Algorithms: New Techniques for Old Problems

Jason Li
Carnegie Mellon University

October 5, 2020

"Meta"-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field

“Meta”-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field
- Many different sub-fields, and many techniques developed
  - Graph sparsification for fast algorithms, LP rounding for approximation algorithms

“Meta”-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field
- Many different sub-fields, and many techniques developed
  - Graph sparsification for fast algorithms, LP rounding for approximation algorithms
- “Meta”: general technique applicable to many areas
  - greedy algorithms, linear programming, amortized analysis

“Meta”-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field
- Many different sub-fields, and many techniques developed
  - Graph sparsification for fast algorithms, LP rounding for approximation algorithms
- “Meta”: general technique applicable to many areas
  - greedy algorithms, linear programming, amortized analysis

New meta-techniques

“Meta”-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field
- Many different sub-fields, and many techniques developed
  - Graph sparsification for fast algorithms, LP rounding for approximation algorithms
- “Meta”: general technique applicable to many areas
  - greedy algorithms, linear programming, amortized analysis

New meta-techniques

- Preconditioning [Spielman-Teng’04]
  - “What if the input graph is an expander?”

“Meta”-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field
- Many different sub-fields, and many techniques developed
  - Graph sparsification for fast algorithms, LP rounding for approximation algorithms
- “Meta”: general technique applicable to many areas
  - greedy algorithms, linear programming, amortized analysis

New meta-techniques

- Preconditioning [Spielman-Teng’04]
  - “What if the input graph is an expander?”
- Iterative methods [CKMST’11, Sherman’13]
  - “Continuous formulation of a discrete problem?”

"Meta"-techniques for Algorithm Design

- Algorithms is a broad, rich, and fast-growing field
- Many different sub-fields, and many techniques developed
  - Graph sparsification for fast algorithms, LP rounding for approximation algorithms
- "Meta": general technique applicable to many areas
  - greedy algorithms, linear programming, amortized analysis

New meta-techniques

- Preconditioning [Spielman-Teng'04]
  - "What if the input graph is an expander?"
- Iterative methods [CKMST'11, Sherman'13]
  - "Continuous formulation of a discrete problem?"
- I work on preconditioning and iterative methods for graphs

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free
- Smoothed analysis: worst-case + random perturbation, think "average case"

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free
- Smoothed analysis: worst-case + random perturbation, think “average case”

Preconditioning

- **“condition” the input while preserving full generality**

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free
- Smoothed analysis: worst-case + random perturbation, think “average case”

Preconditioning

- **“condition” the input while preserving full generality**
  - Lets us assume “w.l.o.g.” that input “behaves like random”
  - Worst case = average case!

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free
- Smoothed analysis: worst-case + random perturbation, think "average case"

Preconditioning

- **"condition" the input while preserving full generality**
  - Lets us assume "w.l.o.g." that input "behaves like random"
  - Worst case = average case!

...for graphs: expanders

- "well-conditioned" $\sim$ high **conductance** (expander)

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free
- Smoothed analysis: worst-case + random perturbation, think "average case"

Preconditioning

- **"condition" the input while preserving full generality**
  - Lets us assume "w.l.o.g." that input "behaves like random"
  - Worst case = average case!

...for graphs: expanders

- "well-conditioned" $\sim$ high **conductance** (expander)
- Easy case: when input graph is an expander

Preconditioning for Graph Algorithms

- In general, worst-case problems are difficult

Beyond worst-case

- Parameterization: degree $\leq \Delta$, treewidth $\leq k$, H-minor free
- Smoothed analysis: worst-case + random perturbation, think "average case"

Preconditioning

- **"condition" the input while preserving full generality**
  - Lets us assume "w.l.o.g." that input "behaves like random"
  - Worst case = average case!

...for graphs: expanders

- "well-conditioned" $\sim$ high **conductance** (expander)
- Easy case: when input graph is an expander
- Precondition by **expander decomposition**

Expanders

- Given unweighted, undirected graph G, its **conductance** is

$$\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}}$$

where $\mathbf{vol}(S) = \sum_{v \in S} \deg(v)$.

