Dynamic MSF with Subpolynomial Worst-case Update Time

Jaehyun Koo (koosaga)

January 9, 2021

Introduction

Dynamic MSF

Given a weighted graph, compute a minimum spanning forest.

Here we will talk about the harder variant: Given a weighted graph, we maintain a minumum spanning forest given edge insertion and deletion.

Dynamic MSF

Given a weighted graph, compute a minimum spanning forest.

Here we will talk about the harder variant: Given a weighted graph, we maintain a minumum spanning forest given edge insertion and deletion.

First known result is $O(\sqrt{m})$ per query. (Frederickson, 1985)

There is an $O((n+q)\log n)$ **offline** algorithm. (Eppstein, 1992)

Seminal $O(\log^4 n)$ **amortized** algorithm appeared. (HDLT, 2001)

This paper: $(n^{\log\log\log n/\log\log n})$ **worst-case** time per query, with randomization (**Las-Vegas algorithm**)

The term is $O(n^{o(1)})$: It is slower than any $O(\log^c n)$ and faster than any $O(n^c)$.

Dynamic MSF

Well of course, Dynamic MSF is an important problem, because MSF is an important primitive.

We can think of several primitives in graph: Shortest paths, Connectivity and global min cuts, mininum spanning trees...

Shortest paths are notoriously hard to solve in dynamic graph.

But others are not, and they are closely related to each other!

Dynamic MSF

Well of course, Dynamic MSF is an important problem, because MSF is an important primitive.

We can think of several primitives in graph: Shortest paths, Connectivity and global min cuts, mininum spanning trees...

Shortest paths are notoriously hard to solve in dynamic graph.

But others are not, and they are closely related to each other!

But probably it's not so interesting to solve in non-amortized time, because $O(\log^4 n)$ *algorithm is much simpler and practically better, I think...* Anyway, promises should be kept, let's go!

Decremental MSF solves Dynamic MSF

Decremental MSF solves Dynamic MSF

Decremental MSF is a easier variant of Dynamic MSF where edges are only deleted.

Note that **Incremental MSF** is trivial to solve in $O(\log N)$ time with LCT (amortized) or Top trees (worst case).

However, decremental MSF is not trivial.

Decremental MSF solves Dynamic MSF

Decremental MSF is a easier variant of Dynamic MSF where edges are only deleted.

Note that **Incremental MSF** is trivial to solve in $O(\log N)$ time with LCT (amortized) or Top trees (worst case).

However, decremental MSF is not trivial.

This paper's main work is to propose a new Las-Vegas Decremental MSF algorithm.

Then, it proves the reduction from Dynamic MSF to Decremental MSF, within a polylogarithmic factor.

Decremental MSF solves Dynamic MSF

Decremental MSF is a easier variant of Dynamic MSF where edges are only deleted.

Note that **Incremental MSF** is trivial to solve in $O(\log N)$ time with LCT (amortized) or Top trees (worst case).

However, decremental MSF is not trivial.

This paper's main work is to propose a new Las-Vegas Decremental MSF algorithm.

Then, it proves the reduction from Dynamic MSF to Decremental MSF, within a polylogarithmic factor.

The main work is very complicated, and requires a lot of new toolkits (Expander pruning, MSF decomposition, and crazy stuffs in between)

Decremental MSF solves Dynamic MSF

I want to finish this slide in at most 25 pages. Thus, my primary goal in the lecture is to introduce the reduction.

Decremental MSF solves Dynamic MSF

I want to finish this slide in at most 25 pages. Thus, my primary goal in the lecture is to introduce the reduction.

This reduction technique of Dynamic MSF to decremental MSF is mostly attributed to Holm et. al and Wulff-Nilsen, although there is some contribution from the author.

So probably this lecture isn't really related to the paper.

Decremental MSF solves Dynamic MSF

I want to finish this slide in at most 25 pages. Thus, my primary goal in the lecture is to introduce the reduction.

