Dynamic Maintenance of Low-Stretch Probabilistic Tree Embeddings with Applications

Sebastian Forster[1], Gramoz Goranci[2], Monika Henzinger[3]

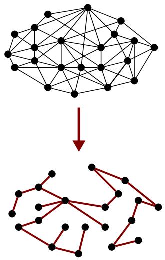
[1]University of Salzburg
[2]University of Toronto → University of Glasgow
[3]University of Vienna

DIMAP Seminar, University of Warwick
May 2021

Tree-Based Graph Approximations

Powerful Theme in Graph Algorithms

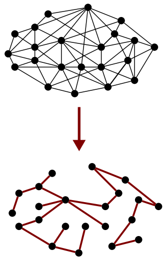
- Approximate arbitrary graphs by trees
- Why? Many graph problems are easy on trees
- Map the tree solution back to the original graph



Tree-Based Graph Approximations

Powerful Theme in Graph Algorithms

- Approximate arbitrary graphs by trees
- Why? Many graph problems are easy on trees
- Map the tree solution back to the original graph

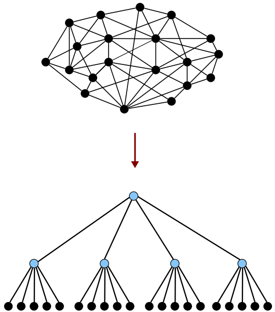


example	property preserved
Spanning Tree/Forest	Connectivity
BFS Tree/Shortest-Path Tree	Distance from a source
Gomory-Hu Tree	Pairwise s-t max flow/min-cut
Tree Cut/Flow Sparsifier	Cut/Flow
Low-Stretch Spanning Trees	Average Pairwise Distance
Prob. Low-Stretch Trees	**(Exp.) Pairwise Distance**

Probabilistic Tree Embedding (PTE) [Bartal'96]

Definition

- For any simple graph $G = (V, E)$, $n = |V|$, $m = |E|$, a probability distribution τ over trees $\{T_i\}_i$ is an $\alpha-$**probabilistic tree embedding** ($\alpha-$PTE) iff for all $u, v \in V$

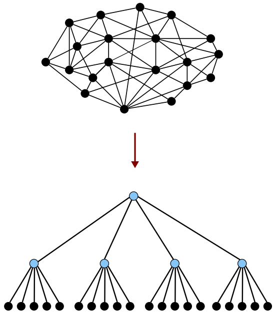


Probabilistic Tree Embedding (PTE) [Bartal'96]

Definition

- For any simple graph $G = (V, E)$, $n = |V|$, $m = |E|$, a probability distribution τ over trees $\{T_i\}_i$ is an $\alpha-$**probabilistic tree embedding** ($\alpha-$PTE) iff for all $u, v \in V$

  (1) $V(G) \subseteq V(T_i)$ for all i



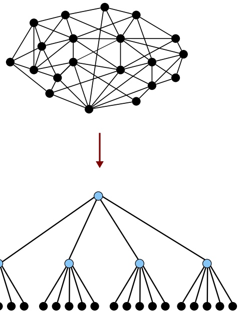
Probabilistic Tree Embedding (PTE) [Bartal'96]

Definition

- For any simple graph $G = (V, E)$, $n = |V|$, $m = |E|$, a probability distribution τ over trees $\{T_i\}_i$ is an $\alpha-$**probabilistic tree embedding** ($\alpha-$PTE) iff for all $u, v \in V$

  (1) $V(G) \subseteq V(T_i)$ for all i

  (2) $\mathbf{dist}_{T_i}(u, v) \geq \mathbf{dist}_G(u, v)$ for all i



Probabilistic Tree Embedding (PTE) [Bartal'96]

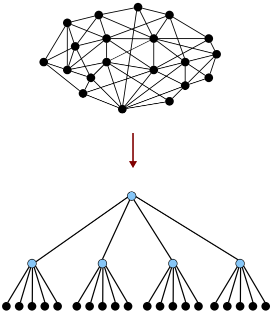
Definition

- For any simple graph $G = (V, E)$, $n = |V|$, $m = |E|$, a probability distribution τ over trees $\{T_i\}_i$ is an $\alpha-$**probabilistic tree embedding** ($\alpha-$PTE) iff for all $u, v \in V$

  (1) $V(G) \subseteq V(T_i)$ for all i

  (2) $\mathbf{dist}_{T_i}(u, v) \geq \mathbf{dist}_G(u, v)$ for all i

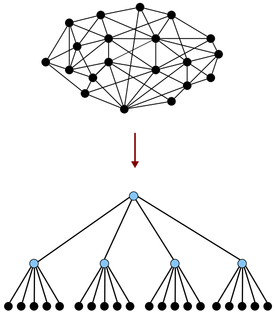
  (3) $\mathbb{E}_{T \sim \tau}[\mathbf{dist}_T(u, v)] \leq \alpha \cdot \mathbf{dist}_G(u, v)$



Probabilistic Tree Embedding (PTE) [Bartal'96]