Expanders

- Given unweighted, undirected graph G, its **conductance** is

$$\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}}$$

  where $\mathbf{vol}(S) = \sum_{v \in S} \deg(v)$.
- G is a ϕ**-expander** if $\Phi(G) \geq \phi$

Expanders

- Given unweighted, undirected graph G, its **conductance** is

$$\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}}$$

  where $\mathbf{vol}(S) = \sum_{v \in S} \deg(v)$.
- G is a ϕ**-expander** if $\Phi(G) \geq \phi$
- Cheeger's inequality: normalized Laplacian has all eigenvalues in $[\phi^2/2, 1]$ (except 0)

Expanders

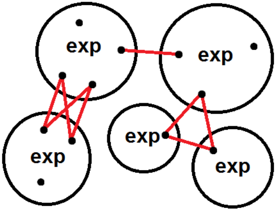
- Given unweighted, undirected graph G, its **conductance** is

$$\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}}$$

  where $\mathbf{vol}(S) = \sum_{v \in S} \deg(v)$.
- G is a ϕ**-expander** if $\Phi(G) \geq \phi$
- Cheeger's inequality: normalized Laplacian has all eigenvalues in $[\phi^2/2, 1]$ (except 0)
- Examples of ϕ-expanders ($\phi \geq \frac{1}{\text{polylog}(n)}$): hypercube, random graphs, peer-to-peer networks

Expander decomposition

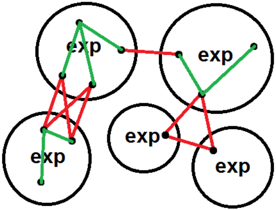
- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders



Expander decomposition

- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

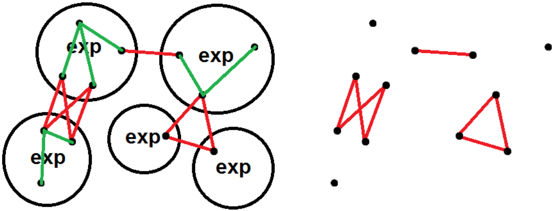


Expander decomposition

- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

(2) Solve recursively on inter-cluster edges (half the size)

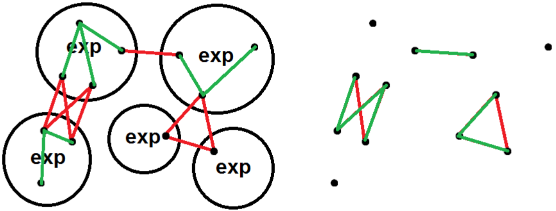


Expander decomposition

- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

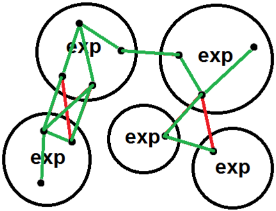
(2) Solve recursively on inter-cluster edges (half the size)



Expander decomposition

- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)
(2) Solve recursively on inter-cluster edges (half the size)
(3) Stitch all solutions together (problem-specific)



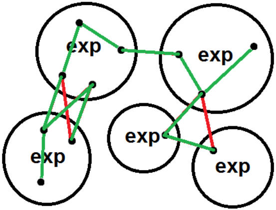
Expander decomposition

- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

(2) Solve recursively on inter-cluster edges (half the size)

(3) Stitch all solutions together (problem-specific)



Expander decomposition

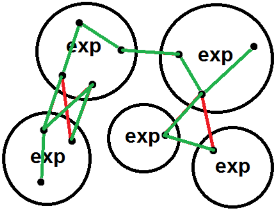
- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

(2) Solve recursively on inter-cluster edges (half the size)

(3) Stitch all solutions together (problem-specific)

- If can handle (3), then reduces to case when G is expander



Expander decomposition

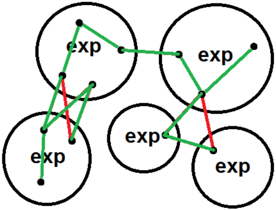
- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

(2) Solve recursively on inter-cluster edges (half the size)

(3) Stitch all solutions together (problem-specific)

- If can handle (3), then reduces to case when G is expander



- Lets us assume “w.l.o.g.” that input is an expander

Expander decomposition

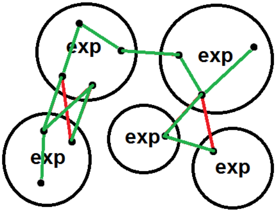
- Given graph $G = (V, E)$, partition V into $V_1, \ldots, V_k$ s.t.
  - Each induced graph $G[V_i]$ is a ϕ-expander for $\phi \geq 1/n^{o(1)}$
  - At most half of edges go between expanders

(1) Solve problem on each expander (easy case)

(2) Solve recursively on inter-cluster edges (half the size)

