Dynamic And Fault Tolerant Algorithms

Manoj Gupta, IIT Gandhinagar

The Problem

- The graph is changing
- Maintain solutions of graph theoretic / optimization problems more efficiently than recomputing from scratch

The Problem

- The graph is changing
- Maintain solutions of graph theoretic / optimization problems more efficiently than recomputing from scratch

Types of Changes

- Incremental/Decremental: only insertions/deletions of edges
- Fully dynamic: both insertions and deletions

Dynamic Graphs

The Problem

- The graph is changing
- Maintain solutions of graph theoretic / optimization problems more efficiently than recomputing from scratch

Types of Changes

- Incremental/Decremental: only insertions/deletions of edges
- Fully dynamic: both insertions and deletions

Performance Evaluation

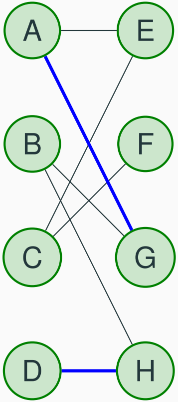
Update time: The time taken to Update the solution

Examples

- Connectivity
- Single source shortest path
- All pair shortest path
- Strongly connected components
- Minimum Spanning Tree
- Topological Sorting

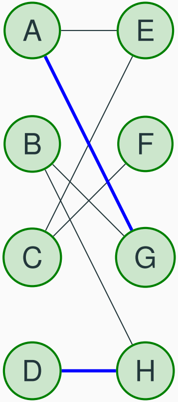
Some Definitions

- A matching in a graph is a set of edges M such that no two edges in M share a common endpoint



Some Definitions

- A matching in a graph is a set of edges M such that no two edges in M share a common endpoint
- We can find a $(1+\epsilon)$-approximate matching in a static unweighted graph in $O\left(\frac{m}{\epsilon}\right)$ time (Micali and Vazirani, 1980)



The Problem

Problem

Maintain approximate maximum matching in a dynamic graph

The Problem

Problem

Maintain approximate maximum matching in a dynamic graph

Model

- At each update step an edge can be added or deleted from the graph
- Compute the matching quickly after each update

In this talk [G. and Peng (FOCS 2013)]

Maintain $(1+\epsilon)$-approximate maximum matching in $O\left(\frac{\sqrt{m}}{\epsilon^2}\right)$ update time

Key Idea

Can we find a smaller subgraph G' of G such that the size of the maximum matching in G' is same as the size of maximum matching in G?

Key Idea

Can we find a smaller subgraph G' of G such that the size of the maximum matching in G' is same as the size of maximum matching in G?

Answer

Yes : If you have a approximate vertex cover of the graph

$(1+\epsilon)$-Approximate matching

- Assume that we have an oracle access to the vertex cover V_{cover} at every update step

$(1+\epsilon)$-Approximate matching

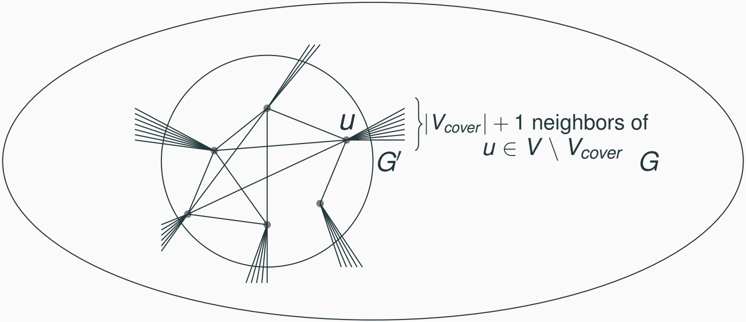
- Assume that we have an oracle access to the vertex cover V_{cover} at every update step
- Use the algorithm of Neiman and Solomon(STOC 2013): maintain 3/2-approximate matching in a dynamic graph

$(1+\epsilon)$-Approximate matching

- Assume that we have an oracle access to the vertex cover V_{cover} at every update step
- Use the algorithm of Neiman and Solomon(STOC 2013): maintain 3/2-approximate matching in a dynamic graph
- Report all the vertices in the matching as V_{cover}

Core Graph

- Include all the edges within the vertex cover
- For each $u \in V_{cover}$, include at most $|V_{cover}| + 1$ neighbors outside the vertex cover



Theorem

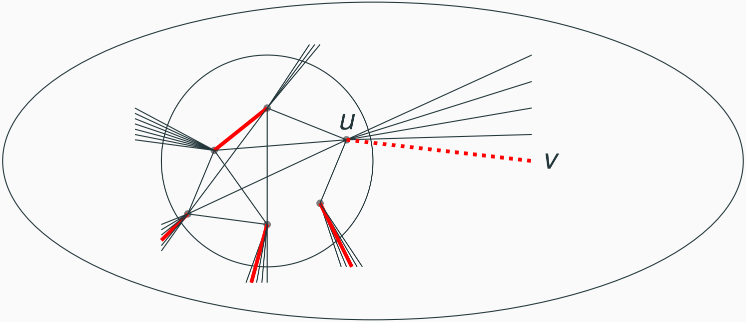
The size of maximum matching in core graph G' is same as the size of maximum matching in G

Proof

Among all maximum matchings in G, let M' be one that uses the maximum number of edges in G'.

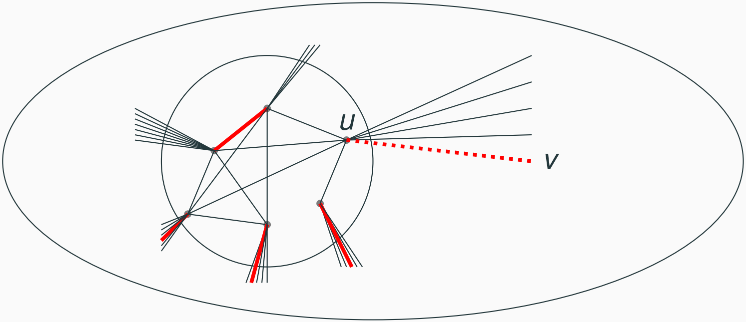
Proof

Among all maximum matchings in G, let M' be one that uses the maximum number of edges in G'.