Definition

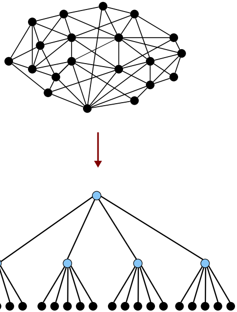
- For any simple graph $G = (V, E)$, $n = |V|$, $m = |E|$, a probability distribution τ over trees $\{T_i\}_i$ is an $\alpha-$**probabilistic tree embedding** ($\alpha-$PTE) iff for all $u, v \in V$
  - (1) $V(G) \subseteq V(T_i)$ for all i
  - (2) $\mathbf{dist}_{T_i}(u, v) \geq \mathbf{dist}_G(u, v)$ for all i
  - (3) $\mathbb{E}_{T\sim\tau}[\mathbf{dist}_T(u, v)] \leq \alpha \cdot \mathbf{dist}_G(u, v)$
- **Goal:** Find an $\alpha-$PTE with small α (**stretch**)



Probabilistic Tree Embedding (PTE) [Bartal'96]

Definition

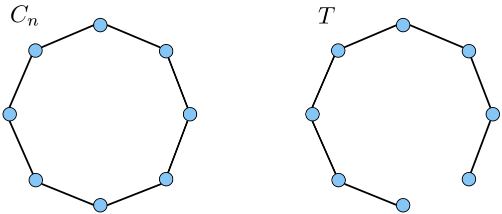
- For any simple graph $G = (V, E)$, $n = |V|$, $m = |E|$, a probability distribution τ over trees $\{T_i\}_i$ is an $\alpha-$**probabilistic tree embedding** ($\alpha-$PTE) iff for all $u, v \in V$
  - (1) $V(G) \subseteq V(T_i)$ for all i
  - (2) $\mathbf{dist}_{T_i}(u, v) \geq \mathbf{dist}_G(u, v)$ for all i
  - (3) $\mathbb{E}_{T\sim\tau}[\mathbf{dist}_T(u, v)] \leq \alpha \cdot \mathbf{dist}_G(u, v)$
- **Goal:** Find an $\alpha-$PTE with small α (**stretch**)



Applications

- **buy-at-bulk network design**, group steiner tree, metric labelling, oblivious routing, min-sum clustering, distributed k-server, mirror placement, linear arrangement, **approx. all-pairs shortest path**

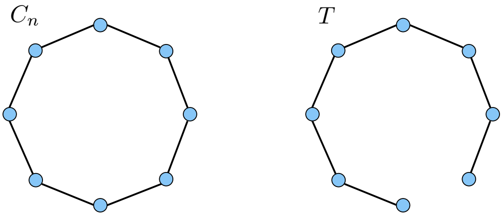
Tree Embedding of Cycles



Bad News [Rabinovich Raz'95]

- For any tree that **deterministically** approximates the n-cycle, it holds that $\alpha = \Omega(n)$

Tree Embedding of Cycles



Bad News [Rabinovich Raz'95]

- For any tree that **deterministically** approximates the n-cycle, it holds that $\alpha = \Omega(n)$

Good News [Karp'89]

- The n-cycle C_n admits a 2-PTE – **ALG**: delete an edge at random!
- For each edge (u, v) in the cycle C_n

$$\mathbb{E}(\mathrm{dist}_T(u,v)) = \frac{1}{n} \cdot (n-1) + \frac{n-1}{n} \cdot 1 \leq 2 \cdot \mathrm{dist}_{C_n}(u,v)$$

Probabilistic Tree Embedding (PTE)

expected stretch α	runtime	reference
$\mathcal{O}(\log^2 n)$	polynomial	[Bartal'96]
$\mathcal{O}(\log n \log\log n)$	polynomial	[Bartal'98]
$\mathcal{O}(\log n)$	polynomial	[Fakcharoenphol et al.'03]
$\mathcal{O}(\log n)$	$\mathcal{O}(m \log^3 n)$	[Mendel Schwob'09]
$\mathcal{O}(\log n)$	$\mathcal{O}(m \log n)$	[Blelloch Guh Sun'17]

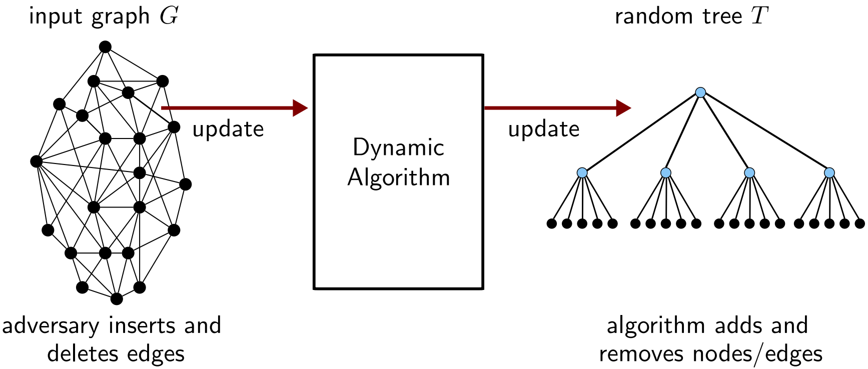
Probabilistic Tree Embedding (PTE)

expected stretch α	runtime	reference
$\mathcal{O}(\log^2 n)$	polynomial	[Bartal'96]
$\mathcal{O}(\log n \log\log n)$	polynomial	[Bartal'98]
$\mathcal{O}(\log n)$	polynomial	[Fakcharoenphol et al.'03]
$\mathcal{O}(\log n)$	$\mathcal{O}(m \log^3 n)$	[Mendel Schwob'09]
$\mathcal{O}(\log n)$	$\mathcal{O}(m \log n)$	[Blelloch Guh Sun'17]

Lower Bound [Bartal'96]

- For any n, there exists a graph G_n such that for any α-PTE of G_n it holds that $\alpha = \Omega(\log n)$.

