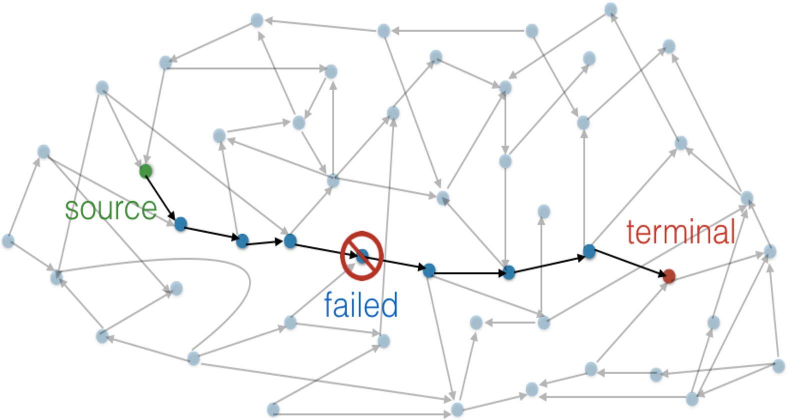
Improved Distance Sensitivity Oracles via Tree Partitioning

Ran Duan **Tianyi Zhang**

Tsinghua University

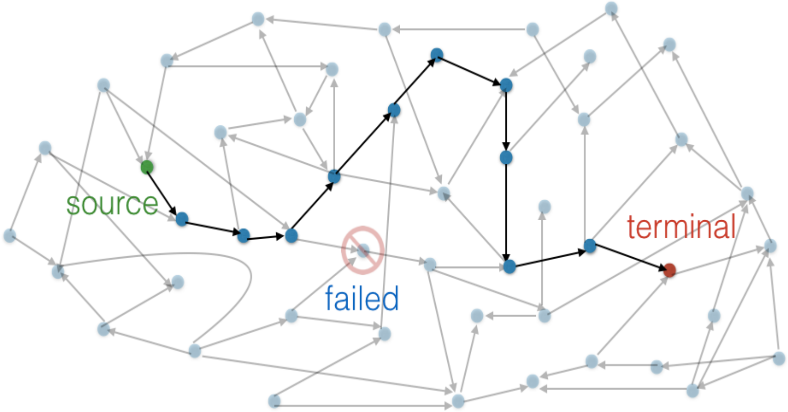
Distance sensitivity oralces (DSO)

Preprocess: given a directed graph $G = (V, E)$, edge weights $\omega : E \to \mathbb{R}^+$.
Query: $(s, t, f) \in V^3$
Answer: $\text{dist}_{G \setminus \{f\}}(s, t)$



Distance sensitivity oralces (DSO)

Preprocess: given a directed graph $G = (V, E)$, edge weights $\omega : E \to \mathbb{R}^+$.
Query: $(s, t, f) \in V^3$
Answer: $\text{dist}_{G \setminus \{f\}}(s, t)$



A short history

Define $n = |V|$, $m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$

A short history

Define $n = |V|$, $m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$
[DT02]	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$

A short history

Define $n = |V|,\ m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$
[DT02]	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$
[BK08]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(n^2\sqrt{m})$

A short history

Define $n = |V|$, $m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$
[DT02]	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$
[BK08]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(n^2\sqrt{m})$
[BK09]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(mn)$

A short history

Define $n = |V|$, $m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$
[DT02]	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$
[BK08]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(n^2\sqrt{m})$
[BK09]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(mn)$
[GW12]	$O(n^{2.34})$	$O(n^{0.9})$	$O(Mn^{2.92})$

A short history

Define $n = |V|$, $m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$
[DT02]	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$
[BK08]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(n^2\sqrt{m})$
[BK09]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(mn)$
[GW12]	$O(n^{2.34})$	$O(n^{0.9})$	$O(Mn^{2.92})$
New	$O(n^2)$	$O(1)$	$\tilde{O}(mn)$

A short history

Define $n = |V|$, $m = |E|$; assume $\mathrm{Im}(\omega) = \{1, 2, \cdots, M\}$.