(3) Stitch all solutions together (problem-specific)

- If can handle (3), then reduces to case when G is expander



- Lets us assume "w.l.o.g." that input is an expander
- [CG**L**NPS'20]: deterministic exp. decomp. in $m^{1+o(1)}$ time
  - Derandomize cut-matching game [KRV'07]

Expander Decomposition: Applications

Fully dynamic connectivity

- Given online sequence of edge inserts and deletions, output whether G is connected at each time

Expander Decomposition: Applications

Fully dynamic connectivity

- Given online sequence of edge inserts and deletions, output whether G is connected at each time
- Amortized polylog(n) [HdLT'01], randomized worst-case [KM'13, NSW'17]

Expander Decomposition: Applications

Fully dynamic connectivity

- Given online sequence of edge inserts and deletions, output whether G is connected at each time
- Amortized polylog(n) [HdLT'01], randomized worst-case [KM'13, NSW'17]
- [CG**L**NPS'20]: deterministic in $n^{o(1)}$ worst-case update time, improves upon $\tilde{O}(\sqrt{n})$ [Frederickson'85]

Expander Decomposition: Applications

Fully dynamic connectivity

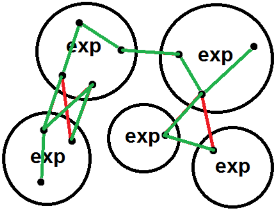
- Given online sequence of edge inserts and deletions, output whether G is connected at each time
- Amortized polylog(n) [HdLT'01], randomized worst-case [KM'13, NSW'17]
- [CG**L**NPS'20]: deterministic in $n^{o(1)}$ worst-case update time, improves upon $\tilde{O}(\sqrt{n})$ [Frederickson'85]
  - Expanders are easy: takes many deletions to disconnect an expander $\implies$ more running time allowed

Expander Decomposition: Applications

Fully dynamic connectivity

- Given online sequence of edge inserts and deletions, output whether G is connected at each time
- Amortized polylog(n) [HdLT'01], randomized worst-case [KM'13, NSW'17]
- [CG**L**NPS'20]: deterministic in $n^{o(1)}$ worst-case update time, improves upon $\tilde{O}(\sqrt{n})$ [Frederickson'85]
    - Expanders are easy: takes many deletions to disconnect an expander $\implies$ more running time allowed
    - Stitching together expanders:

      Dynamic version: framework of [NSW'17]



Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in polylog(n) many s–t max-flows

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in $\text{polylog}(n)$ many s–t max-flows
    - $\tilde{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$ time using Goldberg-Rao

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in polylog(n) many s–t max-flows
    - $\tilde{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$ time using Goldberg-Rao
    - Expanders are easy: in a ϕ-expander, smaller side of min-cut has $\leq 1/\phi$ vertices (this talk)

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in polylog(n) many s–t max-flows
  - $\tilde{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$ time using Goldberg-Rao
  - Expanders are easy: in a ϕ-expander, smaller side of min-cut has $\leq 1/\phi$ vertices (this talk)
  - General case: vertex sparsification (skip)

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in $\mathrm{polylog}(n)$ many s–t max-flows
  - $\tilde{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$ time using Goldberg-Rao
  - Expanders are easy: in a ϕ-expander, smaller side of min-cut has $\leq 1/\phi$ vertices (this talk)
  - General case: vertex sparsification (skip)
- [GK**L**TW'20]: deterministic in $m^{1+o(1)}$ time (skip)

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in polylog(n) many s–t max-flows
  - $\tilde{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$ time using Goldberg-Rao
  - Expanders are easy: in a ϕ-expander, smaller side of min-cut has $\leq 1/\phi$ vertices (this talk)
  - General case: vertex sparsification (skip)
- [GK**L**TW'20]: deterministic in $m^{1+o(1)}$ time (skip)
  - Compute skeleton graph in $m^{1+o(1)}$ and use Karger'00

Expander Decomposition: Applications

Deterministic global min-cut

- Given undirected, weighted graph, remove smallest weight set of edges to disconnect graph
- Randomized $\tilde{O}(m)$ [Karger'00], deterministic $\tilde{O}(mn)$ [HO'92]
- [**L**P'20]: deterministic in polylog(n) many s–t max-flows
  - $\tilde{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$ time using Goldberg-Rao
  - Expanders are easy: in a ϕ-expander, smaller side of min-cut has $\leq 1/\phi$ vertices (this talk)
  - General case: vertex sparsification (skip)
- [GK**L**TW'20]: deterministic in $m^{1+o(1)}$ time (skip)
  - Compute skeleton graph in $m^{1+o(1)}$ and use Karger'00
  - Derandomize graph sparsification by random sampling [BK'96]

Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.

Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
  Proof:
  - $\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}} \geq \phi$

Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
  Proof:
  - $\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}} \geq \phi$
  - Let S be the side of min-cut with $\mathbf{vol}(S) \leq \mathbf{vol}(V \setminus S)$
    $\implies \frac{|E(S, V \setminus S)|}{\mathbf{vol}(S)} \geq \phi$

Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
  Proof:
  - $\Phi(G) = \min_{S \subseteq V} \frac{|E(S,V \setminus S)|}{\min\{\mathbf{vol}(S),\mathbf{vol}(V \setminus S)\}} \geq \phi$
  - Let S be the side of min-cut with $\mathbf{vol}(S) \leq \mathbf{vol}(V \setminus S)$
    $\implies \frac{|E(S,V \setminus S)|}{\mathbf{vol}(S)} \geq \phi$
  - $\mathbf{vol}(S) = \sum_{v \in S} \deg(v) \geq \sum_{v \in S} \lambda = \lambda |S|$ (λ = mincut)

Deterministic Min-cut: Proofs

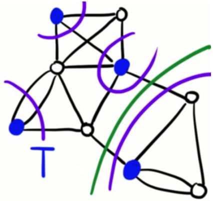
- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
  Proof:
  - $\Phi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}} \geq \phi$
  - Let S be the side of min-cut with $\mathbf{vol}(S) \leq \mathbf{vol}(V \setminus S)$
    $\implies \frac{|E(S, V \setminus S)|}{\mathbf{vol}(S)} \geq \phi$
  - $\mathbf{vol}(S) = \sum_{v \in S} \deg(v) \geq \sum_{v \in S} \lambda = \lambda|S|$ (λ = mincut)
  - $\phi \leq \frac{|E(S, V \setminus S)|}{\mathbf{vol}(S)} \leq \frac{\lambda}{\lambda|S|} \implies |S| \leq 1/\phi$

Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
- Theorem (minimum isolating cuts) [**L**P'20]: for any subset $T \subseteq V$ ($|T| \geq 2$), can compute the following in $\lceil \log_2 |T| \rceil$ calls to s–t max-flow: for each vertex $t \in T$, the min-cut separating t from $T \setminus t$

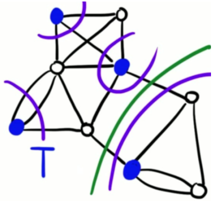
Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
- Theorem (minimum isolating cuts) [**L**P'20]: for any subset $T \subseteq V$ ($|T| \geq 2$), can compute the following in $\lceil \log_2 |T| \rceil$ calls to s–t max-flow: for each vertex $t \in T$, the min-cut separating t from $T \setminus t$
- If $|S \cap T| = 1$, then recover min-cut



Deterministic Min-cut: Proofs

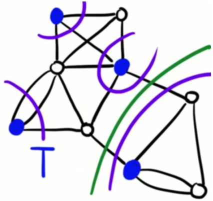
- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
- Theorem (minimum isolating cuts) [**L**P'20]: for any subset $T \subseteq V$ ($|T| \geq 2$), can compute the following in $\lceil \log_2 |T| \rceil$ calls to s–t max-flow: for each vertex $t \in T$, the min-cut separating t from $T \setminus t$
- If $|S \cap T| = 1$, then recover min-cut



- Randomized: sample each vertex into T w.p. $1/k$

Deterministic Min-cut: Proofs

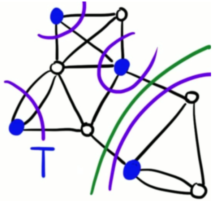
- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \leq 1/\phi$ vertices.
- Theorem (minimum isolating cuts) [**L**P'20]: for any subset $T \subseteq V$ ($|T| \geq 2$), can compute the following in $\lceil \log_2 |T| \rceil$ calls to s–t max-flow: for each vertex $t \in T$, the min-cut separating t from $T \setminus t$
- If $|S \cap T| = 1$, then recover min-cut



- Randomized: sample each vertex into T w.p. $1/k$
- Derandomization: overhead of $k^{O(1)} \log n$

