CLG}(\mathcal{X}_b)) + \text{ed}(\text{CNG}(\mathcal{X}_b), \text{CLG}(\mathcal{X}_c)) + \text{ed}(\text{CNG}(\mathcal{X}_c), \text{CLG}(\mathcal{X}_a)) + D$$

where D is a constant

Proof Sketch II

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.



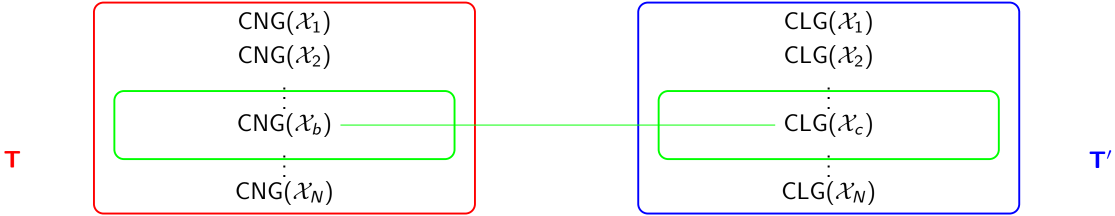
$$\text{ted}(\ \mathbf{T}(\mathcal{X}_a)\ ,\ \mathbf{T}'(\mathcal{X}_a)\ ) =$$

$$\min_{b,c} \text{ed}(\text{CNG}(\mathcal{X}_a), \text{CLG}(\mathcal{X}_b)) + \text{ed}(\text{CNG}(\mathcal{X}_b), \text{CLG}(\mathcal{X}_c)) + \text{ed}(\text{CNG}(\mathcal{X}_c), \text{CLG}(\mathcal{X}_a)) + D$$

$$\stackrel{?}{=} 3C + D$$

Simplified Goal I

Simplified Goal: Construct two trees $\mathbf{T}, \mathbf{T}'$ s.t. $\text{ted}(\mathbf{T}, \mathbf{T}')$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_b \cup \mathcal{X}_c$ is a $2k$-clique.

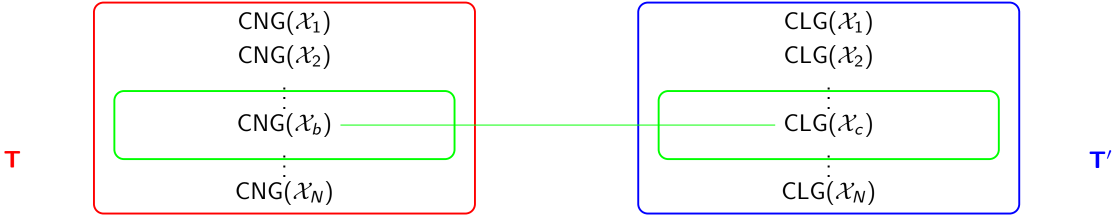


$$\text{ted}(\mathbf{T}, \mathbf{T}') =$$
$$\min_{b,c} \text{ed}(\text{CNG}(\mathcal{X}_b), \text{CLG}(\mathcal{X}_c)) + D$$

where D is a constant

Simplified Goal I

Simplified Goal: Construct two trees $\mathbf{T}, \mathbf{T}'$ s.t. $\text{ted}(\mathbf{T}, \mathbf{T}')$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_b \cup \mathcal{X}_c$ is a $2k$-clique.



$$\text{ted}(\ \mathbf{T}\ ,\ \mathbf{T}'\ ) =$$
$$\min_{b,c} \text{ed}(\text{CNG}(\mathcal{X}_b), \text{CLG}(\mathcal{X}_c)) + D$$
$$\stackrel{?}{=} C + D$$

Simplified Goal II

Simplified Goal: Construct two trees $\mathbf{T}, \mathbf{T}'$ s.t. $\text{ted}(\mathbf{T}, \mathbf{T}')$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_b \cup \mathcal{X}_c$ is a $2k$-clique.

Put all $\text{CLG}(\mathcal{X}_i)/\text{CNG}(\mathcal{X}_i)$ on a single spine.

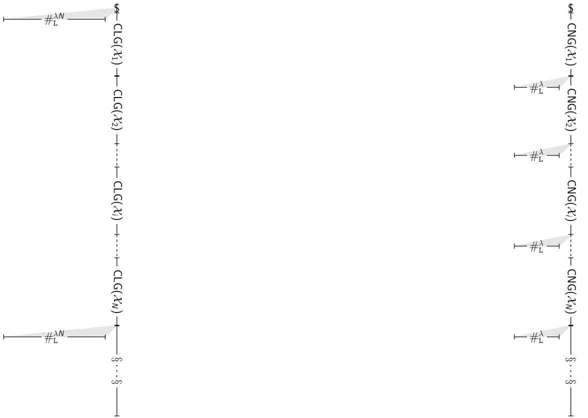
Simplified Goal II

Simplified Goal: Construct two trees $\mathbf{T}, \mathbf{T}'$ s.t. $\text{ted}(\mathbf{T}, \mathbf{T}')$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_b \cup \mathcal{X}_c$ is a $2k$-clique.

Append a long enough tail with a special character §.