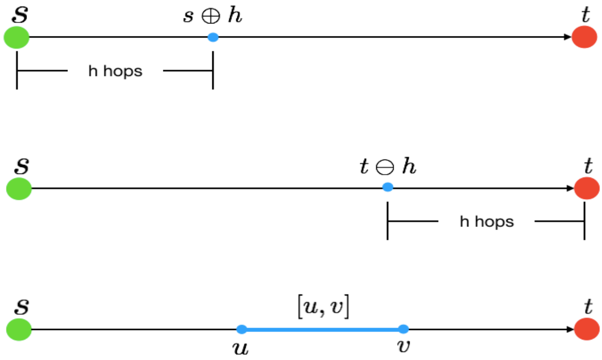
Reference	Space	Query time	Preprocessing
Naive	$O(n^3)$	$O(1)$	$\tilde{O}(mn^2)$
[DT02]	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$
[BK08]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(n^2\sqrt{m})$
[BK09]	$O(n^2 \log n)$	$O(1)$	$\tilde{O}(mn)$
[GW12]	$O(n^{2.34})$	$O(n^{0.9})$	$O(Mn^{2.92})$
New	$O(n^2)$	$O(1)$	$\tilde{O}(mn)$

The set-intersection conjecture [PRT12] implies any reachability oracle with constant query time has space $\tilde{\Omega}(n^2)$. So it is not clear if our space upper bound is tight.

Notations

Definition

On shortest path $s \rightsquigarrow t$, for any $h > 0$, define $s \oplus h$ to be the vertex which is h hops after s, and define $t \ominus h$ to be the vertex which is h hops before t. Also, for any u, v, define interval $[u, v]$ in the natural way.



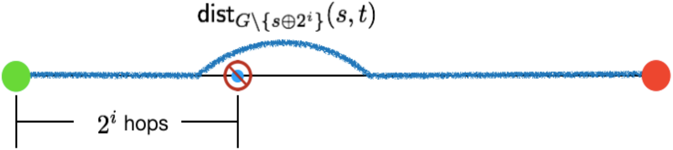
Sparse table [DT02]

Data structure: For any pair of $s, t \in V$, $\forall i \in [0, \log n]$

Sparse table [DT02]

Data structure: For any pair of $s, t \in V$, $\forall i \in [0, \log n]$

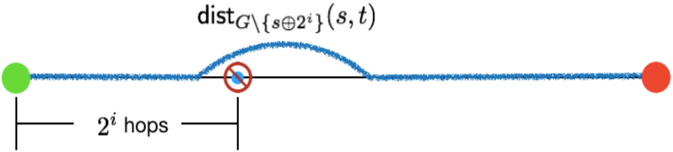
(i) Precompute $\text{dist}_{G \setminus \{s \oplus 2^i\}}(s, t)$ and $\text{dist}_{G \setminus \{t \ominus 2^i\}}(s, t)$.



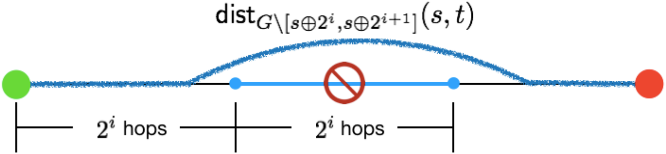
Sparse table [DT02]

Data structure: For any pair of $s, t \in V$, $\forall i \in [0, \log n]$

(i) Precompute $\text{dist}_{G \setminus \{s \oplus 2^i\}}(s, t)$ and $\text{dist}_{G \setminus \{t \ominus 2^i\}}(s, t)$.



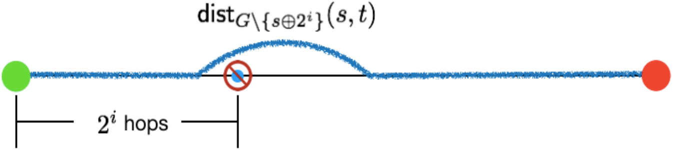
(ii) Precompute $\text{dist}_{G \setminus [s \oplus 2^i, s \oplus 2^{i+1}]}(s, t)$ and $\text{dist}_{G \setminus [t \ominus 2^{i+1}, t \ominus 2^i]}(s, t)$.



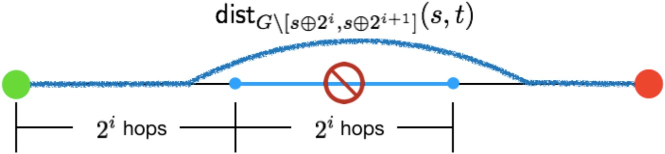
Sparse table [DT02]

Data structure: For any pair of $s, t \in V$, $\forall i \in [0, \log n]$

(i) Precompute $\text{dist}_{G \setminus \{s \oplus 2^i\}}(s, t)$ and $\text{dist}_{G \setminus \{t \ominus 2^i\}}(s, t)$.



(ii) Precompute $\text{dist}_{G \setminus [s \oplus 2^i, s \oplus 2^{i+1}]}(s, t)$ and $\text{dist}_{G \setminus [t \ominus 2^{i+1}, t \ominus 2^i]}(s, t)$.