Deterministic Min-cut: Proofs

- Theorem: if G is a ϕ-expander, then one side of min-cut has $k \le 1/\phi$ vertices.
- Theorem (minimum isolating cuts) [**L**P'20]: for any subset $T \subseteq V$ ($|T| \ge 2$), can compute the following in $\lceil \log_2 |T| \rceil$ calls to s–t max-flow: for each vertex $t \in T$, the min-cut separating t from $T \setminus t$
- If $|S \cap T| = 1$, then recover min-cut



- Randomized: sample each vertex into T w.p. $1/k$
- Derandomization: overhead of $k^{O(1)} \log n$
- Theorem: if G is a ϕ-expander, then deterministic min-cut in $\phi^{-O(1)}\text{polylog}(n)$ many s–t max-flows

Preconditioning: Future Directions

Expander decomposition

- Deterministic graph sparsification: $(1+\epsilon)$-approximate cut sparsifier in near-linear time?

Preconditioning: Future Directions

Expander decomposition

- Deterministic graph sparsification: $(1+\epsilon)$-approximate cut sparsifier in near-linear time?
    - [GK**L**TW'20] Skeleton graph does not preserve very large cuts up to $(1+\epsilon)$

Preconditioning: Future Directions

Expander decomposition

- Deterministic graph sparsification: $(1+\epsilon)$-approximate cut sparsifier in near-linear time?
  - [GK**L**TW'20] Skeleton graph does not preserve very large cuts up to $(1+\epsilon)$
- Gomory-Hu tree in $o(n)$ many s–t max-flow?

Preconditioning: Future Directions

Expander decomposition

- Deterministic graph sparsification: $(1+\epsilon)$-approximate cut sparsifier in near-linear time?
  - [GK**L**TW'20] Skeleton graph does not preserve very large cuts up to $(1+\epsilon)$
- Gomory-Hu tree in $o(n)$ many s–t max-flow?
  - Expanders are easy, but do not know how to stitch together expanders!

Preconditioning: Future Directions

Expander decomposition

- Deterministic graph sparsification: $(1+\epsilon)$-approximate cut sparsifier in near-linear time?
  - [GK**L**TW'20] Skeleton graph does not preserve very large cuts up to $(1+\epsilon)$
- Gomory-Hu tree in $o(n)$ many s–t max-flow?
  - Expanders are easy, but do not know how to stitch together expanders!
- Directed global min-cut in $o(mn)$ time?

Preconditioning: Future Directions

Expander decomposition

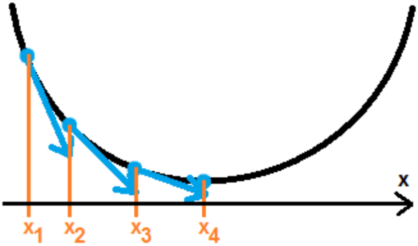
- Deterministic graph sparsification: $(1+\epsilon)$-approximate cut sparsifier in near-linear time?
  - [GK**L**TW'20] Skeleton graph does not preserve very large cuts up to $(1+\epsilon)$
- Gomory-Hu tree in $o(n)$ many s–t max-flow?
  - Expanders are easy, but do not know how to stitch together expanders!
- Directed global min-cut in $o(mn)$ time?
  - Requires directed version of expander decomposition [Louis'10], or other preconditioning

Iterative Methods for Graph Algorithms

- Formulate discrete problem as continuous optimization

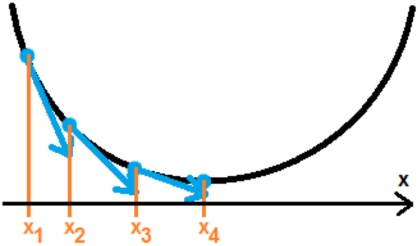
Iterative Methods for Graph Algorithms

- Formulate discrete problem as continuous optimization
- Start from a trivial solution and iteratively improve it



Iterative Methods for Graph Algorithms

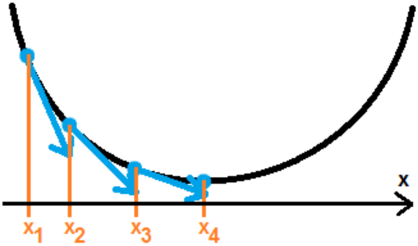
- Formulate discrete problem as continuous optimization
- Start from a trivial solution and iteratively improve it



- Goal: bound # iterations to be small (e.g. polylogarithmic)

Iterative Methods for Graph Algorithms

- Formulate discrete problem as continuous optimization
- Start from a trivial solution and iteratively improve it