Fully-Dynamic Probabilistic Tree Embedding



Goal:

- minimize **update** time (time to handle edge insertions/deletions)
- ensure that T has small (expected) **stretch** after each update

Dynamic PTE – Our Result

- The first dynamic algorithm for maintaining **probabilistic** low-stretch tree embedding in **sub-linear** time per update
  - (1) our bounds are **amortized**, assume **oblivious adversary**
  - (2) can handle graphs with polynomially bounded weights

Stretch	Update time	Stretch type	Tree type	Reference
$\mathcal{O}(\log^4 n)$ $n^{o(1)}$ $\mathcal{O}(\log n)^{3i-2}$	$m^{1/2+o(1)}$ $n^{o(1)}$ $m^{1/i+o(1)}$ ([1])	expected	low-depth	**[Our result]**

[1] $i \leq \sqrt{\log n}$

Dynamic PTE – Our Result

- The first dynamic algorithm for maintaining **probabilistic** low-stretch tree embedding in **sub-linear** time per update
  (1) our bounds are **amortized**, assume **oblivious adversary**
  (2) can handle graphs with polynomially bounded weights

Stretch	Update time	Stretch type	Tree type	Reference
$\mathcal{O}(\log^4 n)$ $n^{o(1)}$ $\mathcal{O}(\log n)^{3i-2}$	$m^{1/2+o(1)}$ $n^{o(1)}$ $m^{1/i+o(1)}$ (1)	expected	low-depth	**[Our result]**
$n^{o(1)}$	$n^{o(1)}$	**average**	spanning	[Chechik Zhang'20]

[1] $i \leq \sqrt{\log n}$

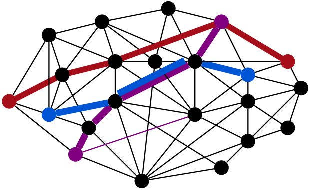
Dynamic PTE – Our Result

- The first dynamic algorithm for maintaining **probabilistic** low-stretch tree embedding in **sub-linear** time per update
  - (1) our bounds are **amortized**, assume **oblivious adversary**
  - (2) can handle graphs with polynomially bounded weights

Stretch	Update time	Stretch type	Tree type	Reference
$\mathcal{O}(\log^4 n)$ $n^{o(1)}$ $\mathcal{O}(\log n)^{3i-2}$	$m^{1/2+o(1)}$ $n^{o(1)}$ $m^{1/i+o(1)}$ ([1])	expected	low-depth	**[Our result]**
$n^{o(1)}$	$n^{o(1)}$	**average**	spanning	[Chechik Zhang'20]
$n^{o(1)}$	$n^{1/2+o(1)}$			[Forster Goranci'19]

[1] $i \leq \sqrt{\log n}$

Buy-At-Bulk Network Design



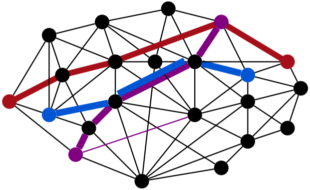
Input

- Graph $G = (V, E)$, positive lengths ℓ_e
- k source-sink s_i, t_i with demand $\mathbf{dem}(i)$
- non-decreasing, sub-additive function f

Buy-At-Bulk Network Design

Input

- Graph $G = (V, E)$, positive lengths ℓ_e
- k source-sink s_i, t_i with demand $\mathbf{dem}(i)$
- non-decreasing, sub-additive function f



Routing & Edge cost

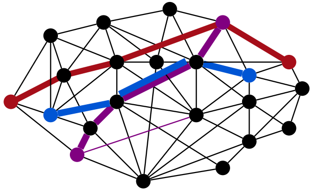
- **routing** of demands is a collection of paths $\{P_i\}_i$ that sends $\mathrm{dem}(s_i, t_i)$ units of commodity from s_i to t_i
- **cost:** $c(e)$ amount of commodity set along the edge, i.e.,

$$c_e := \sum_{i:e \in P_i} \mathrm{dem}(i)$$

Buy-At-Bulk Network Design

Input

- Graph $G = (V, E)$, positive lengths ℓ_e
- k source-sink s_i, t_i with demand $\mathbf{dem}(i)$
- non-decreasing, sub-additive function f



Routing & Edge cost

- **routing** of demands is a collection of paths $\{P_i\}_i$ that sends $\mathrm{dem}(s_i, t_i)$ units of commodity from s_i to t_i
- **cost:** $c(e)$ amount of commodity set along the edge, i.e.,

$$c_e := \sum_{i:e \in P_i} \mathrm{dem}(i)$$

Goal:

- find a routing $\{P_i\}_i$ that minimizes **total cost** $\sum_{e \in E} \ell_e f(c_e)$

Applications of Dynamic PTE

Fully Dynamic All-Pairs Shortest Path

Approx	Update time	Query time	Reference
$\mathcal{O}(\log n)^{3i-2}$	$m^{1/i+o(1)}$	$\mathcal{O}(\log n)^{5/2}$	**[Our result]**
$2^{\mathcal{O}(k\rho)}$ ([1])	$\tilde{\mathcal{O}}(\sqrt{m}n^{1/k})$	$\mathcal{O}(k^2\rho^2)$	[Abraham et al.'14]

- goes below the $\mathcal{O}(\sqrt{m})$ bound on update time with non-trivial approx.