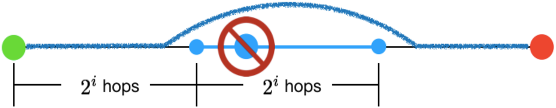


Space complexity: $O(n^2 \log n)$

Sparse table [DT02]

Query algorithm: On input (s, t, f), find the largest i such that $s \oplus 2^i$ comes before f, and the largest j such that $t \ominus 2^j$ comes after f.

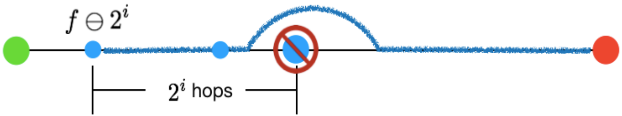
(1) Paths that skip interval $[s \oplus 2^i, s \oplus 2^{i+1}]$ entirely.



Sparse table [DT02]

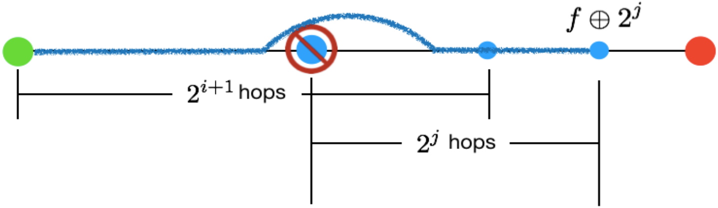
Query algorithm: On input (s, t, f), find the largest i such that $s \oplus 2^i$ comes before f, and the largest j such that $t \ominus 2^j$ comes after f.

(2) Paths that pass through $s \oplus 2^i$.



Query algorithm: On input (s, t, f), find the largest i such that $s \oplus 2^i$ comes before f, and the largest j such that $t \ominus 2^j$ comes after f.

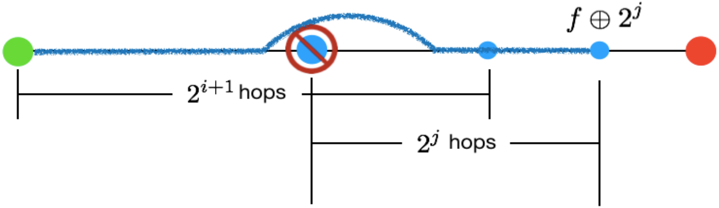
(3) Paths that pass through $s \oplus 2^{i+1}$. Such a path must also pass through $f \oplus 2^j$, so we retrieve $\text{dist}_{G \setminus \{f\}}(s, f \oplus 2^j)$ from storage.



Sparse table [DT02]

Query algorithm: On input (s, t, f), find the largest i such that $s \oplus 2^i$ comes before f, and the largest j such that $t \ominus 2^j$ comes after f.

(3) Paths that pass through $s \oplus 2^{i+1}$. Such a path must also pass through $f \oplus 2^j$, so we retrieve $\text{dist}_{G \setminus \{f\}}(s, f \oplus 2^j)$ from storage.



Query time: $O(1)$

Observations

Previous designs of DSO rely on sparse tables. Sparse table stores $O(\log n)$ entries for each pair of source & terminal, thus $\Omega(n^2 \log n)$ in total.

Observations

Previous designs of DSO rely on sparse tables. Sparse table stores $O(\log n)$ entries for each pair of source & terminal, thus $\Omega(n^2 \log n)$ in total.

New idea

- For each s, only choose a set of special terminals to store $O(\log n)$ entries.

Observations

Previous designs of DSO rely on sparse tables. Sparse table stores $O(\log n)$ entries for each pair of source & terminal, thus $\Omega(n^2 \log n)$ in total.

New idea

- For each s, only choose a set of special terminals to store $O(\log n)$ entries.
- If this proportion goes down to $O(n/\log n)$, then we have a **sparser table** of size $O(n^2)$.

Observations

Previous designs of DSO rely on sparse tables. Sparse table stores $O(\log n)$ entries for each pair of source & terminal, thus $\Omega(n^2 \log n)$ in total.

New idea

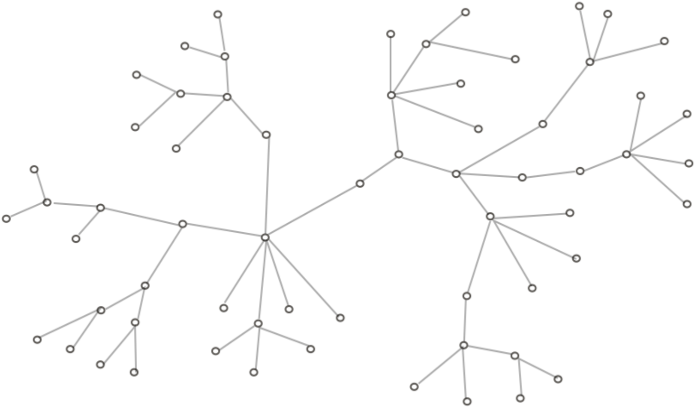
- For each s, only choose a set of special terminals to store $O(\log n)$ entries.
- If this proportion goes down to $O(n/\log n)$, then we have a **sparser table** of size $O(n^2)$.