Proof

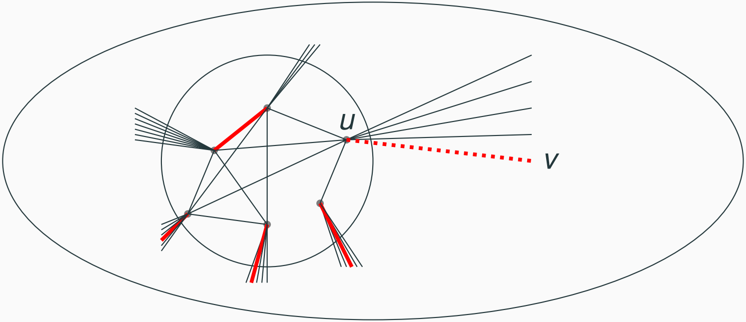
Among all maximum matchings in G, let M' be one that uses the maximum number of edges in G'.



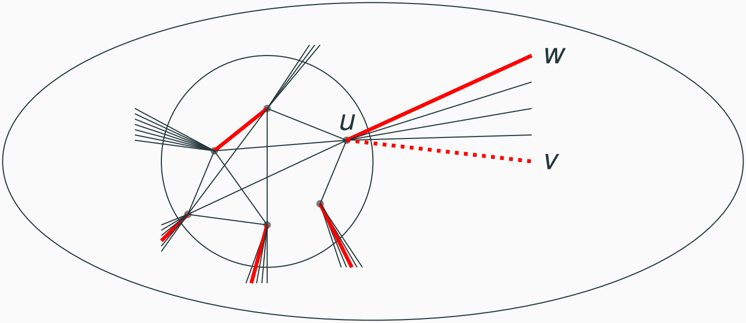
- By construction, u has $|V_{cover}|+1$ neighbors outside the vertex cover in G'.

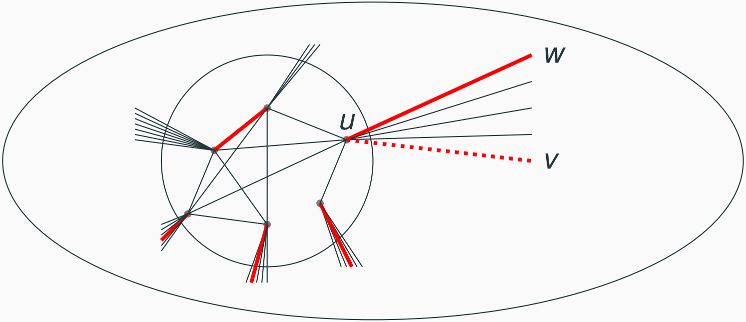
Proof

Among all maximum matchings in G, let M' be one that uses the maximum number of edges in G'.

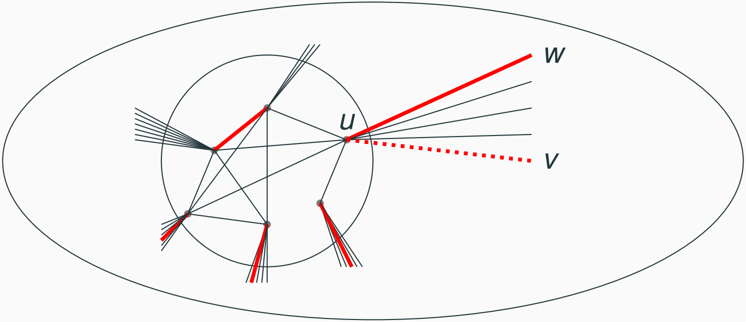


- By construction, u has $|V_{cover}|$+1 neighbors outside the vertex cover in G'.
- Atleast one of them is unmatched in M', because $|M'| \leq$ size of any vertex cover


w
u
v



- $M'' \leftarrow M' \setminus (u, v) \cup (u, w)$



- $M'' \leftarrow M' \setminus (u, v) \cup (u, w)$
- M'' is a maximum matching and its intersection with G' is larger than that of M'
- A contradiction

Partial Algorithm

- Construct a core graph G' of G
- Find a $(1 + \epsilon/2)$ approximate matching M in G'

Partial Algorithm

- Construct a core graph G' of G
- Find a $(1 + \epsilon/2)$ approximate matching M in G'

Size of Core Graph G'

- Size of G' is $min\{m, O(|V_{cover}|^2)\}$

Partial Algorithm

- Construct a core graph G' of G
- Find a $(1+\epsilon/2)$ approximate matching M in G'

Size of Core Graph G'

- Size of G' is $min\{m, O(|V_{cover}|^2)\}$
- $|V_{cover}| = 2|M_{3/2}| \leq 2|M^*|$ and $|M^*| \leq (1+\epsilon/2)|M|$

Partial Algorithm

- Construct a core graph G' of G
- Find a $(1 + \epsilon/2)$ approximate matching M in G'

Size of Core Graph G'

- Size of G' is $min\{m, O(|V_{cover}|^2)\}$
- $|V_{cover}| = 2|M_{3/2}| \leq 2|M^*|$ and $|M^*| \leq (1 + \epsilon/2)|M|$
- $|V_{cover}| \leq 2(1 + \epsilon/2)|M|$

Partial Algorithm

- Construct a core graph G' of G
- Find a $(1 + \epsilon/2)$ approximate matching M in G'

Size of Core Graph G'

- Size of G' is $min\{m, O(|V_{cover}|^2)\}$
- $|V_{cover}| = 2|M_{3/2}| \leq 2|M^*|$ and $|M^*| \leq (1 + \epsilon/2)|M|$
- $|V_{cover}| \leq 2(1 + \epsilon/2)|M|$
- Size of G' is $min\{m, O(|M|^2)\}$

Partial Algorithm

- Construct a core graph G' of G
- Find a $(1 + \epsilon/2)$ approximate matching M in G'