- Goal: bound # iterations to be small (e.g. polylogarithmic)
- multiplicative weight update / gradient descent in online algorithms, convex programming, machine learning

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a continuous problem: $\min \|f\|_\infty : Af = b$

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a continuous problem: min $\|f\|_\infty : Af = b$
- Sherman '13 for max-flow: $(1+\epsilon)$-approx. in $\tilde{O}(m)$ time

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a continuous problem: min $\|f\|_\infty : Af = b$
- Sherman '13 for max-flow: $(1+\epsilon)$-approx. in $\tilde{O}(m)$ time
  - compute "oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\text{maxflow}(G, b) \leq \|Rb\|_\infty \leq \alpha \cdot \text{maxflow}(G, b)$

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a continuous problem: $\min \|f\|_\infty : Af = b$
- Sherman '13 for max-flow: $(1+\epsilon)$-approx. in $\tilde{O}(m)$ time
  - compute "oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\text{maxflow}(G, b) \le \|Rb\|_\infty \le \alpha \cdot \text{maxflow}(G, b)$
  - minimize $\|f\|_\infty + 2\alpha \underbrace{\|R(b - Af)\|_\infty}_{\text{“regularizer”: } \approx \text{ cost to route remaining demands } b - Af} : f \in \mathbb{R}^E$,
    solve up to $(1+\epsilon)$ factor using MWU / gradient descent

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a <u>continuous</u> problem: min $\|f\|_\infty : Af = b$
- Sherman '13 for max-flow: $(1+\epsilon)$-approx. in $\tilde{O}(m)$ time
  - compute "oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\text{maxflow}(G, b) \le \|Rb\|_\infty \le \alpha \cdot \text{maxflow}(G, b)$
  - minimize $\|f\|_\infty + 2\alpha \underbrace{\|R(b - Af)\|_\infty}_{\text{"regularizer": } \approx \text{ cost to route remaining demands } b - Af} : f \in \mathbb{R}^E$,
    solve up to $(1+\epsilon)$ factor using MWU / gradient descent
    - Preconditions $A' \leftarrow RA$, $b' \leftarrow Rb$, to minimize $\|f\|_\infty + 2\alpha \|b' - A'f\|_\infty$

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a <u>continuous</u> problem: $\min \|f\|_\infty : Af = b$
- Sherman '13 for max-flow: $(1+\epsilon)$-approx. in $\tilde{O}(m)$ time
  - compute "oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\text{maxflow}(G, b) \leq \|Rb\|_\infty \leq \alpha \cdot \text{maxflow}(G, b)$
  - minimize $\|f\|_\infty + 2\alpha \underbrace{\|R(b - Af)\|_\infty}_{\text{“regularizer”: } \approx \text{ cost to route remaining demands } b - Af} : f \in \mathbb{R}^E$,
    solve up to $(1+\epsilon)$ factor using MWU / gradient descent
    - Preconditions $A' \leftarrow RA$, $b' \leftarrow Rb$, to minimize $\|f\|_\infty + 2\alpha \|b' - A'f\|_\infty$
  - Recursively route remaining demands $b - Af$ (small)

Max-flow as continuous problem

- Integrality of flow: max discrete flow = max continuous flow
- Max-flow as a continuous problem: min $\|f\|_\infty : Af = b$
- Sherman '13 for max-flow: $(1+\epsilon)$-approx. in $\tilde{O}(m)$ time
  - compute "oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\mathrm{maxflow}(G, b) \le \|Rb\|_\infty \le \alpha \cdot \mathrm{maxflow}(G, b)$
  - minimize $\|f\|_\infty + 2\alpha \underbrace{\|R(b - Af)\|_\infty}_{\text{"regularizer": } \approx \text{ cost to route remaining demands } b - Af} : f \in \mathbb{R}^E$,
    solve up to $(1+\epsilon)$ factor using MWU / gradient descent
    - Preconditions $A' \leftarrow RA$, $b' \leftarrow Rb$, to minimize $\|f\|_\infty + 2\alpha \|b' - A'f\|_\infty$
  - Recursively route remaining demands $b - Af$ (small)
  - $(1+\epsilon)$-approximate max-flow in $\tilde{O}(m)$ time

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: min $\|f\|_1 : Af = b$

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: min $\|f\|_1 : Af = b$
- Sherman '17 for transshipment