This reduction technique of Dynamic MSF to decremental MSF is mostly attributed to Holm et. al and Wulff-Nilsen, although there is some contribution from the author.

So probably this lecture isn't really related to the paper.

In 2001, Holm et. al proposed an $O(\log^2 N)$ amortized Decremental MSF algorithm (*The Notorious HDLT*).

It is a modification of online dynamic connectivity algo in the same paper.

So if you are aware of the online dynamic connectivity algo, probably this lecture conveys everything you need to know about Dynamic MSF.

The Theorem

Theorem. Suppose there is a Las-Vegas decremental MSF algorithm D with the following properties:

1. Graph have at most m' edges and have maximum degree at most 3.

2. We will perform at most $T(m')$ edge deletion queries.

3. Preprocess works on $t_{pre}(m', p)$ time and update works on $t_u(m', p)$ in $1 - p$ probability.

The Theorem

Theorem. Suppose there is a Las-Vegas decremental MSF algorithm D with the following properties:

1. Graph have at most m' edges and have maximum degree at most 3.

2. We will perform at most $T(m')$ edge deletion queries.

3. Preprocess works on $t_{pre}(m', p)$ time and update works on $t_u(m', p)$ in $1 - p$ probability.

Then, you can obtain a dynamic MSF algorithm with the following properties. Note that $p' = \Theta(p/\log k)$.

1. Graph have at most m' edges and have at most k non-tree edges.

2. Preprocess works on $t_{pre}(15k, p') + O(m \log^2 m)$ time.

3. You can add $\leq B$ edges or remove one edge each query. It works on $t'_u(m, k, B, p) =$
$O(\frac{B \log k}{k} \times t_{pre}(15k, p') + B \log^2 m + \frac{k \log k}{T(k)} + \log k \times t_u(15k, p'))$ time.

Strategy

We will first reduce the problem into a Decremental MSF algorithm with small non-tree edges.

Then we will reduce the problem into Decremental MSF algorithm with small size.

Then it's easy to add degree 3 condition (we will get back to this).

Strategy

We will first reduce the problem into a Decremental MSF algorithm with small non-tree edges.

Then we will reduce the problem into Decremental MSF algorithm with small size.

Then it's easy to add degree 3 condition (we will get back to this).

FAQ. Why small non-tree edges?

Answer: Indeed we will simply plug in the full graph. We don't use it here. The main part of paper uses this property, which we hopefully want to get back to (but probably fail to!!).

FAQ. Why bulk add when you can only have $B = 1$?

Answer: Ditto.

The first reduction

Main Lemma

Theorem. Suppose there is a Las-Vegas decremental MSF algorithm D with the following properties:

1. Graph have at most m edges and have at most k non-tree edges.

2. Preprocess works on $t_{pre}(m, k, p)$ time and update works on $t_u(m, k, p)$ in $1 - p$ probability.

Main Lemma

Theorem. Suppose there is a Las-Vegas decremental MSF algorithm D with the following properties:

1. Graph have at most m edges and have at most k non-tree edges.

2. Preprocess works on $t_{pre}(m, k, p)$ time and update works on $t_u(m, k, p)$ in $1 - p$ probability.

Then, for some $B \geq 5\lceil \log k \rceil$, you can obtain a dynamic MSF algorithm F with the following properties. Note that $p' = \Theta(p/\log k)$.

1. Preprocess works on $t'_{pre}(m, k, B, p) = T_{pre}(m, k, p') + O(m \log m)$ time.

2. You can add $\leq B$ edges or remove one edge each query. It works on $t'_u(m, k, B, p) =$
$O(\sum_{i=0}^{\lceil \log k \rceil} t_{pre}(m, min(2^{i+1}B, k), p')/2^i + B \log m + \log k \times t_u(m, k, p'))$ time.

Preprocessing

Basically we will build a sort of hierarchical machine that processes the queries.