[1] $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$

Applications of Dynamic PTE

Fully Dynamic All-Pairs Shortest Path

Approx	Update time	Query time	Reference
$\mathcal{O}(\log n)^{3i-2}$	$m^{1/i+o(1)}$	$\mathcal{O}(\log n)^{5/2}$	**[Our result]**
$2^{\mathcal{O}(k\rho)}$ ([1])	$\tilde{\mathcal{O}}(\sqrt{m}n^{1/k})$	$\mathcal{O}(k^2\rho^2)$	[Abraham et al.'14]

- goes below the $\mathcal{O}(\sqrt{m})$ bound on update time with non-trivial approx.

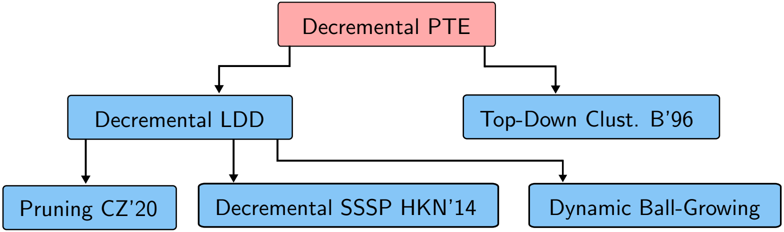
Fully Dynamic Buy-At-Bulk Network Design

Approx	Update time	Query time	Reference
$\mathcal{O}(\log n)^{3i-2}$	$m^{1/i+o(1)}$	$\mathcal{O}(k\log^{3/2} n)$	**[Our result]**

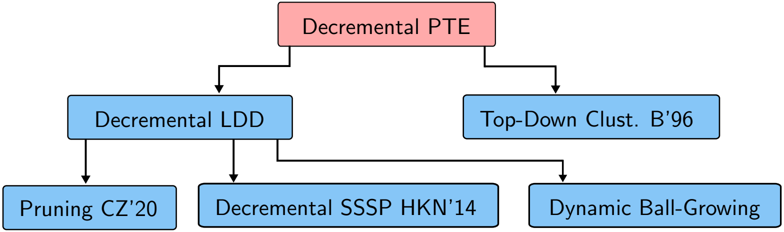
- this constitutes the first dynamic algorithm for the problem

[1] $\rho = 1 + \lceil {}^{\log n^{1-1/k}}/_{\log(m/n^{1-1/k})} \rceil$

Overview


Decremental PTE
Decremental LDD
Top-Down Clust. B'96
Pruning CZ'20
Decremental SSSP HKN'14
Dynamic Ball-Growing

Overview



Extension to Fully Dynamic Setting

- Introduce a new "bootstrapping" idea
- Recursively employ fully dynamic algorithms in the reduction

Probabilistic Low-Diameter Decompositions (LDD)

Idea:

- cluster graphs into **small** diameter clusters w/ **few** inter-cluster edges

Probabilistic Low-Diameter Decompositions (LDD)

Idea:

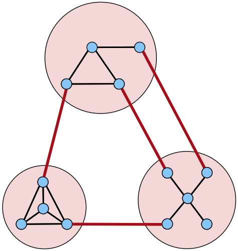
- cluster graphs into **small** diameter clusters w/ **few** inter-cluster edges

$(\beta, \gamma)-$**(probabilistic) LDD** [Linial Saks'93, Bartal'96]

- A randomized partitioning of $G = (V, E)$ into vertex-disjoint clusters $C_1 \dots C_k$ such that

  (1) **weak diameter** of each C_i is at most β

  (2) $\mathbb{P}(u \in C_i, v \in C_{j \neq i}) \leq \gamma$ for each edge (u, v)



Probabilistic Low-Diameter Decompositions (LDD)

Idea:

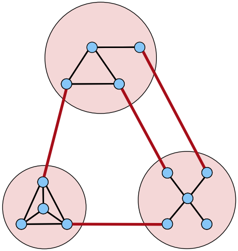
- cluster graphs into **small** diameter clusters w/ **few** inter-cluster edges

$(\beta, \gamma)-$**(probabilistic) LDD** [Linial Saks'93, Bartal'96]

- A randomized partitioning of $G = (V, E)$ into vertex-disjoint clusters $C_1 \dots C_k$ such that

  (1) **weak diameter** of each C_i is at most β

  (2) $\mathbb{P}(u \in C_i, v \in C_{j \neq i}) \leq \gamma$ for each edge (u, v)



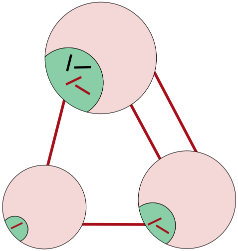
Applications

- key tool for constructing tree-based graph approximation for distances, i.e.g, low-stretch spanning trees, probabilistic tree embeddings
- approximation algorithms, e.g., min-max graph partitioning

Probabilistic LDD under Edge Deletions

Goal

- maintain (β, γ)-probabilistic LDD $\{C_i\}_{i=1}^{k}$ of graph G under **edge deletions**; k may change



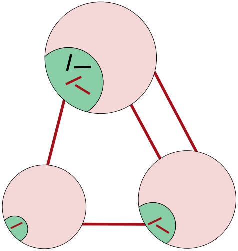
Probabilistic LDD under Edge Deletions

Goal

- maintain (β, γ)-probabilistic LDD $\{C_i\}_{i=1}^{k}$ of graph G under **edge deletions**; k may change

Theorem [Forster **G** Henzinger'21]

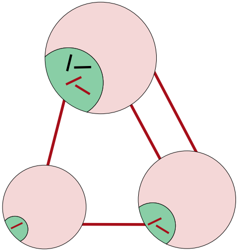
- For $\beta \in (0, 1)$, there is a data-structure for maintaining a $(\beta, \mathcal{O}(\beta^{-1} \log^2 n))-$**probabilistic LDD** of G in $m^{1+o(1)}$ total update time.