Size of Core Graph G'

- Size of G' is $min\{m, O(|V_{cover}|^2)\}$
- $|V_{cover}| = 2|M_{3/2}| \leq 2|M^*|$ and $|M^*| \leq (1 + \epsilon/2)|M|$
- $|V_{cover}| \leq 2(1 + \epsilon/2)|M|$
- Size of G' is $min\{m, O(|M|^2)\}$
- Time to find a matching in G' is $O\Big(\frac{min\{m, |M|^2\}}{\epsilon}\Big)$

$(1+\epsilon)$-approximate matching

Algorithm

- Construct a core graph G' of G
- Find a $(1+\epsilon/2)$ approximate matching M in G'
- Use this matching for the next $\epsilon|M|/2$ update steps

$(1+\epsilon)$-approximate matching

Algorithm

- Construct a core graph G' of G
- Find a $(1+\epsilon/2)$ approximate matching M in G'
- Use this matching for the next $\epsilon|M|/2$ update steps

Analysis

- M can reduce by atmost 1 wrt maximum matching after each update step

$(1+\epsilon)$-approximate matching

Algorithm

- Construct a core graph G' of G
- Find a $(1+\epsilon/2)$ approximate matching M in G'
- Use this matching for the next $\epsilon|M|/2$ update steps

Analysis

- M can reduce by atmost 1 wrt maximum matching after each update step
- After $\epsilon|M|/2$ steps, the matching M is $(1+\epsilon)$-approximate

Running time

- If $|M| \geq \sqrt{m}$, the amortized update time is $O\Big(\frac{m\epsilon^{-1}}{\epsilon|M|}\Big) = O\Big(\frac{\sqrt{m}}{\epsilon^2}\Big)$ $\qquad (min\{m, |M|^2\})$

Running time

- If $|M| \geq \sqrt{m}$, the amortized update time is $O\Big(\frac{m\epsilon^{-1}}{\epsilon|M|}\Big) = O\Big(\frac{\sqrt{m}}{\epsilon^2}\Big) \qquad (min\{m, |M|^2\})$
- If $|M| < \sqrt{m}$, the amortized update time is $O\Big(\frac{|M|^2\epsilon^{-1}}{\epsilon|M|}\Big) = O\Big(\frac{|M|}{\epsilon^2}\Big) = O\Big(\frac{\sqrt{m}}{\epsilon^2}\Big) \qquad (min\{m, |M|^2\})$

Theorem

Maintain $(1+\epsilon)$-approximate maximum matching in $O\left(\frac{\sqrt{m}}{\epsilon^2}\right)$ update time

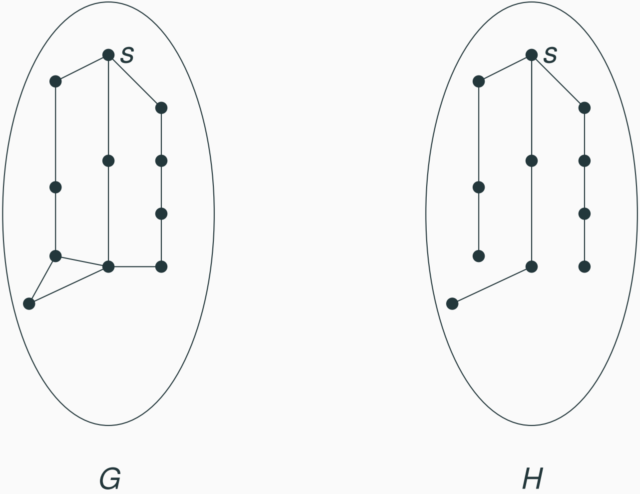
Make your own problem

- Incremental, Decremental or Fully Dynamic
- Unweighted or Weighted graphs
- Approximate matching or maximum matching
- Randomized or deterministic
- Worst case update time or Amortized running time
- Directed or Undirected graph

Fault Tolerant Algorithms

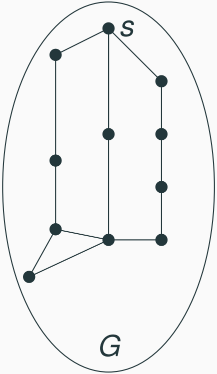
A Fault Tolerant System continues to perform at a desired level in spite of failures in some of its components.

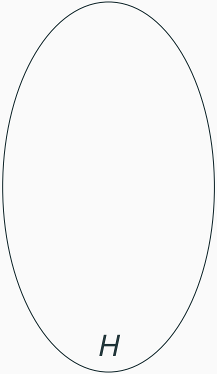
Fault Tolerant Subgraph Problem



Find a subgraph H of G such that the shortest path from s to all other vertices are preserved in H.

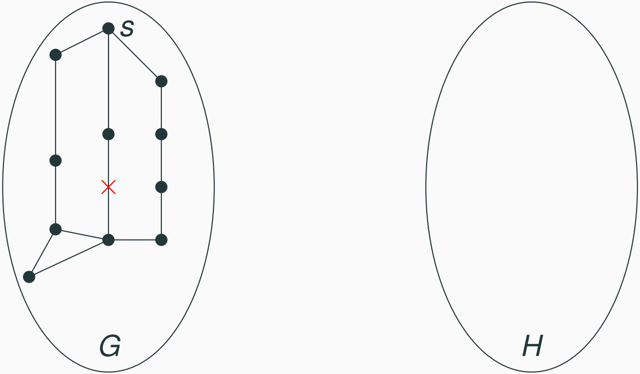
Fault Tolerant Subgraph Problem





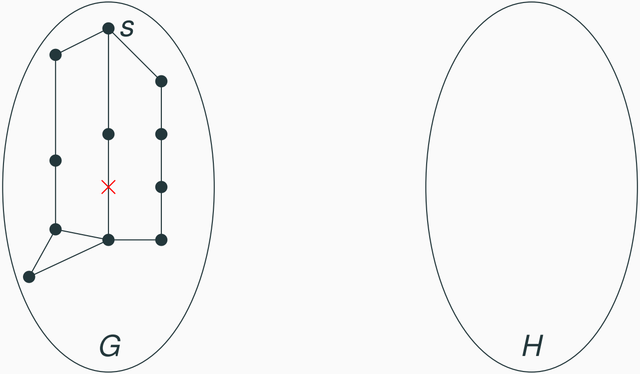
- Find a subgraph H of G such that the shortest path from s to all other vertices avoiding a single edge are preserved in H.