Let $G = (V, E)$ the input of the graph. $F = MSF(G), N = E - F$, we have $|E| \leq m, |N| \leq k$ invariant.

Preprocessing

Basically we will build a sort of hierarchical machine that processes the queries.

Let $G = (V, E)$ the input of the graph. $F = MSF(G), N = E - F$, we have $|E| \leq m, |N| \leq k$ invariant.

Let $L = \lceil \log k \rceil$. In this algorithm we will maintain subgraph $G_{i,j}$ for $0 \leq i \leq L, 1 \leq j \leq 4$. Also we have special case $G_{L,0}$.

Let $N_{i,j} = E(G_{i,j}) - MSF(G_{i,j})$. The algorithm maintains an invariant $N = \bigcup_{i,j} N_{i,j}$, $|N_{i,j}| \leq min(2^i B, k)$.

Preprocessing

Basically we will build a sort of hierarchical machine that processes the queries.

Let $G = (V, E)$ the input of the graph. $F = MSF(G), N = E - F$, we have $|E| \leq m, |N| \leq k$ invariant.

Let $L = \lceil \log k \rceil$. In this algorithm we will maintain subgraph $G_{i,j}$ for $0 \leq i \leq L, 1 \leq j \leq 4$. Also we have special case $G_{L,0}$.

Let $N_{i,j} = E(G_{i,j}) - MSF(G_{i,j})$. The algorithm maintains an invariant $N = \bigcup_{i,j} N_{i,j}$, $|N_{i,j}| \leq min(2^i B, k)$.

Let $D_{i,j}$ be an instance of decremental MSF D maintaining $MSF(G_{i,j})$. We first let $G_{L,1} = G, G_{i,j} = \emptyset$. In the initialization we preprocess $D_{L,1}$ and construct a top tree $T(F)$ of F.

Preprocessing

We have a total of $4L + 5$ subgraph, four per each level, due to following reasons.

Each level is a processor that does a certain work over 2^i time period.

First two graph is a *queue*: It doesn't do anything, but it stacks a future job fetched from level $i - 1$.

Last two graph is where the calculation happens. Here we copy two subgraphs (in $O(1)$ time, just move pointers) and do the query processing.

After everything is done, result fetched from last two graph will go to level $i + 1$. That's why we have two graph (stack two and send one in 2^{i+1} time).

Update: Easy things

We will talk about the bulk insertion and deletion. After the both queries we will go through the process of *clean-up* that will be defined later.

Edge insertions. Let I be a set of inserting edges and R be a future clean-up candidate. For all edge $e = (u, v) \in I$, if u, v is not connected, $F \leftarrow F \cup \{e\}$.

Otherwise, find the maximum weight edge f in top tree $T(F)$ in $O(\log N)$ time. If $w(e) > w(f)$ add e to R. Otherwise replace e to f in F and add f to R.

Update: Easy things

We will talk about the bulk insertion and deletion. After the both queries we will go through the process of *clean-up* that will be defined later.

Edge insertions. Let I be a set of inserting edges and R be a future clean-up candidate. For all edge $e = (u, v) \in I$, if u, v is not connected, $F \leftarrow F \cup \{e\}$.

Otherwise, find the maximum weight edge f in top tree $T(F)$ in $O(\log N)$ time. If $w(e) > w(f)$ add e to R. Otherwise replace e to f in F and add f to R.

Edge removal. For all i, j, remove e from $G_{i,j}$ (with decremental algos). Let R_0 be a replacement edges over all $MSF(G_{i,j})$ Take the smallest weight edge that reconnects F (if exists). If it exists, replace e to f in F and remove f in R_0. Otherwise remove e. Now $R \leftarrow R_0$.

Then we will go through the clean-up process.