How to select special terminals?

Tree partition lemma

Lemma

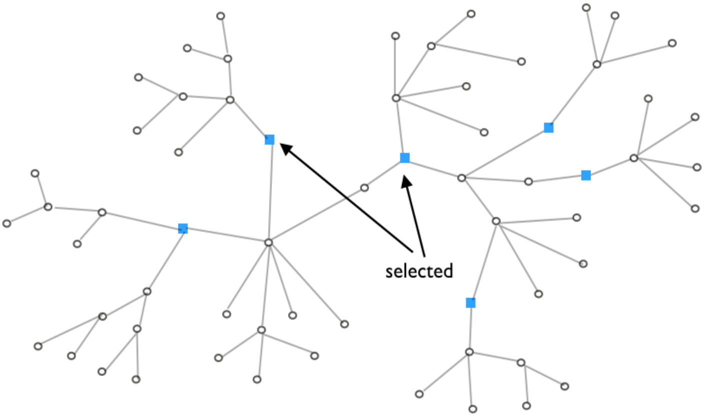
Let T be a spanning tree on n vertices. For any integer $2 \leq k \leq n$, we can select a subset of $\leq 3k - 5$ vertices whose removal partitions T into subtrees of size $\leq n/k$.



Tree partition lemma

Lemma

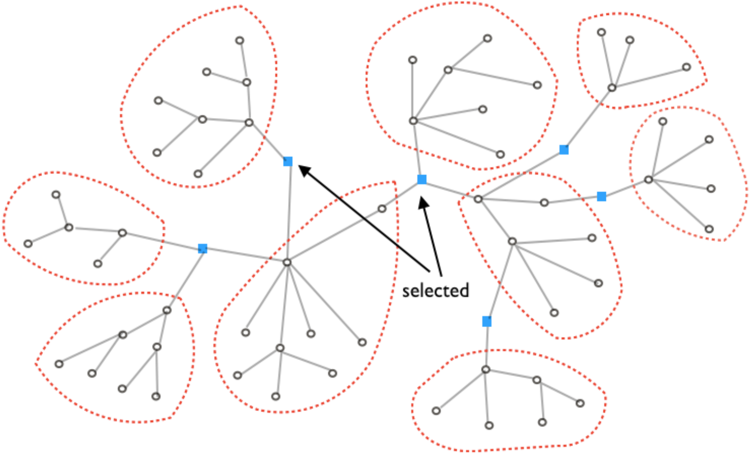
Let T be a spanning tree on n vertices. For any integer $2 \leq k \leq n$, we can select a subset of $\leq 3k - 5$ vertices whose removal partitions T into subtrees of size $\leq n/k$.



Tree partition lemma

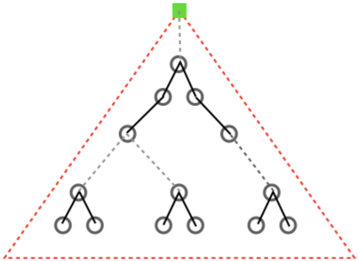
Lemma

Let T be a spanning tree on n vertices. For any integer $2 \leq k \leq n$, we can select a subset of $\leq 3k - 5$ vertices whose removal partitions T into subtrees of size $\leq n/k$.

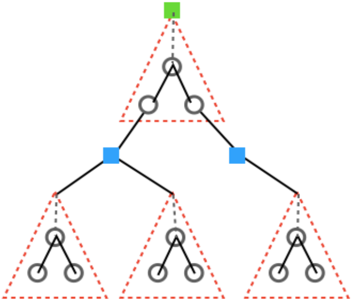


Partition single-source shortest paths trees

Data structure: For every $s \in V$, let T_s be the single-source shortest paths tree rooted at s, and apply Tree-partition Lemma on T_s with parameter n/L.



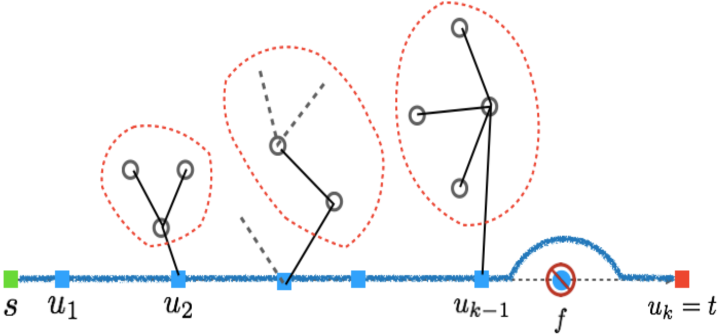
An SSSP tree.