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: $\min \|f\|_1 : Af = b$
- Sherman '17 for transshipment
  - compute "ℓ_1-oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\mathrm{tship}(G, b) \le \|Rb\|_1 \le \alpha \cdot \mathrm{tship}(G, b)$

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: $\min \|f\|_1 : Af = b$
- Sherman '17 for transshipment
  - compute "ℓ_1-oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\mathrm{tship}(G, b) \leq \|Rb\|_1 \leq \alpha \cdot \mathrm{tship}(G, b)$
  - minimize $\|f\|_1 + 2\alpha \|R(b - Af)\|_1 : f \in \mathbb{R}^E$

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: $\min \|f\|_1 : Af = b$
- Sherman '17 for transshipment
  - compute "ℓ_1-oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\mathrm{tship}(G, b) \leq \|Rb\|_1 \leq \alpha \cdot \mathrm{tship}(G, b)$
  - minimize $\|f\|_1 + 2\alpha \|R(b - Af)\|_1 : f \in \mathbb{R}^E$
  - $(1+\epsilon)$-approximate transshipment in $m^{1+o(1)}$ time

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: $\min \|f\|_1 : Af = b$
- Sherman '17 for transshipment
  - compute "ℓ_1-oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\mathrm{tship}(G, b) \leq \|Rb\|_1 \leq \alpha \cdot \mathrm{tship}(G, b)$
  - minimize $\|f\|_1 + 2\alpha \|R(b - Af)\|_1 : f \in \mathbb{R}^E$
  - $(1+\epsilon)$-approximate transshipment in $m^{1+o(1)}$ time
- [**L**'20]: can compute "ℓ_1-oblivious routing" matrix R in $\tilde{O}(m)$ time, for $\alpha = \mathrm{polylog}(n)$

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: $\min \|f\|_1 : Af = b$
- Sherman '17 for transshipment
  - compute "ℓ_1-oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    $\mathrm{tship}(G, b) \leq \|Rb\|_1 \leq \alpha \cdot \mathrm{tship}(G, b)$
  - minimize $\|f\|_1 + 2\alpha \|R(b - Af)\|_1 : f \in \mathbb{R}^E$
  - $(1 + \epsilon)$-approximate transshipment in $m^{1+o(1)}$ time
- [**L**'20]: can compute "ℓ_1-oblivious routing" matrix R in $\tilde{O}(m)$ time, for $\alpha = \mathrm{polylog}(n)$
- [**L**'20] $\implies$ $(1 + \epsilon)$-approx. transshipment in $\tilde{O}(m)$ time

Transshipment as continuous problem

- Transshipment: uncapacitated min-cost flow, also integral
- Transshipment as a continuous problem: min $\|f\|_1 : Af = b$
- Sherman '17 for transshipment
  - compute "ℓ_1-oblivious routing" matrix $R \in \mathbb{R}^{? \times V}$ such that for all demands $b \in \mathbb{R}^V$,
    tship$(G, b) \leq \|Rb\|_1 \leq \alpha \cdot$ tship(G, b)
  - minimize $\|f\|_1 + 2\alpha \|R(b - Af)\|_1 : f \in \mathbb{R}^E$
  - $(1 + \epsilon)$-approximate transshipment in $m^{1+o(1)}$ time
- [**L**'20]: can compute "ℓ_1-oblivious routing" matrix R in $\tilde{O}(m)$ time, for $\alpha = \text{polylog}(n)$
- [**L**'20] $\implies$ $(1 + \epsilon)$-approx. transshipment in $\tilde{O}(m)$ time
- Main technical contribution: ℓ_1-oblivious routing scheme in $\tilde{O}(m)$ time given an ℓ_1-embedding of the graph

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)
- Claim: Exact SSSP reduces to exact transshipment with $n-1$ supply at s and 1 demand at each $v \neq s$
  Proof:

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)
- Claim: Exact SSSP reduces to exact transshipment with $n-1$ supply at s and 1 demand at each $v \neq s$
  Proof:
  - Must send 1 unit flow $s \to v$ for all $v \neq s$

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)
- Claim: Exact SSSP reduces to exact transshipment with $n-1$ supply at s and 1 demand at each $v \neq s$
  Proof:
  - Must send 1 unit flow $s \to v$ for all $v \neq s$
  - Uncapacitated, so different $s \to v$ flows don't interfere
    $\implies$ might as well send along shortest $s \to v$ path

