Combinatorial Max-Flow via Weighted Push-Relabel on Shortcut Graphs

Jason Li (CMU)

Joint work with Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, Ta-Wei Tu

Max-Flow

Input: directed (uncap.) graph G=(V,E) and vertices s,t

Find max # of edge-disjoint s-t paths <=> Find s-t flow of maximum value s.t. each edge has congestion <= 1

History and Our Results

Combinatorial algorithms: Ford-Fulkerson, Dinic's, Edmonds-Karp, Sleator-Tarjan

--> Goldberg-Rao (1998): $m^{1.5}$ time

History and Our Results

Combinatorial algorithms: Ford-Fulkerson, Dinic's, Edmonds-Karp, Sleator-Tarjan

--> Goldberg-Rao (1998): $m^{1.5}$ time

Continuous optimization: Daitch-Spielman, CKMST, Madry, Sherman (approx)

--> CKLPPS (2022): $m^{1+o(1)}$ time

History and Our Results

CKLPPS (2022): $m^{1+o(1)}$ time

- max-flow has been "solved", but is there a fast combinatorial algorithm?

History and Our Results

CKLPPS (2022): $m^{1+o(1)}$ time

- max-flow has been "solved", but is there a fast combinatorial algorithm?
- augmenting paths easier to understand and implement

History and Our Results

Bernstein-Blikstad-Saranurak-Tu (FOCS'24): $n^{2+o(1)}$ time by augmenting paths and expander decomposition

History and Our Results

Bernstein-Blikstad-Saranurak-Tu (FOCS'24):
$n^{2+o(1)}$ time by augmenting paths and expander decomposition

This work: n^2 polylog(n) and simpler
- exploits rich combinatorial structure
- even implemented by Blikstad-Tu (2025)

Roadmap

combinatorial structure of max-flow: this talk

- short flow paths --> fast algorithm
- combinatorial preconditioning
- expanders and expander hierarchy

Roadmap

combinatorial structure of max-flow: this talk

- short flow paths --> fast algorithm
- combinatorial preconditioning
- expanders and expander hierarchy

weighted push-relabel: generalization of the push-relabel algorithm for max-flow (skip)

Short Paths --> Fast Algorithm

Theorem: for any h, can compute an s-t flow s.t. (1) "short paths":

average length of a flow path is <= h

Short Paths --> Fast Algorithm

Theorem: for any h, can compute an s-t flow s.t. (1) "short paths":

average length of a flow path is <= h

(2) "no remaining short paths":

in the residual graph, s-t distance > h

Short Paths --> Fast Algorithm

Theorem: for any h, can compute an s-t flow s.t. (1) "short paths":

average length of a flow path is <= h

(2) "no remaining short paths":

in the residual graph, s-t distance > h

Call this a short flow. Can be computed by blocking flow or standard push-relabel

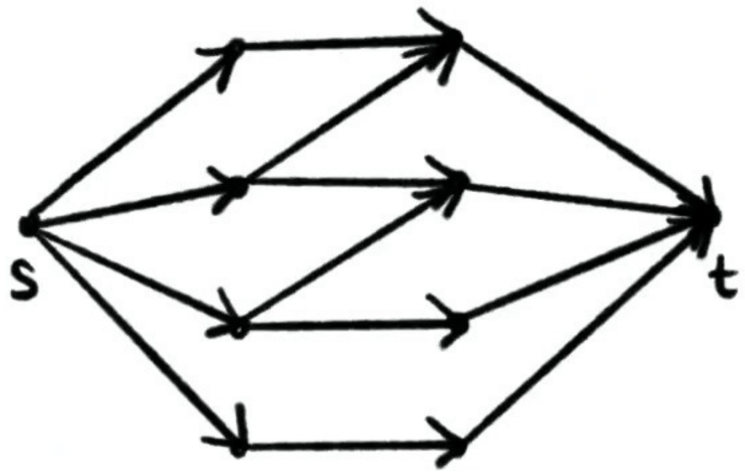
Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