Fault Tolerant Subgraph Problem



- Find a subgraph H of G such that the shortest path from s to all other vertices avoiding a single edge are preserved in H.

Fault Tolerant Subgraph Problem



- Find a subgraph H of G such that the shortest path from s to all other vertices avoiding a single edge are preserved in H.
- Parter and Peleg [ESA 2013] showed that $O(n^{3/2})$ edges are both sufficient and necessary.

Fault Tolerant Algorithm

Input $\Longrightarrow$ **Data-Structure**

- Preprocess the input to build a data-structure.
- Preprocessing is free.

Input $\Longrightarrow$ **Data-Structure** $\Longleftrightarrow$ **Query Algorithm**

- Design a query algorithm that will use your data-structure to answer queries efficiently.

Fault Tolerant Algorithm

Input $\Longrightarrow$ **Data-Structure** $\Longleftrightarrow$ **Query Algorithm**

- Given a graph G design a data-structure that can answer the following query: find the length of shortest path from a source s to v where $v \in V$.

Input $\Longrightarrow$ **Data-Structure** $\Longleftrightarrow$ **Query Algorithm**

- Given a graph G design a data-structure that can answer the following query: find the length of shortest path from a source s to v where $v \in V$.
- Store the distances from s in $O(n)$ space, so that queries can be answered in $O(1)$ time.

Our problem

- Given an undirected and unweighted graph G, design a data-structure that can find the shortest path from a source node s to any destination node avoiding a single edge.

Our problem

- Given an undirected and unweighted graph G, design a data-structure that can find the shortest path from a source node s to any destination node avoiding a single edge.
- Formally, the query algorithm should answer the following query quickly, $\textsc{Query}(s, t, e)$: Find the length of the shortest path from s to $t \in V$ avoiding the edge e.
- Such a (data-structure + query algorithm) is known as distance oracle.

In this talk [G. and Singh, ICALP 2018]

We present a distance oracle of size $\tilde{O}(n^{3/2})$ that can answer queries in $\tilde{O}(1)$ time.

1. Sample a set of terminals $\mathcal{T}$ of size $\tilde{O}(\sqrt{n})$ vertices.

1. Sample a set of terminals $\mathcal{T}$ of size $\tilde{O}(\sqrt{n})$ vertices.
2. With a high probability, on any *st* path, there exists a vertex $t_s \in \mathcal{T}$ such that $|t_s t| = \tilde{O}(\sqrt{n})$.

Near case: $e \in t_s t$

Far case: $e \in s t_s$

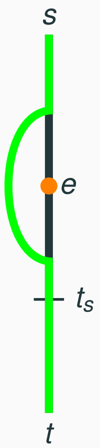
The Near Case

1. Store all replacement paths that avoid edges in $t_s t$.
2. Number of shortest paths stored (for a fixed t) is $|t_s t| = \tilde{O}(\sqrt{n})$

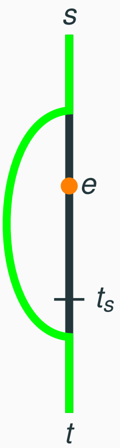
The Near Case

1. Store all replacement paths that avoid edges in $t_s t$.
2. Number of shortest paths stored (for a fixed t) is $|t_s t| = \tilde{O}(\sqrt{n})$
3. The size of the data-structure for a fixed t is $\tilde{O}(\sqrt{n})$.
4. The total size of the data-structure is $\tilde{O}(n^{3/2})$.

The Far Case



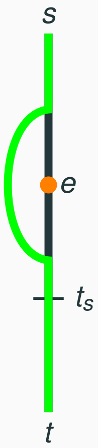
Replacement path passes through t_s



Replacement path avoids t_s

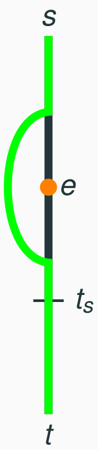
Replacement path passes through t_s

1. Store the length of the shortest path from s to $t_s \in \mathcal{T}$ avoiding each edge on st_s path.
2. The space taken = # terminals $\times$# edges on st_s path
   = $\tilde{O}(\sqrt{n}) \times n = \tilde{O}(n^{3/2})$

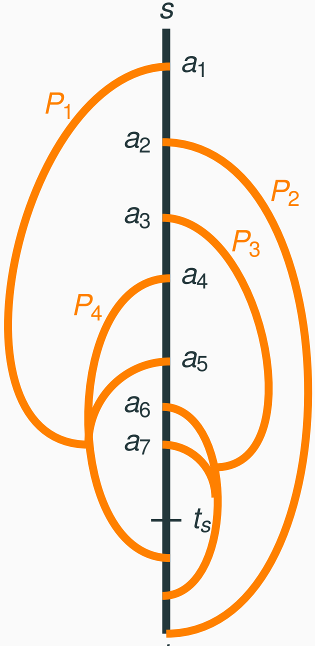


Replacement path passes through t_s

1. Store the length of the shortest path from s to $t_s \in \mathcal{T}$ avoiding each edge on st_s path.
2. The space taken = # terminals $\times$# edges on st_s path
   $= \tilde{O}(\sqrt{n}) \times n = \tilde{O}(n^{3/2})$
3. Store the length of the shortest path from $t_s \in \mathcal{T}$ to $t \in V$.
4. The space taken = # terminals $\times$# vertices
   $= \tilde{O}(\sqrt{n}) \times n = \tilde{O}(n^{3/2})$