Update: Hard things

Clean up. Let R be a set of edges subject to clean-up. Let R' be a set of edges (to be defined). For each level $0 \leq i \leq L+1$ we will do *clean-up of level* i. For each level i:

If $i = 0$, we define new decremental MSF instance D'_0 with underlying graph $G'_0 = (V, F \cup R \cup R')$. For some $j \in \{1, 2\}$ where $D_{0,j} = \emptyset$ we assign $D_{0,j} \leftarrow D'_0$. (Again, assignment takes constant time.)

Update: Hard things

Clean up. Let R be a set of edges subject to clean-up. Let R' be a set of edges (to be defined). For each level $0 \leq i \leq L+1$ we will do *clean-up of level* i. For each level i:

If $i = 0$, we define new decremental MSF instance D'_0 with underlying graph $G'_0 = (V, F \cup R \cup R')$. For some $j \in \{1, 2\}$ where $D_{0,j} = \emptyset$ we assign $D_{0,j} \leftarrow D'_0$. (Again, assignment takes constant time.)

If $i > 0$, we will discuss over the time interval $[k \times 2^i, (k+1) \times 2^i]$: Everything will go periodic.

In the beginning of the interval, we initialize D'_i in level i (contents to be defined), which ends in the end of the interval. As D'_i is initialized (in some time τ divisible by 2^i). it will be assigned to some $j \in \{1, 2\}$ with $D_{i,j} = \emptyset$. If $i = L+1$, $D_{L,0} \leftarrow D'_i$.

Then for each $1 \leq i \leq L+1$, we will have $(D_{i-1,3}, D_{i-1,4}) \leftarrow (D_{i-1,1}, D_{i-1,2})$, $(D_{i-1,1}, D_{i-1,2}) \leftarrow (\emptyset, \emptyset)$.

Update: Hard things

That's all for the clean-up: Except that we don't know what is D'_i!

Let $G'_i = (V, F \cup N'_i), N'_i = N_{i-1,3} \cup N_{i-1,4}$
($N'_{L+1} = N_{L,0} \cup N_{L,3} \cup N_{L,4}$). We will initialize D'_i from G'_i.

Update: Hard things

That's all for the clean-up: Except that we don't know what is D'_i!

Let $G'_i = (V, F \cup N'_i), N'_i = N_{i-1,3} \cup N_{i-1,4}$
($N'_{L+1} = N_{L,0} \cup N_{L,3} \cup N_{L,4}$). We will initialize D'_i from G'_i.

Divide the interval to two blocks
$I_1 = [\tau, \tau + 2^{i-1}), I_2 = [\tau + 2^{i-1}, \tau + 2^i)$. In I_1 we initialize G'_i in *constant speed*: If we have X operations, each time we perform $X/2^{i-1}$ operations. Here, D'_i now reflects all information applied until τ.

We have 2^i updates to do in 2^{i-1} time. *It's fast forward time!* We do two updates in one time. Formally, we do updates on time $\tau + 2k, \tau + 2k + 1$ in time $\tau + 2^{i-1} + k$. Edge addition updates are ignored here.

Update: Hard things

That's all for the clean-up: Except that we don't know what is D'_i!

Let $G'_i = (V, F \cup N'_i), N'_i = N_{i-1,3} \cup N_{i-1,4}$
($N'_{L+1} = N_{L,0} \cup N_{L,3} \cup N_{L,4}$). We will initialize D'_i from G'_i.

Divide the interval to two blocks
$I_1 = [\tau, \tau + 2^{i-1}), I_2 = [\tau + 2^{i-1}, \tau + 2^i)$. In I_1 we initialize G'_i in *constant speed*: If we have X operations, each time we perform $X/2^{i-1}$ operations. Here, D'_i now reflects all information applied until τ.

We have 2^i updates to do in 2^{i-1} time. *It's fast forward time!* We do two updates in one time. Formally, we do updates on time $\tau + 2k, \tau + 2k + 1$ in time $\tau + 2^{i-1} + k$. Edge addition updates are ignored here.

Finally, for each time τ, let R' be the set of reconnecting edges returned by *fast-forward* updates in I_2, among all levels. R' is emptied every time by clean-up procedure on level 0.