Probabilistic LDD under Edge Deletions

Goal

- maintain (β, γ)-probabilistic LDD $\{C_i\}_{i=1}^k$ of graph G under **edge deletions**; k may change



Theorem [Forster **G** Henzinger'21]

- For $\beta \in (0, 1)$, there is a data-structure for maintaining a $(\beta, \mathcal{O}(\beta^{-1} \log^2 n))$**−probabilistic LDD** of G in $m^{1+o(1)}$ total update time.

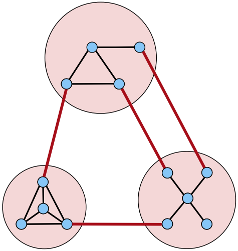
Important Feature

- our runtime is **independent** of β^{-1}
- **key** requirement for **top-down** graph clustering
- all previous dynamic graph clusterings [Saranurak Wang'19], [Chechik Zhang'20], [Forster Goranci'19] have runtimes depending on β^{-1}

Ball-Growing for Static Probabilistic LDD

Algorithm — Ball-Growing [Bartal'96]

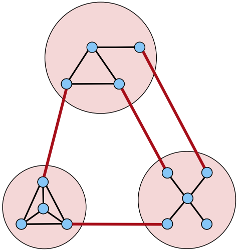
- Set $i \leftarrow 1$, every vertex is unmarked initially
- While there are unmarked vertices:
  - Pick an unmarked vertex v
  - Sample $R_v \sim \mathbf{Geom}(p)$ with $p = \mathcal{O}(\beta^{-1} \log n)$
  - Add all unmarked vertices in $\mathbf{Ball}_G(v, R_v)$ to C_i
  - Set $i \leftarrow i + 1$



Ball-Growing for Static Probabilistic LDD

Algorithm — Ball-Growing [Bartal'96]

- Set $i \leftarrow 1$, every vertex is unmarked initially
- While there are unmarked vertices:
  - Pick an unmarked vertex v
  - Sample $R_v \sim \mathbf{Geom}(p)$ with $p = \mathcal{O}(\beta^{-1} \log n)$
  - Add all unmarked vertices in $\mathbf{Ball}_G(v, R_v)$ to C_i
  - Set $i \leftarrow i + 1$



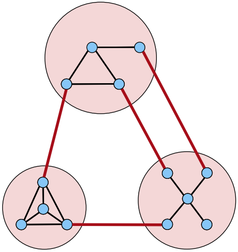
Claim

- Ball-Growing constructs a $(\beta, \mathcal{O}(\beta^{-1} \log n))-$**LDD** in $\tilde{\mathcal{O}}(m)$ time

Ball-Growing for Static Probabilistic LDD

Algorithm — Ball-Growing [Bartal'96]

- Set $i \leftarrow 1$, every vertex is unmarked initially
- While there are unmarked vertices:
  - Pick an unmarked vertex v
  - Sample $R_v \sim \mathbf{Geom}(p)$ with $p = \mathcal{O}(\beta^{-1} \log n)$
  - Add all unmarked vertices in $\mathbf{Ball}_G(v, R_v)$ to C_i
  - Set $i \leftarrow i + 1$



Claim

- Ball-Growing constructs a $(\beta, \mathcal{O}(\beta^{-1} \log n))-$**LDD** in $\tilde{\mathcal{O}}(m)$ time

How to make it Dynamic?

- white box and extend cluster **pruning** of [Chechik Zhang'20]

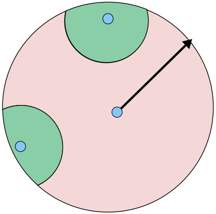
Handling Deletions – Cluster Pruning

$\textsc{Delete}(e)$

- $G \leftarrow G \setminus \{e\}$ and $\textsc{Prune}(C)$ for all C with $e \in C$

$\textsc{Prune}(C)$

- If $|C| > 1$ and $\exists v \in C$ s.t. $\mathbf{dist}^*_C(c, v) > p^{-1} \log n$:
  - Sample $R \sim \mathrm{Geom}(p)$, set $B \leftarrow \mathrm{Ball}_C(v, R)$
  - If $\mathbf{vol}(B) \leq 1/2 \cdot \mathbf{vol}^*(C)$:
    - $C \leftarrow C \setminus B$, Form new cluster B
    - $\textsc{AssignCenter}(B)$, $\textsc{Prune}(B)$
  - Else: $\textsc{AssignCenter}(C)$
  - $\textsc{Prune}(C)$



$\textsc{AssignCenter}(C)$

- Pick a **random** vertex as **center** c proportional to vertex degrees
- Init 2-approx. decremental SSSP $\mathrm{A}_{\mathrm{HKN}}$ on C [Henzinger et al.'14]