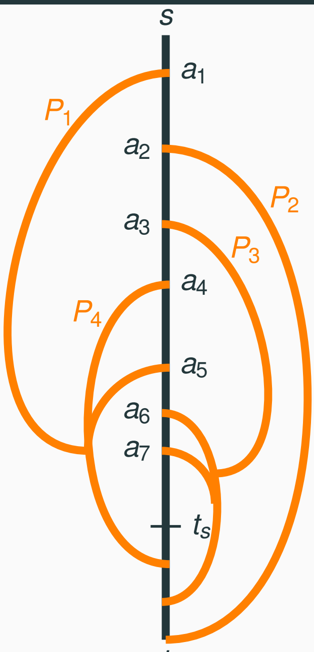
The replacement path avoids t_s



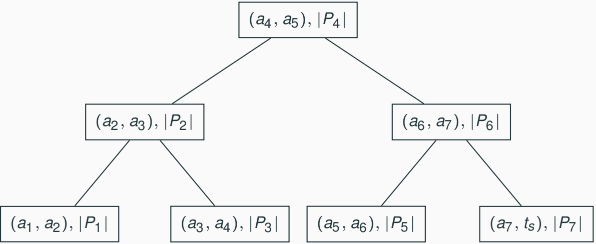
The replacement path avoids t_s

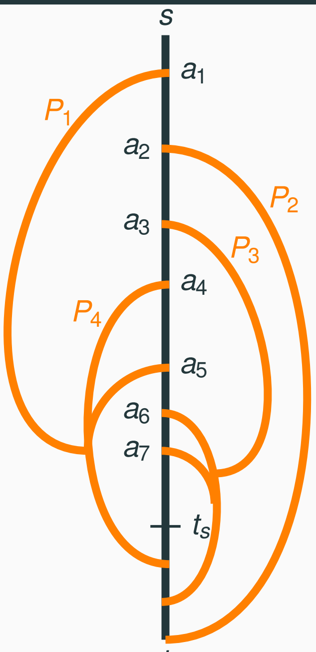
Main Technical Result

The total number of replacement paths from s to t that avoid t_s is $O(\sqrt{n})$.



The replacement path avoids t_s





Size of our last data-structure

Since the size of the BST is $O(\sqrt{n})$ (for a fixed t), the total size of the data-structure is $O(n^{3/2})$.

Main Theorem

There exists a distance oracle of size $\tilde{O}(n^{3/2})$ that can answer queries in $\tilde{O}(1)$ time.

Main Theorem

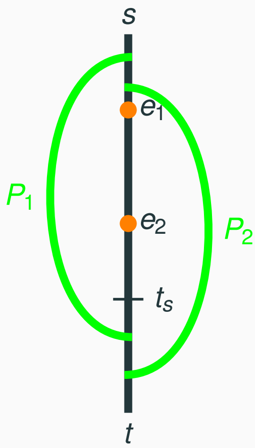
There exists a distance oracle of size $\tilde{O}(n^{3/2})$ that can answer queries in $\tilde{O}(1)$ time.

Rest of the talk

The total number of replacement paths from s to t that avoid t_s is $O(\sqrt{n})$.

The replacement path avoids t_s

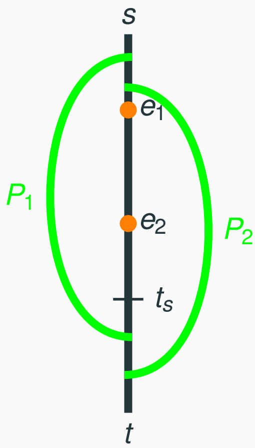
Few basic observations



The replacement path avoids t_s

Few basic observations

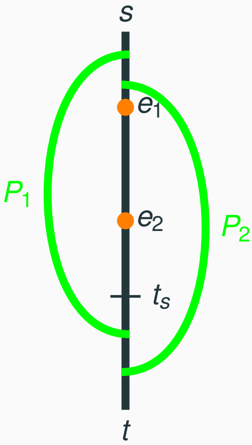
- Is the picture correct?



The replacement path avoids t_s

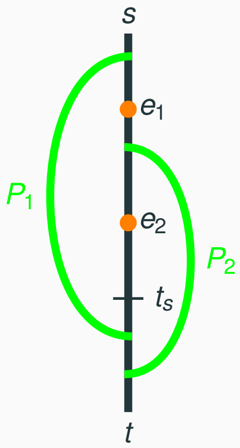
Few basic observations

- Is the picture correct?
- No, because if $|P_1| \leq |P_2|$, then the replacement path that avoids e_2 is also P_1.



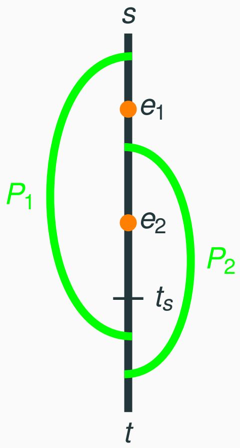
The replacement path avoids t_s

- Is the picture in the right correct?
- No, because if $|P_1| \leq |P_2|$, then the replacement path avoid e_2 is also P_1.
- $\mathcal{P}_1$: The lower replacement path will pass through the edge avoided by the upper replacement path.



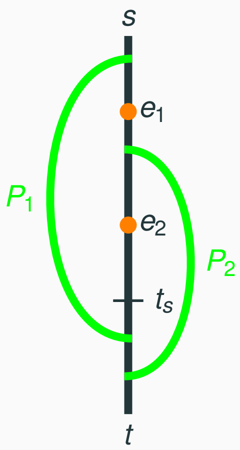
The replacement path avoids t_s

- Can $|P_2| \geq |P_1|$?



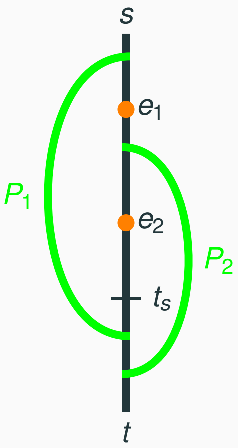
The replacement path avoids t_s

- Can $|P_2| \geq |P_1|$?
- No, because if $|P_2| \geq |P_1|$, then the replacement path avoiding e_2 is P_1.