Blue squares are selected vertices.

Sparser table:

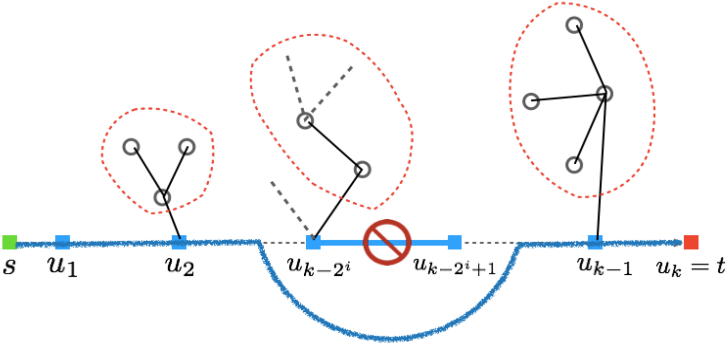
(i) For every selected t, $\text{dist}_{G \setminus \{f\}}(s, t)$.



$u_1, u_2, \cdots, u_k$ are selected in SSSP tree T_s.

Sparser table:

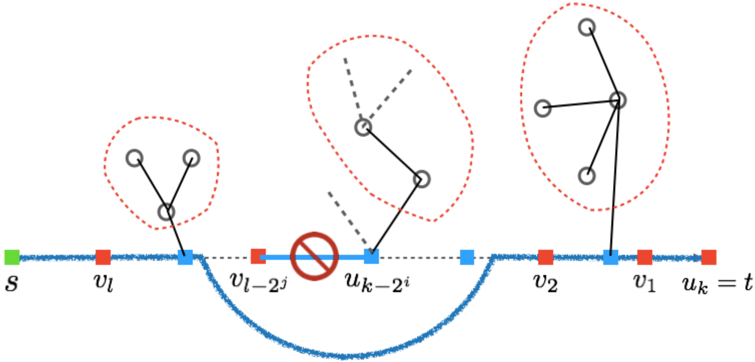
(ii) For every selected t, $\text{dist}_{G\setminus[u_{k-2^i},u_{k-2^i+1}]}(s,t)$.



$u_1, u_2, \cdots, u_k$ are selected in SSSP tree T_s.

Sparser table:

(iii) For every selected t, $\text{dist}_{G\backslash[v_{l-2^j}, u_{k-2^i}]}(s,t)$.



$u_1, u_2, \cdots, u_k$ are selected in SSSP tree T_s.
$v_1, v_2, \cdots, v_l$ are selected in **reverse** SSSP tree $\widehat{\mathrm{T}}_t$.

Sparser table

Space complexity: $O(n^2 \log^2 n / L)$

If $L = \log^2 n$, then the data structure so far occupies space $O(n^2)$.

t, f are not in the same subtree

Query algorithm: On input (s, t, f).

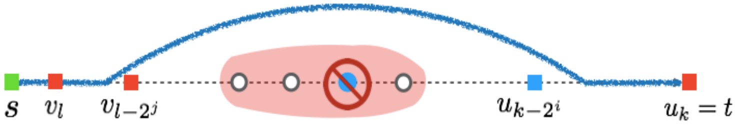
First consider when a **subtree-path** containing f is skipped over.

t, f are not in the same subtree

Query algorithm: On input (s, t, f).

First consider when a **subtree-path** containing f is skipped over.

(1) Paths that skip interval $[v_{l-2^j}, u_{k-2^i}]$ entirely.

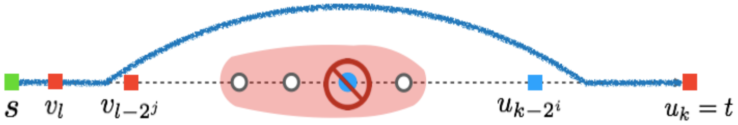


t, f are not in the same subtree

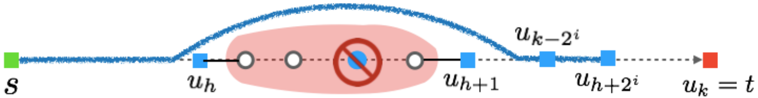
Query algorithm: On input (s, t, f).

First consider when a **subtree-path** containing f is skipped over.

(1) Paths that skip interval $[v_{l-2^j}, u_{k-2^i}]$ entirely.



(2) Paths that pass through u_{k-2^i}.