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)
- Claim: Exact SSSP reduces to exact transshipment with $n-1$ supply at s and 1 demand at each $v \neq s$
  Proof:
  - Must send 1 unit flow $s \to v$ for all $v \neq s$
  - Uncapacitated, so different $s \to v$ flows don't interfere $\implies$ might as well send along shortest $s \to v$ path
  - For each $v \neq s$, its distance from s is cost of $s \to v$ flow

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)
- Claim: Exact SSSP reduces to exact transshipment with $n-1$ supply at s and 1 demand at each $v \neq s$
  Proof:
  - Must send 1 unit flow $s \to v$ for all $v \neq s$
  - Uncapacitated, so different $s \to v$ flows don't interfere $\implies$ might as well send along shortest $s \to v$ path
  - For each $v \neq s$, its distance from s is cost of $s \to v$ flow
- [BFKL'16, **L**'20] Reduce approx. SSSP to approx. transshipment

Transshipment and SSSP

Single-source shortest path in parallel

- [**L**'20] $\implies$ $(1+\epsilon)$-approx. SSSP in $\tilde{O}(m)$ work and polylog(n) time
  - Previous best was $m^{1+\delta}$ work and polylog(n) time [Cohen'00] via hopsets (combinatorial)
- Claim: Exact SSSP reduces to exact transshipment with $n-1$ supply at s and 1 demand at each $v \neq s$
  Proof:
  - Must send 1 unit flow $s \to v$ for all $v \neq s$
  - Uncapacitated, so different $s \to v$ flows don't interfere $\implies$ might as well send along shortest $s \to v$ path
  - For each $v \neq s$, its distance from s is cost of $s \to v$ flow
- [BFKL'16, **L**'20] Reduce approx. SSSP to approx. transshipment
- <u>Iterative methods are inherently parallelizable</u>

Iterative Methods: Future Directions

Directed graphs

- Approximate transshipment and SSSP on directed graphs? Both sequential and parallel

Iterative Methods: Future Directions

Directed graphs

- Approximate transshipment and SSSP on directed graphs? Both sequential and parallel
    - [CFR'20] Parallel SSSP on directed graphs in $\tilde{O}(m)$ work and $n^{1/2+o(1)}$ time via hopsets

Iterative Methods: Future Directions

Directed graphs

- Approximate transshipment and SSSP on directed graphs? Both sequential and parallel
  - [CFR'20] Parallel SSSP on directed graphs in $\tilde{O}(m)$ work and $n^{1/2+o(1)}$ time via hopsets
- Exact SSSP reduces to $O(1)$-approximate SSSP on directed graphs + "subtractive triangle inequality" [KS'97] $\implies$ parallel exact SSSP

Iterative Methods: Future Directions

Directed graphs

- Approximate transshipment and SSSP on directed graphs? Both sequential and parallel
  - [CFR'20] Parallel SSSP on directed graphs in $\tilde{O}(m)$ work and $n^{1/2+o(1)}$ time via hopsets
- Exact SSSP reduces to $O(1)$-approximate SSSP on directed graphs + "subtractive triangle inequality" [KS'97] $\implies$ parallel exact SSSP
  - [**L**'20] iterative methods obtain subtractive triangle inequality for free (hopsets do not!)

Iterative Methods: Future Directions

Directed graphs

- Approximate transshipment and SSSP on directed graphs? Both sequential and parallel
  - [CFR'20] Parallel SSSP on directed graphs in $\tilde{O}(m)$ work and $n^{1/2+o(1)}$ time via hopsets
- Exact SSSP reduces to $O(1)$-approximate SSSP on <u>directed</u> graphs + "subtractive triangle inequality" [KS'97] $\implies$ parallel exact SSSP
  - [**L**'20] iterative methods obtain subtractive triangle inequality for free (hopsets do not!)
  - $\tilde{O}(m)$ work, $o(n)$ time directed?

Iterative Methods: Future Directions

Directed graphs

- Approximate transshipment and SSSP on directed graphs? Both sequential and parallel
  - [CFR'20] Parallel SSSP on directed graphs in $\tilde{O}(m)$ work and $n^{1/2+o(1)}$ time via hopsets
- Exact SSSP reduces to $O(1)$-approximate SSSP on directed graphs + "subtractive triangle inequality" [KS'97] $\implies$ parallel exact SSSP
  - [**L**'20] iterative methods obtain subtractive triangle inequality for free (hopsets do not!)
  - $\tilde{O}(m)$ work, $o(n)$ time directed?
  - Sherman's framework breaks down on directed graphs. Directed version?