Proof of Correctness

Proposition 14.1. For all i, j, $G_{i,j}$ is a subgraph of G every time.

Proposition 14.2. For all $0 \leq i \leq L$, when you set $D_{i,j} \leftarrow D'_i$, there exists some $j \in \{1, 2\}$ where $D_{i,j} = \emptyset$.

Lemma 14.3. Let $N \subseteq \bigcup_{i,j} N_{i,j}$. If $e \in F$ is removed, let f' be a lightest reconnecting edges connecting F from G. $f' \in R_0$.

Proof of Correctness

Proposition 14.1. For all i, j, $G_{i,j}$ is a subgraph of G every time.

Proposition 14.2. For all $0 \leq i \leq L$, when you set $D_{i,j} \leftarrow D'_i$, there exists some $j \in \{1, 2\}$ where $D_{i,j} = \emptyset$.

Lemma 14.3. Let $N \subseteq \bigcup_{i,j} N_{i,j}$. If $e \in F$ is removed, let f' be a lightest reconnecting edges connecting F from G. $f' \in R_0$.

Lemma 14.4. The invariant $N = \bigcup_{i,j} N_{i,j}$ $F = MSF(G)$ holds through each updates.

Proof. Statement is obvious after preprocessing. We prove, that each update does not violate the invariant.

Proof of Correctness (Lemma 14.4)

Part 1 (abridged). We first prove $N \subseteq \bigcup_{i,j} N_{i,j}$. Observe that R contains a candidate of new possible non-tree edges, in insertion and deletion. We plug them in $N_{0,j} = R \cup R'$, so they do not escape $\bigcup_{i,j} N_{i,j}$.

It only remains to prove the statement for clean-up step. We initialize $N'_i = N_{i-1,3} \cup N_{i-1,4}$ in time $\tau - 2^i$, and destroy $N_{i-1,3}, N_{i-1,4}$ and insert $N_{i,j} \leftarrow N'_i$ in time τ. So we could only miss a thing in interval $[\tau - 2^i, \tau)$: R' is there to catch them.

Proof of Correctness (Lemma 14.4)

Part 1 (abridged). We first prove $N \subseteq \bigcup_{i,j} N_{i,j}$. Observe that R contains a candidate of new possible non-tree edges, in insertion and deletion. We plug them in $N_{0,j} = R \cup R'$, so they do not escape $\bigcup_{i,j} N_{i,j}$.

It only remains to prove the statement for clean-up step. We initialize $N'_i = N_{i-1,3} \cup N_{i-1,4}$ in time $\tau - 2^i$, and destroy $N_{i-1,3}, N_{i-1,4}$ and insert $N_{i,j} \leftarrow N'_i$ in time τ. So we could only miss a thing in interval $[\tau - 2^i, \tau)$: R' is there to catch them.

Part 2 (skipped). Then we prove $\bigcup_{i,j} N_{i,j} \subseteq N$. This is slightly harder, so we skip it.

Part 3 (easy). As $N = \bigcup_{i,j} N_{i,j}$, we can see $f^* \in R_0$. With this you can easily see $F = MSF(G)$ holds.

Running Time

Lemma 16.1. For any i, j $|N_{i,j}| \leq min(2^{i+1}B, k)$.

Proof. $N_{i,j} \subseteq N$ so trivially $|N_{i,j}| \leq k$. If you write stuffs down you can see $|R \cup R'| \leq max(B, 4L + 5) + 2L + 2 \leq 2B$. By observing the recurrence you can see that each level doubles the size, thus $|N_{0,j}| \leq 2B$ implies $|N_{i,j}| \leq 2^{i+1}B$.

Lemma 16.2. Preprocessing takes $t_{pre}(m, k, p') + O(m \log m)$.

Proof. Top tree takes $O(m \log m)$.