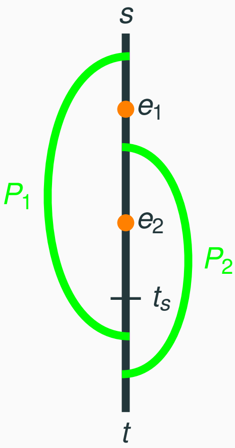
The replacement path avoids t_s

- Can $|P_2| \geq |P_1|$?
- No, because if $|P_2| \geq |P_1|$, then the replacement path avoiding e_2 is P_1.
- $\mathcal{P}_2$: The lower replacement path has length strictly less than the upper replacement path.



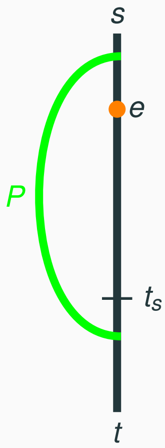
The replacement path avoids t_s

- Can $|P_2| \geq |P_1|$?
- No, because if $|P_2| \geq |P_1|$, then the replacement path avoiding e_2 is P_1.
- $\mathcal{P}_2$: The lower replacement path has length strictly less than the upper replacement path.
- Corollary: The length of these paths are distinct.

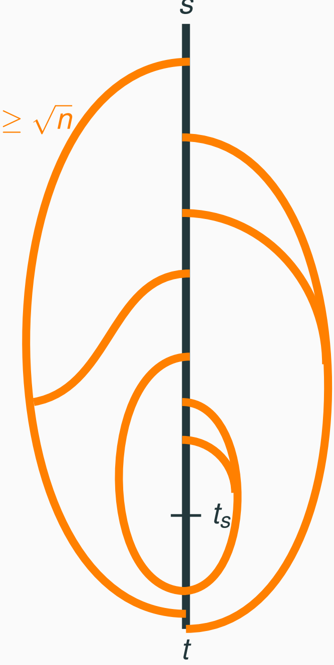


Some Definitions

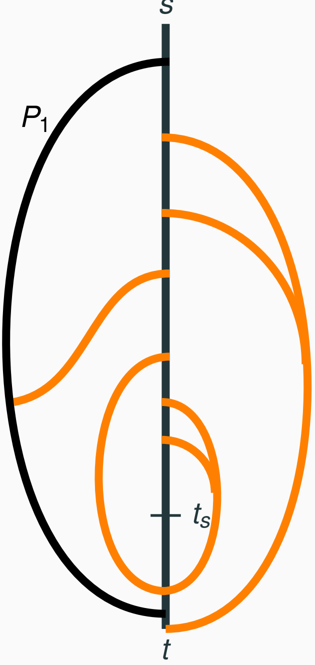
- Detour of a replacement path.
- Green path or formally $P \setminus st$



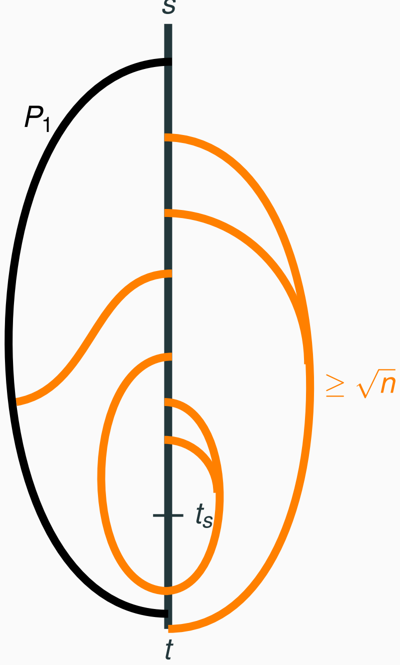
- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path



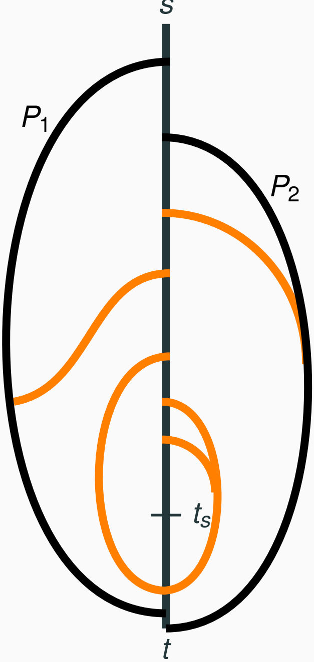
- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path



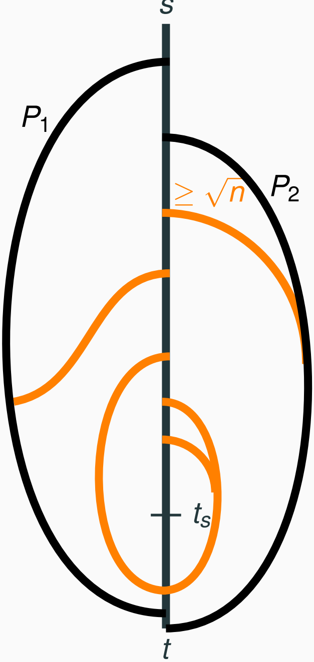
- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path



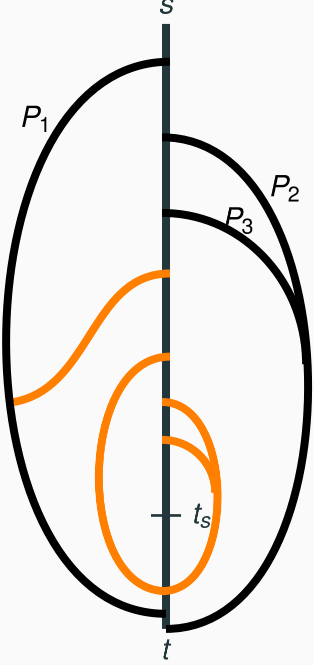
- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path