Cluster Pruning – Dynamic Ball Growing

Dynamic Ball-Growing Process

- For rounds $i = 1$ to k:

  (1) Select a vertex c_i of $G_i = (V \setminus (B_1 \cup \ldots \cup B_{i-1}), E \setminus (E_1 \cup \ldots \cup E_{i-1}))$ with $G_1 = G$

  (2) Sample $R_i \sim \mathrm{Geom}(p)$ and grow ball B_i from c_i of radius R_i in G_i

Cluster Pruning – Dynamic Ball Growing

Dynamic Ball-Growing Process

- For rounds $i = 1$ to k:

  (1) Select a vertex c_i of $G_i = (V \setminus (B_1 \cup \ldots \cup B_{i-1}), E \setminus (E_1 \cup \ldots \cup E_{i-1}))$ with $G_1 = G$

  (2) Sample $R_i \sim \mathrm{Geom}(p)$ and grow ball B_i from c_i of radius R_i in G_i

Guarantees

- after each round i, $R_i = \mathcal{O}(p^{-1} \log n)$ with high probability
- for any edge $e \in E \setminus (E_1 \cup \ldots \cup E_k)$ the probability of e leaving a ball is at most p
- whenever a cluster is created we associate a dynamic ball-grow. process
- each edge can participate in at most $O(\log n)$ clusters
- deleted edges $E_1, \ldots, E_k$ don't see the values $R_1, \ldots, R_k$

Cluster Pruning – Running Time

Decremental approx. SSSP [Henzinger et al.'14]

(1) can maintain 2-approx to SSSP in $m^{1+o(1)}$ total update time

Local Ball Growing and Center Reassignments

(2) Can compute $B := \mathrm{Ball}_C(v, R)$ in $\mathcal{O}(\mathbf{vol}(B) \log \mathbf{vol}(B))$

(3) Total number of **center reassignments** is $\mathcal{O}(\log n_C)$

Cluster Pruning – Running Time

Decremental approx. SSSP [Henzinger et al.'14]

(1) can maintain 2-approx to SSSP in $m^{1+o(1)}$ total update time

Local Ball Growing and Center Reassignments

(2) Can compute $B := \mathrm{Ball}_C(v, R)$ in $\mathcal{O}(\mathbf{vol}(B) \log \mathbf{vol}(B))$

(3) Total number of **center reassignments** is $\mathcal{O}(\log n_C)$

Analysis

- Consider cluster C, charge runtime to calls $\textsc{AssignCenter}(C)$ and $\textsc{Prune}(C)$, **sans** B's
- Runtime of $\textsc{AssignCenter}$ is dominated by (1)
- By (3), total cost of $\textsc{AssignCenter}$ on C is $\mathcal{O}(m_C^{1+o(1)})$
- By (2), and as we remove each ball B with volume $\leq 1/2 \cdot \mathrm{vol}^*(C)$, charge $\mathcal{O}(\log m_C)$ to each edge in C for $O(m_c \log m_c)$ runtime
- Charged run time to C is $m_C^{1+o(1)}$, clusters are **disjoint**!
- As volume halves, we have $\mathcal{O}(\log n)$ levels, thus $m^{1+o(1)}$ total update time

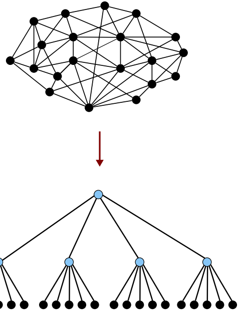
Decremental Probabilistic Tree Embedding

Theorem [Forster **G** Henzinger'21]

- Given a graph $G = (V, E)$ undergoing edge deletions, can maintain a **random** tree T of height $O(\log n)$ with

  (1) $O(\log^3 n)$ **expected stretch**, and

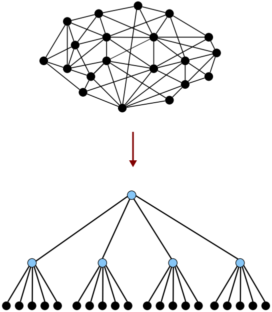
  (2) $m^{1+o(1)}$ **total update time**



Decremental Probabilistic Tree Embedding

Theorem [Forster **G** Henzinger'21]

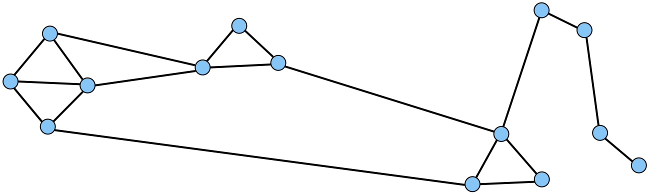
- Given a graph $G = (V, E)$ undergoing edge deletions, can maintain a **random** tree T of height $O(\log n)$ with
  (1) $O(\log^3 n)$ **expected stretch**, and
  (2) $m^{1+o(1)}$ **total update time**



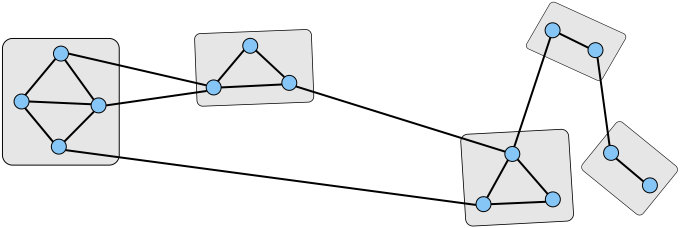
High Level Idea

- Apply decremental LDDs in a **non-recursive** way using **top-down** graph clustering.