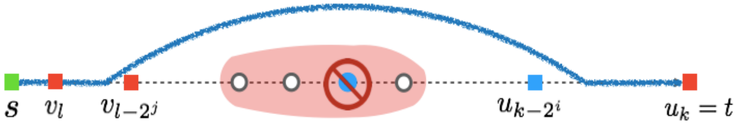


t, f are not in the same subtree

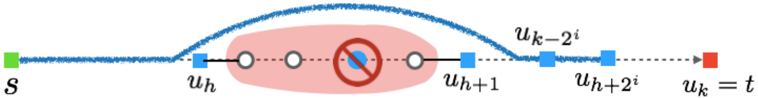
Query algorithm: On input (s, t, f).

First consider when a **subtree-path** containing f is skipped over.

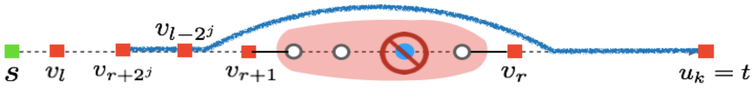
(1) Paths that skip interval $[v_{l-2^j}, u_{k-2^i}]$ entirely.



(2) Paths that pass through u_{k-2^i}.



(3) Paths that pass through v_{l-2^j}.



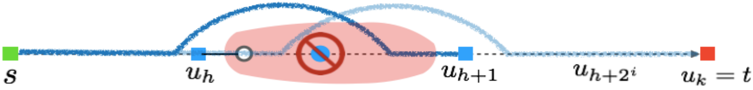
t, f are not in the same subtree

Query algorithm: On input (s, t, f). Now consider when the replacement path enters a **subtree-path**. For simplicity we only discuss two easy cases.

t, f are not in the same subtree

Query algorithm: On input (s, t, f). Now consider when the replacement path enters a **subtree-path**. For simplicity we only discuss two easy cases.

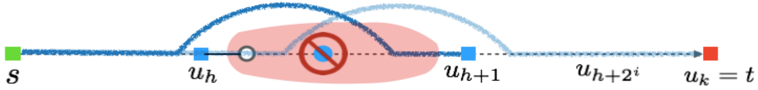
(1) Entry from the right is the easier case.



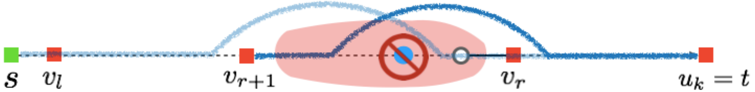
t, f are not in the same subtree

Query algorithm: On input (s, t, f). Now consider when the replacement path enters a **subtree-path**. For simplicity we only discuss two easy cases.

(1) Entry from the right is the easier case.



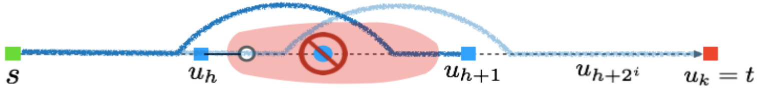
(2) Entry from the left is the easier case.



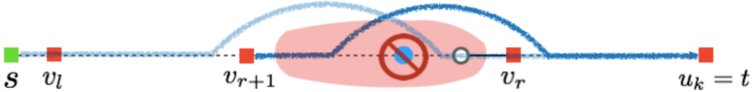
t, f are not in the same subtree

Query algorithm: On input (s, t, f). Now consider when the replacement path enters a **subtree-path**. For simplicity we only discuss two easy cases.

(1) Entry from the right is the easier case.



(2) Entry from the left is the easier case.



Now what if t, f are in the same subtree?

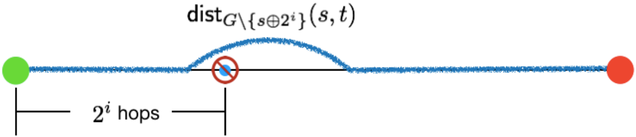
An $O(n^2 \log\log n)$-space DSO

Data structure: A truncated version of the sparse table in [DT02].

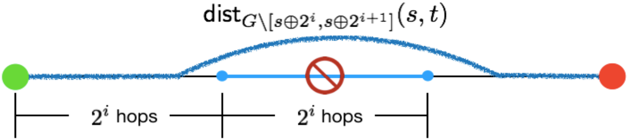
An $O(n^2 \log\log n)$-space DSO

Data structure: A truncated version of the sparse table in [DT02].

(i) For any $i \leq \log(4L)$, precompute $\text{dist}_{G\setminus\{s\oplus 2^i\}}(s,t)$ and $\text{dist}_{G\setminus\{t\ominus 2^i\}}(s,t)$.