Lemma 16.3. Clean up takes
$O(\sum_{i=0}^{\lceil \log k \rceil} t_{pre}(m, min(2^{i+1}B, k), p')/2^i + t_u(m, k, p') \log k)$.

Lemma 16.4. Update takes
$O(\sum_{i=0}^{\lceil \log k \rceil} t_{pre}(m, min(2^{i+1}B, k), p')/2^i + t_u(m, k, p') \log k + B \log m)$

This finishes the proof of the first reduction.

The second reduction

Tree Compression

Tired?

Tree Compression

Tired?

I am.... But the second reduction is more accessible to competitive programmer, probably.

We use the **Tree compression** technique introduced in JOI 2014 Factories. Let's define it in a boring way.

Tree Compression

Tired?

I am.... But the second reduction is more accessible to competitive programmer, probably.

We use the **Tree compression** technique introduced in JOI 2014 Factories. Let's define it in a boring way.

Definition (Connecting Paths). For a tree $T = (V, E)$ and a set of terminal $S \subseteq V$, $P_S(T)$ is defined as follows.

1. $P_S(T)$ is a set of mutually edge disjoint paths.

2. $\cup_{P \in P_S(T)} P$ forms a connected subtree of T.

3. For all terminal $u \in S$, u is an endpoint of some path $P \in P_S(T)$.

4. Any endpoint of some path $p \in P_S(T)$, either satisfies $u \in S$ or is a endpoint of other two path p', p''.

Tree Compression

Now let's take a look at some typical corollary.

Lemma 18.1 For fixed $T, S \subseteq V$, $P_S(T)$ is unique.

Proposition 18.2. Let $end(E)$ be a set of vertices that is either endpoint of some edges in E. Let $G = (V, E), F \subseteq E$, vertex subset $end(E - F) \subseteq S$. Any path forming $P_S(F)$ is an induced path over G.

Tree Compression

Definition 18.3 (Contracted Graphs/Forests and super edges). For a weighted graph $G = (V, E, w)$, forest $F \subseteq E$, vertex subset $end(E - F) \subseteq S$, a contracted (multi)graph G' and contracted forest F', with respect to S, can be constructed from G, F as follows.

1. Remove all edges from $F \setminus \cup_{P \in P_S(F)} P$. (aka remove leafs repeatedly)

2. Replace all paths with one super edge, with weights set as a path maximum.

Tree Compression

Definition 18.3 (Contracted Graphs/Forests and super edges). For a weighted graph $G = (V, E, w)$, forest $F \subseteq E$, vertex subset $end(E - F) \subseteq S$, a contracted (multi)graph G' and contracted forest F', with respect to S, can be constructed from G, F as follows.

1. Remove all edges from $F \setminus \cup_{P \in P_S(F)} P$. (aka remove leafs repeatedly)

2. Replace all paths with one super edge, with weights set as a path maximum.

In layman's term, F' is a tree compression of S w.r.t F, and we add non-tree edges to obtain G.

Let $(G', F') = Contract_S(G, F)$. For some $e \in P_{uv} \subseteq F$ Let $(u, v) \in F'$ be a *super edge covering* e.

Tree Compression

Definition 18.3 (Contracted Graphs/Forests and super edges). For a weighted graph $G = (V, E, w)$, forest $F \subseteq E$, vertex subset $end(E - F) \subseteq S$, a contracted (multi)graph G' and contracted forest F', with respect to S, can be constructed from G, F as follows.

1. Remove all edges from $F \setminus \cup_{P \in P_S(F)} P$. (aka remove leafs repeatedly)

2. Replace all paths with one super edge, with weights set as a path maximum.

In layman's term, F' is a tree compression of S w.r.t F, and we add non-tree edges to obtain G.

Let $(G', F') = Contract_S(G, F)$. For some $e \in P_{uv} \subseteq F$ Let $(u, v) \in F'$ be a *super edge covering* e.

Proposition 18.4. $E(G') = N \cup E(F')$ (easy).