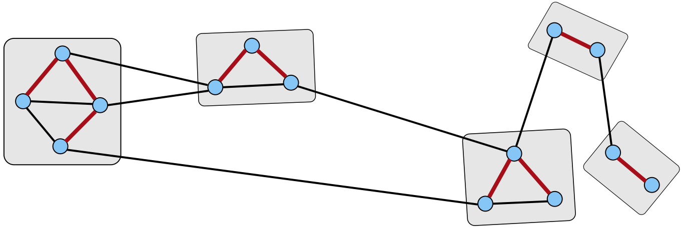
Attempt #1: Bottom-Up Clustering [AKPW'91]



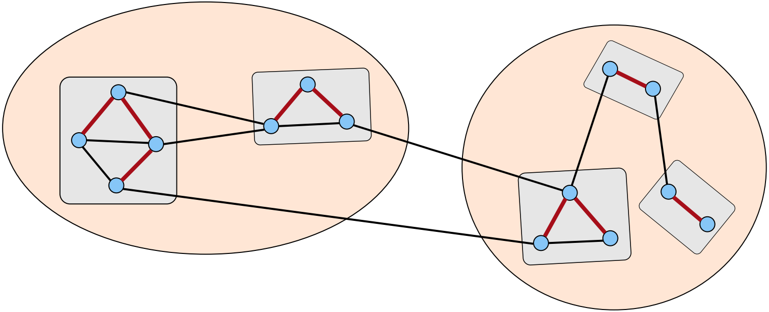
Attempt #1: Bottom-Up Clustering [AKPW'91]



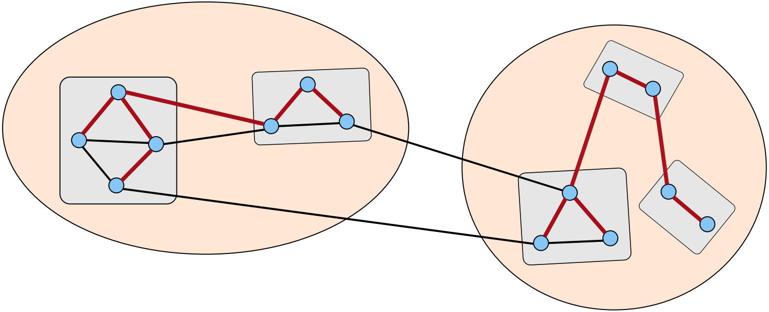
Attempt #1: Bottom-Up Clustering [AKPW'91]



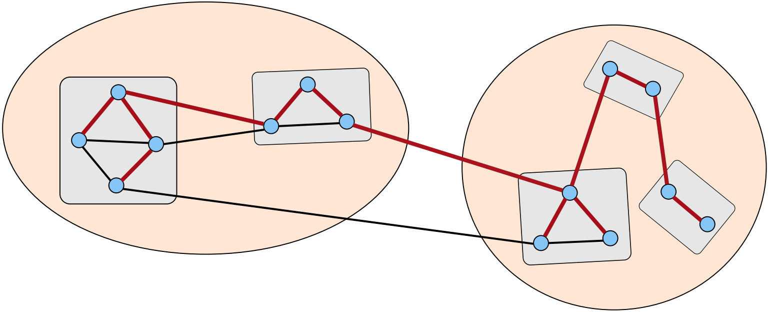
Attempt #1: Bottom-Up Clustering [AKPW'91]



Attempt #1: Bottom-Up Clustering [AKPW'91]

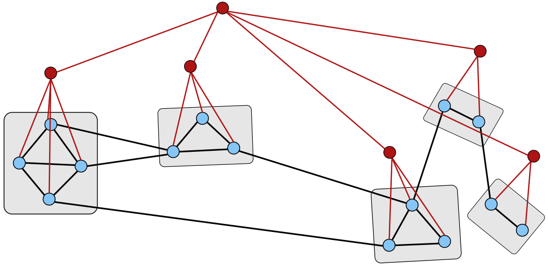


Attempt #1: Bottom-Up Clustering [AKPW'91]



- gives only **subpolynomial** expected stretch $2^{\sqrt{\log n}} = n^{o(1)}$
- requires fully-dynamic LDDs; deletions in one level translate to **insertions/deletions** in the levels below

Attempt #2: Decremental LDD to Decremental PTE



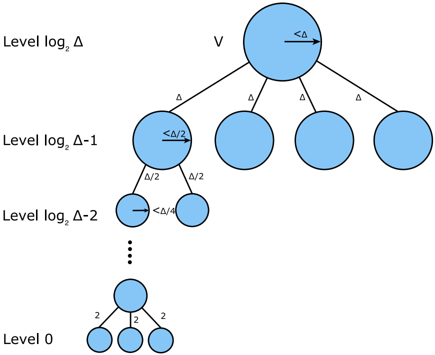
Recursive Top-Down Clustering [Bartal'96]

- Find an LDD with diameter $\Delta/2$ in G
- For each cluster C_i **recursively** find a rooted tree T_i with diameter $\Delta/4$
- Construct T by creating a **root** node v_G and connecting it to the root node of each T_i with weight Δ
- **Challenge:** difficult to control **recourse** – propagation of updates among decremental LDDs