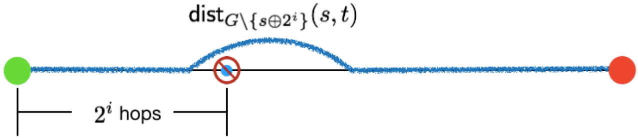
(ii) For any $i \leq \log(4L)$, precompute $\text{dist}_{G\setminus[s\oplus 2^i, s\oplus 2^{i+1}]}(s,t)$ and $\text{dist}_{G\setminus[t\ominus 2^{i+1}, t\ominus 2^i]}(s,t)$.



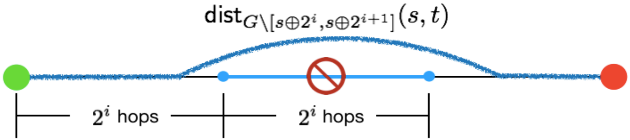
An $O(n^2 \log\log n)$-space DSO

Data structure: A truncated version of the sparse table in [DT02].

(i) For any $i \leq \log(4L)$, precompute $\mathrm{dist}_{G \setminus \{s \oplus 2^i\}}(s, t)$ and $\mathrm{dist}_{G \setminus \{t \ominus 2^i\}}(s, t)$.



(ii) For any $i \leq \log(4L)$, precompute $\mathrm{dist}_{G \setminus [s \oplus 2^i, s \oplus 2^{i+1}]}(s, t)$ and $\mathrm{dist}_{G \setminus [t \ominus 2^{i+1}, t \ominus 2^i]}(s, t)$.



Space complexity: $O(n^2 \log L)$

An $O(n^2 \log\log n)$-space DSO

Query algorithm: Suppose t, f are in the same subtree.

Analysis from [DT02] still works!

Query time: $O(1)$

How to obtain an $O(n^2)$-space DSO?

Assume an $\Omega(\log n)$-RAM model.

Rough idea

How to obtain an $O(n^2)$-space DSO?

Assume an $\Omega(\log n)$-RAM model.

Rough idea

- Further partition subtrees into even smaller ones.

How to obtain an $O(n^2)$-space DSO?

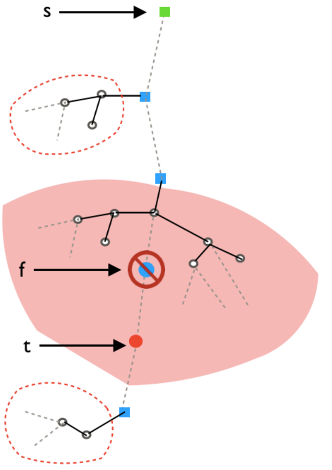
Assume an $\Omega(\log n)$-RAM model.

Rough idea

- Further partition subtrees into even smaller ones.
- Apply the bit-tricks ("Four Russians").

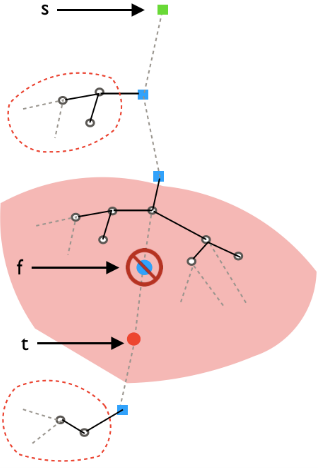
Two-level partition of SSSP trees

Hard cases: t, f lie in the same subtree.

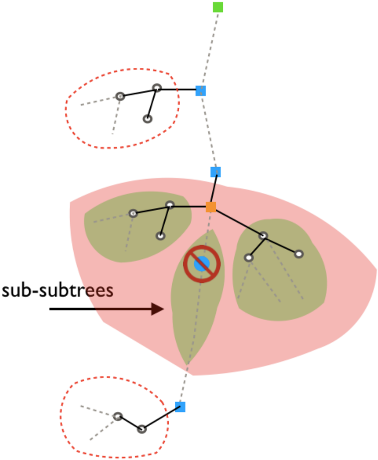


Two-level partition of SSSP trees

Hard cases: t, f lie in the same subtree.

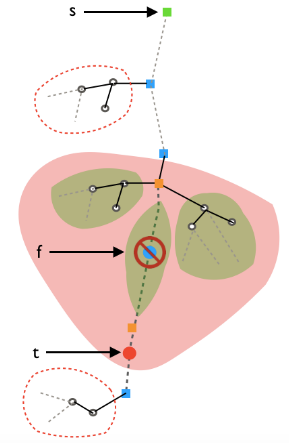


Two-level partition: Tree partition with $L' = \log^2 L$.



Two-level partition of SSSP trees

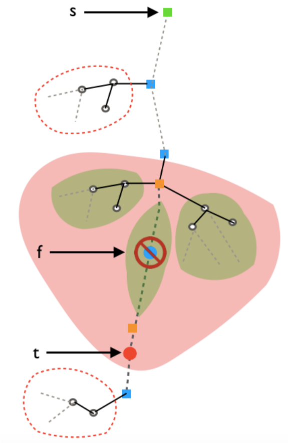
Easy cases: If t, f lie in different sub-subtrees, solve it using a truncated version of the data structure in previous slides.