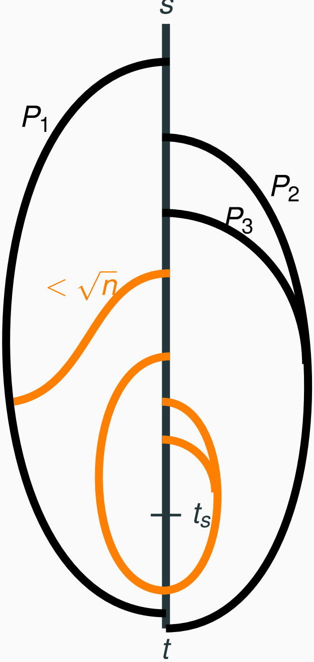
- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path



- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path

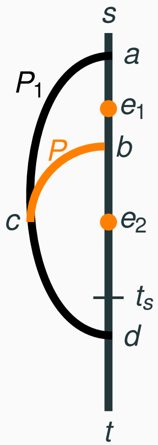


- Process replacement paths from top to bottom.
- Try to associate $\sqrt{n}$ unique vertices of the detour with each replacement path



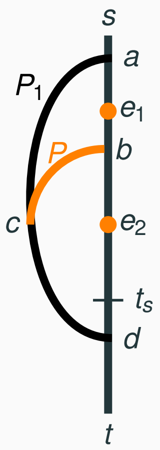
Bad case when $|bc| < \sqrt{n}$

- $|P_1| = |sa| + |ac| + |cd| + |dt|$



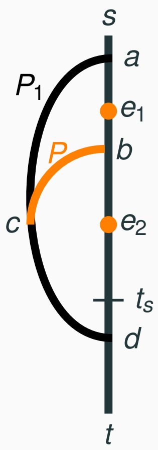
Bad case when $|bc| < \sqrt{n}$

- $|P_1| = |sa| + |ac| + |cd| + |dt|$
- But there is another path from s to t that avoids e_1.



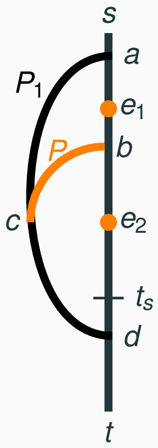
Bad case when $|bc| < \sqrt{n}$

- $|P_1| = |sa| + |ac| + |cd| + |dt|$
- But there is another path from s to t that avoids e_1.
- |sa| + |ac| + |cb| + |bt|



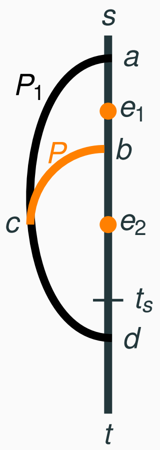
Bad case when $|bc| < \sqrt{n}$

- $|P_1| = |sa| + |ac| + |cd| + |dt|$
- But there is another path from s to t that avoids e_1.
- |sa| + |ac| + |cb| + |bt|
- Why is this a valid path avoiding e_1?.



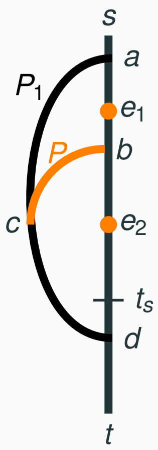
Bad case when $|bc| < \sqrt{n}$

- $|P_1| = |sa| + |ac| + |cd| + |dt|$
- But there is another path from s to t that avoids e_1.
- |sa| + |ac| + |cb| + |bt|
- Why is this a valid path avoiding e_1?.
  - cb is the part of the detour. So, it cannot pass through e_1.
  - Regarding bt, by $\mathcal{P}_1$, lower replacement path (P) passes through the edge avoided by the higher replacement path. So, b lies below e_1. Thus, bt doest not contain e_1.



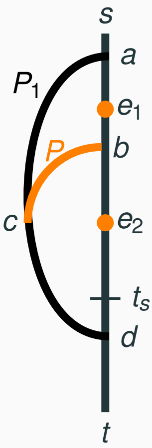
Bad case when $|bc| < \sqrt{n}$

- $|P_1| = |sa| + |ac| + |cd| + |dt|$
- But there is another path from s to t that avoids e_1.
- |sa| + |ac| + |cb| + |bt|
- Why is this a valid path avoiding e_1?.
    - cb is the part of the detour. So, it cannot pass through e_1.
    - Regarding bt, by $\mathcal{P}_1$, lower replacement path (P) passes through the edge avoided by the higher replacement path. So, b lies below e_1. Thus, bt doest not contain e_1.
- Since this path was not chosen by our algorithm as the replacement path, its length must be $>$ the length of P_1.



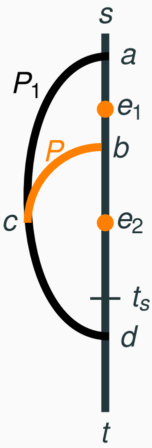
Bad case when $|bc| < \sqrt{n}$

- $|sa| + |ac| + |cd| + |dt| < |sa| + |ac| + |cb| + |bt|$



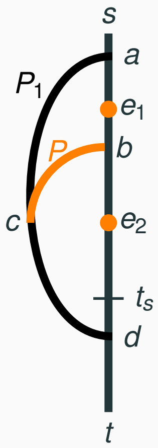
Bad case when $|bc| < \sqrt{n}$

- $|sa| + |ac| + |cd| + |dt| < |sa| + |ac| + |cb| + |bt|$
  $\implies |cd| + |dt| < |cb| + |bt|$



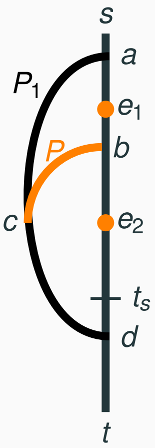
Bad case when $|bc| < \sqrt{n}$

- $|sa| + |ac| + |cd| + |dt| < |sa| + |ac| + |cb| + |bt|$
  $\implies |cd| + |dt| < |cb| + |bt|$
  $\implies |bc| + |cd| + |dt| < 2|cb| + |bt|$