Proposition 18.5. $F = MSF(G) \implies F' = MSF(G')$ (easy).

Tree Compression

Lemma 20.1. There is an algorithm C that works in two phases:

In phase 1, C maintains forest with at most m edges. Edges can be inserted/deleted in $O(\log m)$ worst case time.

In phase 2, given any edge subest N, C can return $(G', F') = Contract_{end(E-F)}(G, F)$ in $O(|N| \log m)$ time. Here $G = (V, F \cup N)$. For any edge $e \in F$, in $O(\log m)$, C can return a super edge $e' = (u', v') \in F'$ that covers e (or not, if it doesn't exist).

Proof. Paper mentions top tree, but I think ET-tree does a job too.

Tree Compression

Now with this data structure, and some casework (non-tree edge, tree-but-not-compressed edges, tree edge that is bridge, tree edge), you can reduce the problem into Decremental MSF with at most $5k$ vertices (small vertices).

Tree Compression

Now with this data structure, and some casework (non-tree edge, tree-but-not-compressed edges, tree edge that is bridge, tree edge), you can reduce the problem into Decremental MSF with at most $5k$ vertices (small vertices).

Sort of.

The non-trivial issue arises in initialization of C. We does a sort of *toggling* with the technique of redistributing operation clocks. We will skip this.

Tree Compression

Now with this data structure, and some casework (non-tree edge, tree-but-not-compressed edges, tree edge that is bridge, tree edge), you can reduce the problem into Decremental MSF with at most $5k$ vertices (small vertices).

Sort of.

The non-trivial issue arises in initialization of C. We does a sort of *toggling* with the technique of redistributing operation clocks. We will skip this.

Now, the only goal is to satisfy the degree 3 condition.

Here, you can simply construct a binary tree over an adjacency list, with infinite weights.

The reduction is finished!

Behind-the-scenes

After the reduction, we need the algorithm that solves Decremental MSF in graphs with maximum degree 3.

To do this, we construct a *MSF decomposition*, which is a hierarchical decomposition on the degree 3 graph.

Behind-the-scenes

After the reduction, we need the algorithm that solves Decremental MSF in graphs with maximum degree 3.

To do this, we construct a *MSF decomposition*, which is a hierarchical decomposition on the degree 3 graph.

Each cluster on this decomposition have high *conductance*.

A *conductance* of the cut is defined as $\phi(S) = \frac{\delta(S)}{min(vol(S),vol(V-S))}$.

A *conductance* of the graph is defined as a minimum conductance over all proper cuts.

If a conductance is 0, G is disconnected. Conductance is a rough indicator of how *well-connected* a graph is.

Behind-the-scenes

After the reduction, we need the algorithm that solves Decremental MSF in graphs with maximum degree 3.

To do this, we construct a *MSF decomposition*, which is a hierarchical decomposition on the degree 3 graph.

Behind-the-scenes

After the reduction, we need the algorithm that solves Decremental MSF in graphs with maximum degree 3.

To do this, we construct a *MSF decomposition*, which is a hierarchical decomposition on the degree 3 graph.

This have a tree-like structure, and MSF can be updated in recursive, tree-dp fashion... Let's say.

Each cluster on this decomposition have high *conductance*.

Behind-the-scenes

After the reduction, we need the algorithm that solves Decremental MSF in graphs with maximum degree 3.

To do this, we construct a *MSF decomposition*, which is a hierarchical decomposition on the degree 3 graph.

This have a tree-like structure, and MSF can be updated in recursive, tree-dp fashion... Let's say.

Each cluster on this decomposition have high *conductance*.

A *conductance* of the cut is defined as $\phi(S) = \frac{\delta(S)}{min(vol(S),vol(V-S))}$.

A *conductance* of the graph is defined as a minimum conductance over all proper cuts.

If a conductance is 0, G is disconnected. Conductance is a rough indicator of how *well-connected* a graph is.