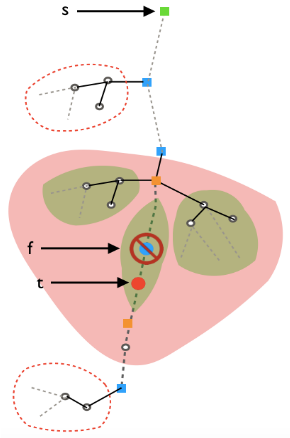


Two-level partition of SSSP trees

Easy cases: If t, f lie in different sub-subtrees, solve it using a truncated version of the data structure in previous slides.

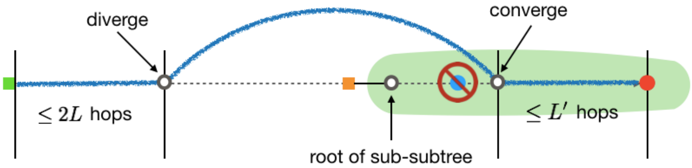
Hard cases: If t, f lie in the same sub-subtree, solve it later using the tabulation technique ("Four Russians").





Conclude with bit-tricks

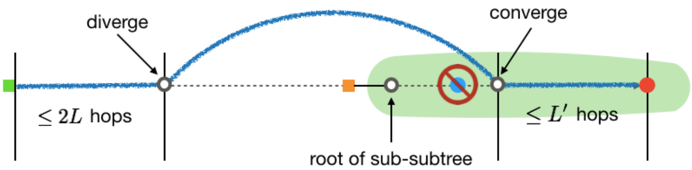
Use an $o(n)$-space table to store **all** replacement paths of the following kind; don't care about other replacement paths.



- The replacement path is encoded as:
  (1) # hops before diverge; (2) # hops after converge.

Conclude with bit-tricks

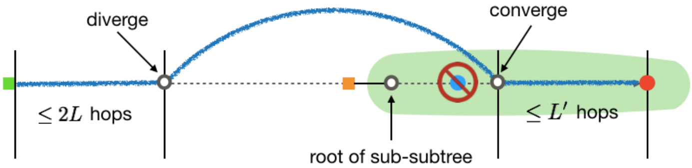
Use an $o(n)$-space table to store **all** replacement paths of the following kind; don't care about other replacement paths.



- The replacement path is encoded as: (1) # hops before diverge; (2) # hops after converge.
- (1)(2) can be stored in $\log(2L \cdot L')$-bits.

Conclude with bit-tricks

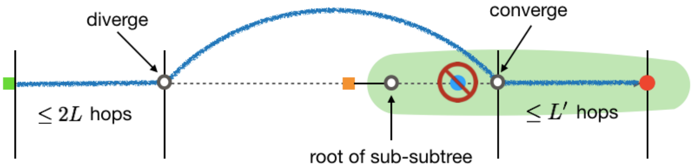
Use an $o(n)$-space table to store **all** replacement paths of the following kind; don't care about other replacement paths.



- Configuration of this sub-subtree can be stored in $O((L')^2 \cdot \log(2L \cdot L')) = O(\log \log^5 n)$ bits.

Conclude with bit-tricks

Use an $o(n)$-space table to store **all** replacement paths of the following kind; don't care about other replacement paths.



- Configuration of this sub-subtree can be stored in $O((L')^2 \cdot \log(2L \cdot L')) = O(\log\log^5 n)$ bits.
- All possible sub-subtree configurations can be stored in an indexable table of size $2^{O(\log\log^5 n)} = o(n)$.

Thanks for listening

References I

Aaron Bernstein and David Karger.
Improved distance sensitivity oracles via random sampling.
In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 34–43. Society for Industrial and Applied Mathematics, 2008.

Aaron Bernstein and David Karger.
A nearly optimal oracle for avoiding failed vertices and edges.
In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 101–110. ACM, 2009.

Camil Demetrescu and Mikkel Thorup.
Oracles for distances avoiding a link-failure.
In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 838–843. Society for Industrial and Applied Mathematics, 2002.

References II

Fabrizio Grandoni and Virginia Vassilevska Williams.
Improved distance sensitivity oracles via fast single-source replacement paths.
In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 748–757. IEEE, 2012.

Mihai Patrascu, Liam Roditty, and Mikkel Thorup.
A new infinity of distance oracles for sparse graphs.
In *Foundations of Computer Science (focs), 2012 Ieee 53rd Annual Symposium on*, pages 738–747. IEEE, 2012.