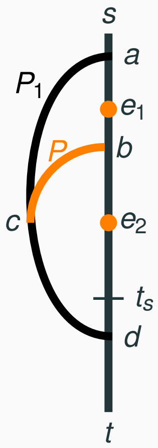
Bad case when $|bc| < \sqrt{n}$

- $|sa| + |ac| + |cd| + |dt| < |sa| + |ac| + |cb| + |bt|$
  $\implies |cd| + |dt| < |cb| + |bt|$
  $\implies |bc| \ + |cd| + |dt| < 2|cb| + |bt|$
  $\implies |bc| \ + |cd| + |dt| < 2\sqrt{n} + |bt|$



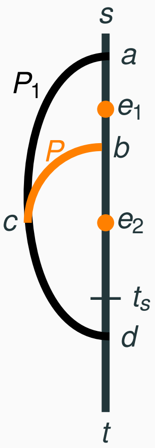
Bad case when $|bc| < \sqrt{n}$

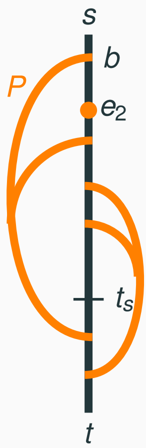
- $|sa| + |ac| + |cd| + |dt| < |sa| + |ac| + |cb| + |bt|$
  $\implies |cd| + |dt| < |cb| + |bt|$
  $\implies |bc| + |cd| + |dt| < 2|cb| + |bt|$
  $\implies |bc| + |cd| + |dt| < 2\sqrt{n} + |bt|$
- On the left hand side we have a replacement path from b to t avoiding e_2.



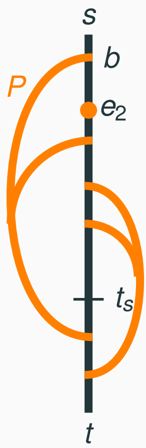
Bad case when $|bc| < \sqrt{n}$

- $|sa| + |ac| + |cd| + |dt| < |sa| + |ac| + |cb| + |bt|$
  $\implies |cd| + |dt| < |cb| + |bt|$
  $\implies |bc| + |cd| + |dt| < 2|cb| + |bt|$
  $\implies |bc| + |cd| + |dt| < 2\sqrt{n} + |bt|$
- On the left hand side we have a replacement path from b to t avoiding e_2.
- A good property of this replacement path is that its length is just $2\sqrt{n}$ greater than bt. We now exploit this property.

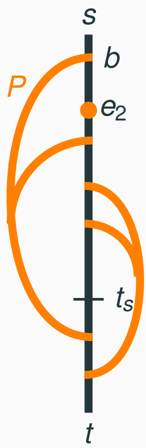



s
b
P
e_2
t_s
t

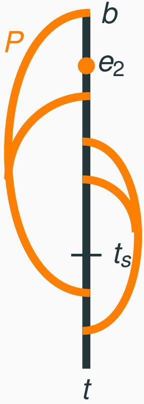
- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.



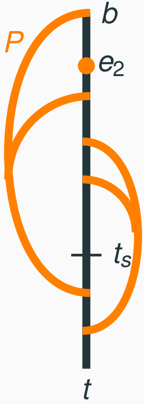
- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.
- We can thus assume that these paths are starting from b.



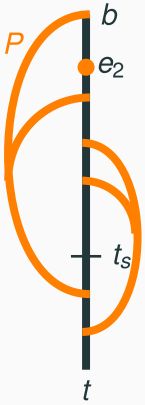
- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.
- We can thus assume that these paths are starting from b.
- By Property $\mathcal{P}_2$, the lower replacement path have length strictly less than the upper replacement path, that is P.



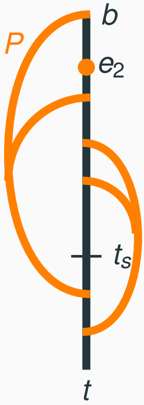
- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.
- We can thus assume that these paths are starting from b.
- By Property $\mathcal{P}_2$, the lower replacement path have length strictly less than the upper replacement path, that is P.
- The corollary of $\mathcal{P}_2$ states that length of these paths are distinct.



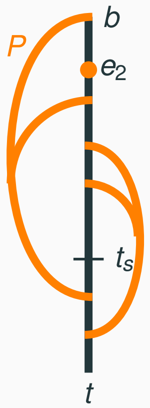
- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.
- We can thus assume that these paths are starting from b.
- By Property $\mathcal{P}_2$, the lower replacement path have length strictly less than the upper replacement path, that is P.
- The corollary of $\mathcal{P}_2$ states that length of these paths are distinct.
- Length of these path strictly lie in the range $[|bt|, |bt| + 2\sqrt{n}]$



- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.
- We can thus assume that these paths are starting from b.
- By Property $\mathcal{P}_2$, the lower replacement path have length strictly less than the upper replacement path, that is P.
- The corollary of $\mathcal{P}_2$ states that length of these paths are distinct.
- Length of these path strictly lie in the range $[|bt|, |bt| + 2\sqrt{n}]$



- By Property $\mathcal{P}_1$, all these lower replacement path pass through the edge avoided by P, that is e_2.
- We can thus assume that these paths are starting from b.
- By Property $\mathcal{P}_2$, the lower replacement path have length strictly less than the upper replacement path, that is P.
- The corollary of $\mathcal{P}_2$ states that length of these paths are distinct.
- Length of these path strictly lie in the range $[|bt|, |bt| + 2\sqrt{n}]$ $=O(\sqrt{n})$



Main Technical Result

The total number of replacement paths from s to t that avoid t_s is $O(\sqrt{n})$.

- We extend the above result to multiple sources.
- The extension, though technically involved, uses the strategy shown in this talk.

Open Problems

- What happens for two edge faults?
- For any general k edge faults?
- Fault tolerant all pair shortest path.

Thank You