Behind-the-scenes

Theorem (Dynamic Expander Pruning). For some function $\epsilon(n) = o(1)$, let $\alpha_0(n) = 1/n^{\epsilon(n)}$. In a graph with edge deletion of length $T = O(m\alpha_0^2(n))$, there is a decremental algorithm A that maintains the following vertex subset $P \subseteq V$.

Let G_τ, P_τ be the graph G and set P after τ-th deletion.

1. First, A sets $P_0 = \emptyset$, and receives $G_0 = (V, E)$ as an input (in constant time: It receives an adjacency list pointer).

2. After τ-th deletion, in $n^{O(\log\log \frac{1}{\epsilon(n)} / \log \frac{1}{\epsilon(n)})} = n^{o(1)}$ time, A returns a set S to add to $P_{\tau-1}$, then $P_\tau = P_{\tau-1} \cup S$.

3. For each step τ, there exists $W_\tau \subseteq P_\tau$ such that $G_\tau[V - W_\tau]$ is connected (aka, $V - P_\tau$ is within a single connected component of G_τ), if $\phi(G_0) \geq \alpha_0(n)$.

Behind-the-scenes

Q1. Isn't it trivial? It is trivial in a sense that we can call $P_1 = V$ and call it a day, but note that we should return the set S: That move will break the worst case time bound for first query which does it.

Behind-the-scenes

Q1. Isn't it trivial? It is trivial in a sense that we can call $P_1 = V$ and call it a day, but note that we should return the set S: That move will break the worst case time bound for first query which does it.

Q2. Isn't it impossible? Consider two huge connected component connected by bridges: Then we always need $|P_i - P_{i-1}| = |V|/2$. This is why we have a *conductance guarantee* of $\phi(G_0) \geq \alpha_0(n)$.

Behind-the-scenes

Q1. Isn't it trivial? It is trivial in a sense that we can call $P_1 = V$ and call it a day, but note that we should return the set S: That move will break the worst case time bound for first query which does it.

Q2. Isn't it impossible? Consider two huge connected component connected by bridges: Then we always need $|P_i - P_{i-1}| = |V|/2$. This is why we have a *conductance guarantee* of $\phi(G_0) \geq \alpha_0(n)$.

Q3. WTF, I don't care, why do you? Basically this expander pruning enables the *divide-and-conquer*, aka hierarchial decomposition over the graph G, with good guarantees. We use the result of expander pruning P, as a *guide* the division scheme.

Behind-the-scenes

Q1. Isn't it trivial? It is trivial in a sense that we can call $P_1 = V$ and call it a day, but note that we should return the set S: That move will break the worst case time bound for first query which does it.

Q2. Isn't it impossible? Consider two huge connected component connected by bridges: Then we always need $|P_i - P_{i-1}| = |V|/2$. This is why we have a *conductance guarantee* of $\phi(G_0) \geq \alpha_0(n)$.

Q3. WTF, I don't care, why do you? Basically this expander pruning enables the *divide-and-conquer*, aka hierarchial decomposition over the graph G, with good guarantees. We use the result of expander pruning P, as a *guide* the division scheme.

Q4. So how to solve? Long story, but basically we can relax this as a cut instance, which is then considered as a flow problem, that is solved with push-relabel flows (and only with that).

Summing up

In the lecture, we tried to learn how to reduce Dynamic MSF to Decremental MSF.

The reduction is composed with two steps.

We tried to get a grasp on the actual solving of decremental MSF, and reviewed it's toolkits.

Summing up

In the lecture, we tried to learn how to reduce Dynamic MSF to Decremental MSF.

The reduction is composed with two steps.

We tried to get a grasp on the actual solving of decremental MSF, and reviewed it's toolkits.

Probably this wasn't a good introduction to the Dynamic MSF literature (we should've read HDLT instead), but hope you still enjoyed it.

Next project-tcs iteration will be announced in late January.

Thank you for the attention!