Simplified Goal II

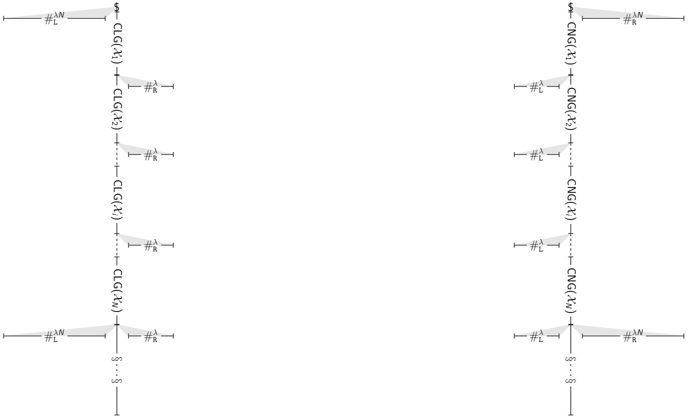
Simplified Goal: Construct two trees $\mathbf{T}, \mathbf{T}'$ s.t. $\mathrm{ted}(\mathbf{T}, \mathbf{T}')$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_b \cup \mathcal{X}_c$ is a $2k$-clique.



Append left special character $\#_{\mathrm{L}}$, here $\lambda \approx 100 \cdot$ length of CLG/CNG.

Simplified Goal II

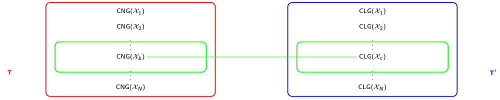
Simplified Goal: Construct two trees $\mathbf{T}, \mathbf{T}'$ s.t. $\text{ted}(\mathbf{T}, \mathbf{T}')$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_b \cup \mathcal{X}_c$ is a $2k$-clique.

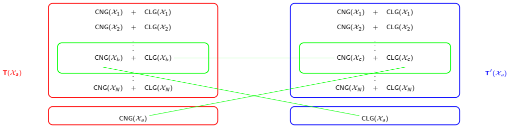


Do the same on the right.

Proof Sketch III

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.





$$\text{ted}(\ \mathbf{T}\ ,\ \mathbf{T}'\ ) =$$

$$\min_{b,c} \text{ed}(\text{CNG}(\mathcal{X}_b), \text{CLG}(\mathcal{X}_c)) + D$$

$$\text{ted}(\ \mathbf{T}(\mathcal{X}_a)\ ,\ \mathbf{T}'(\mathcal{X}_a)\ ) =$$

$$\min_{b,c} \text{ed}(\text{CNG}(\mathcal{X}_a), \text{CLG}(\mathcal{X}_b)) + \text{ed}(\text{CNG}(\mathcal{X}_b), \text{CLG}(\mathcal{X}_c)) + \text{ed}(\text{CNG}(\mathcal{X}_c), \text{CLG}(\mathcal{X}_a)) + D$$

Proof Sketch IV

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\mathrm{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

Squeeze in dependence on $\mathcal{X}_a$.

Proof Sketch IV

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

Size requirement: $n^{k+\mathcal{O}(1)}$. ✔

Proof Sketch IV

Goal: For each k-clique $\mathcal{X}_a$, construct two trees $\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a)$ s.t. $\text{ted}(\mathbf{T}(\mathcal{X}_a), \mathbf{T}'(\mathcal{X}_a))$ tells us whether there are k-cliques $\mathcal{X}_b, \mathcal{X}_c \subseteq V^k$ s.t. $\mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$ is a $3k$-clique.

Dependence on $\mathcal{X}_a$: $n^{\mathcal{O}(1)}$. ✔

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;
- Each round, we fix one k-clique and check whether there are 3 other k-cliques that together form $4k$-clique;

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;
- Each round, we fix one k-clique and check whether there are 3 other k-cliques that together form $4k$-clique;
- Somehow easier because static lower bound alreay tells us how to find $3k$-clique.

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;
- Each round, we fix one k-clique and check whether there are 3 other k-cliques that together form $4k$-clique;
- Somehow easier because static lower bound alreay tells us how to find $3k$-clique.

Weighted Tree Edit Distance

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;
- Each round, we fix one k-clique and check whether there are 3 other k-cliques that together form $4k$-clique;
- Somehow easier because static lower bound alreay tells us how to find $3k$-clique.

Weighted Tree Edit Distance

- We use Weighted $4k$-Clique;

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;
- Each round, we fix one k-clique and check whether there are 3 other k-cliques that together form $4k$-clique;
- Somehow easier because static lower bound alreay tells us how to find $3k$-clique.

Weighted Tree Edit Distance

- We use Weighted $4k$-Clique;
- Each round, we fix one node and find three nodes that minimize 4-clique where fixed node is in;

Other Lower Bounds

RNA Folding/Dyck Edit Distance:

- We use $4k$-Clique Detection;
- We again partition in k-cliques;
- Each round, we fix one k-clique and check whether there are 3 other k-cliques that together form $4k$-clique;
- Somehow easier because static lower bound alreay tells us how to find $3k$-clique.

Weighted Tree Edit Distance

- We use Weighted $4k$-Clique;
- Each round, we fix one node and find three nodes that minimize 4-clique where fixed node is in;
- Also somehow easier because static lower bound alreay tells us how to detect minimum weight 3-clique, i.e., triangle.

Thanks!