Attempt #3: Decremental LDD to Decremental PTE

Iterative Top-Down Clustering

- Hierarchy **Invariant**: All inter-cluster edges at level i are deleted from the LDDs at level $i-1, \ldots, 0$
- Maintain a decremental probabilistic LDD for **each** level in the hierarchy to handle the deletions from the levels above
- Maintain cluster connections between neighbouring levels in the hierarchy so we have access to an explicit tree after each deletion



Fully Dynamic PTE

Decremental PTE + Static PTE = Fully Dynamic PTE

- Rebuild every k updates
- Pass deletions to decremental PTE T_{A} with stretch $\mathcal{O}(\log^3 n)$, height $\mathcal{O}(\log n)$
- Add inserted edge into set I, let U be the endpoints of I
- After each update:
  - Let $P = \bigcup_{v \in U} p_v$, where p_v is the path from v to root of T_{A}
  - Compute a (static) PTE T_{B} of $I \cup P$
  - Maintain $T_{\mathrm{C}} = (T_{\mathrm{A}} \setminus P) \cup T_{\mathrm{B}}$.

Fully Dynamic PTE

Decremental PTE + Static PTE = Fully Dynamic PTE

- Rebuild every k updates
- Pass deletions to decremental PTE T_{A} with stretch $\mathcal{O}(\log^3 n)$, height $\mathcal{O}(\log n)$
- Add inserted edge into set I, let U be the endpoints of I
- After each update:
  - Let $P = \bigcup_{v \in U} p_v$, where p_v is the path from v to root of T_{A}
  - Compute a (static) PTE T_{B} of $I \cup P$
  - Maintain $T_{\mathrm{C}} = (T_{\mathrm{A}} \setminus P) \cup T_{\mathrm{B}}$.

Analysis

- Expected stretch increases to $\mathcal{O}(\log^4 n)$ – due to T_{B}
- Runtime: $m^{1+o(1)}/k + k \log^2 n = m^{1/2+o(1)}$, optimized when $k = m^{1/2}$

Fully Dynamic PTE

Decremental PTE + Static PTE = Fully Dynamic PTE

- Rebuild every k updates
- Pass deletions to decremental PTE T_{A} with stretch $\mathcal{O}(\log^3 n)$, height $\mathcal{O}(\log n)$
- Add inserted edge into set I, let U be the endpoints of I
- After each update:
  - Let $P = \bigcup_{v \in U} p_v$, where p_v is the path from v to root of T_{A}
  - Compute a (static) PTE T_{B} of $I \cup P$
  - Maintain $T_{\mathrm{C}} = (T_{\mathrm{A}} \setminus P) \cup T_{\mathrm{B}}$.

Analysis

- Expected stretch increases to $\mathcal{O}(\log^4 n)$ – due to T_{B}
- Runtime: $m^{1+o(1)}/k + k \log^2 n = m^{1/2+o(1)}$, optimized when $k = m^{1/2}$

Extensions

- Can generalize the above approach to multiple levels
- Requires bounding the number of **changes** to the aux. graph $I \cup P$

Fully Dynamic APSP via Dynamic PTE

$\textsc{Preprocessing}(G)$

- Maintain $\mathcal{O}(\log n)$ copies of dynamic PTEs $\{T_i\}_i$

$\textsc{Insert}/\textsc{Delete}(e)$

- Pass the insertion/deletion of e to each T_i

$\textsc{Query}(s, t)$

- Compute shortest path from s to t on each T_i
- Return the one that attains the minimum

Fully Dynamic APSP via Dynamic PTE

Preprocessing(G)

- Maintain $\mathcal{O}(\log n)$ copies of dynamic PTEs $\{T_i\}_i$

Insert/Delete(e)

- Pass the insertion/deletion of e to each T_i

Query(s, t)

- Compute shortest path from s to t on each T_i
- Return the one that attains the minimum

- approx: $\mathcal{O}(\log n)^{3i-2}$, updateT: $m^{1/i+o(1)}$, queryT: $\mathcal{O}(\log n)^{5/2}$

Summary and Future Directions

Summary

- The first fully dynamic algorithm for probabilistic tree embedding with competitive guarantees on expected stretch and update time
- Applied to All-Pairs Shortest Path and Buy-At-Bulk Network Design

Summary and Future Directions

Summary

- The first fully dynamic algorithm for probabilistic tree embedding with competitive guarantees on expected stretch and update time
- Applied to All-Pairs Shortest Path and Buy-At-Bulk Network Design

Future Directions

- Improve the expected stretch and running time of our dynamic PTE to polylogarithmic, respectively
- Apply our dynamic PTE to other optimization problems
- Transfer our ideas to fully dynamic cut-based tree sparsifiers with polylogarithmic guarantees

Summary and Future Directions

Summary

- The first fully dynamic algorithm for probabilistic tree embedding with competitive guarantees on expected stretch and update time
- Applied to All-Pairs Shortest Path and Buy-At-Bulk Network Design

Future Directions

- Improve the expected stretch and running time of our dynamic PTE to polylogarithmic, respectively
- Apply our dynamic PTE to other optimization problems
- Transfer our ideas to fully dynamic cut-based tree sparsifiers with polylogarithmic guarantees

Thank you!