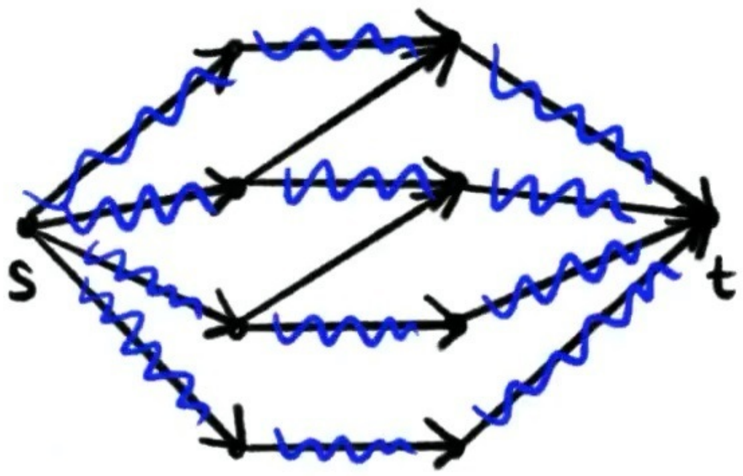
Example: in bipartitite matching instance, flow path length <= 3.



Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

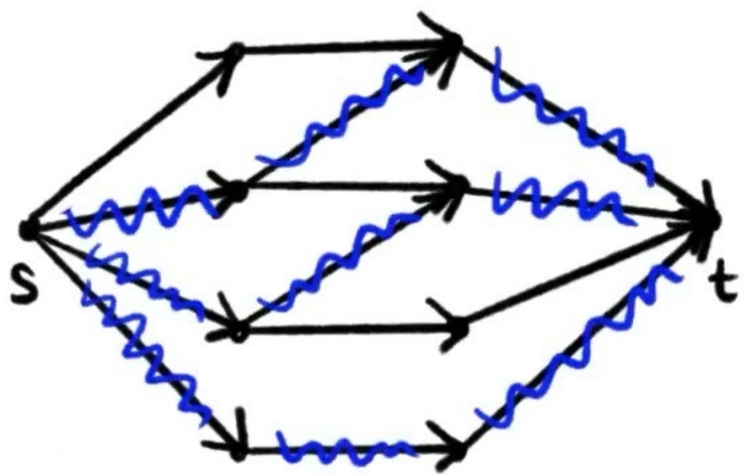
Example: in bipartitite matching instance, flow path length <= 3.



Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

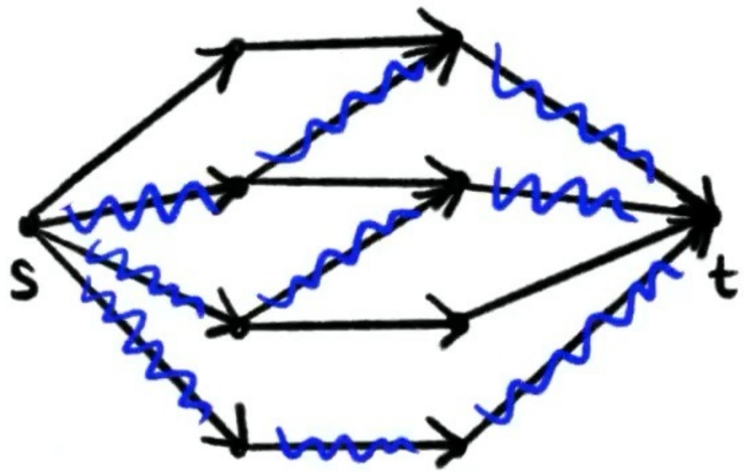
Example: in bipartitite matching instance, flow path length <= 3.



Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

Example: in bipartitite matching instance, flow path length <= 3.



Short flow ~= greedy maximal matching which is 1/2-approx.

Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

Proof: suppose not. Take the residual graph of the short flow, where s-t distance > h. Undo the short flow, then send the max-flow.

Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

Proof: suppose not. Take the residual graph of the short flow, where s-t distance > h. Undo the short flow, then send the max-flow. This flow has value >= 3F/4 and total path length <= F/4*h (undo)

+ F*h/2 (max-flow)

= 3F/4*h.

Short Paths --> Fast Algorithm

Theorem: if max-flow has value F and average flow path length <= h/2, then any h-short flow has value >= F/4.

Proof: suppose not. Take the residual graph of the short flow, where s-t distance > h. Undo the short flow, then send the max-flow. This flow has value >= 3F/4 and total path length <= F/4*h (undo)

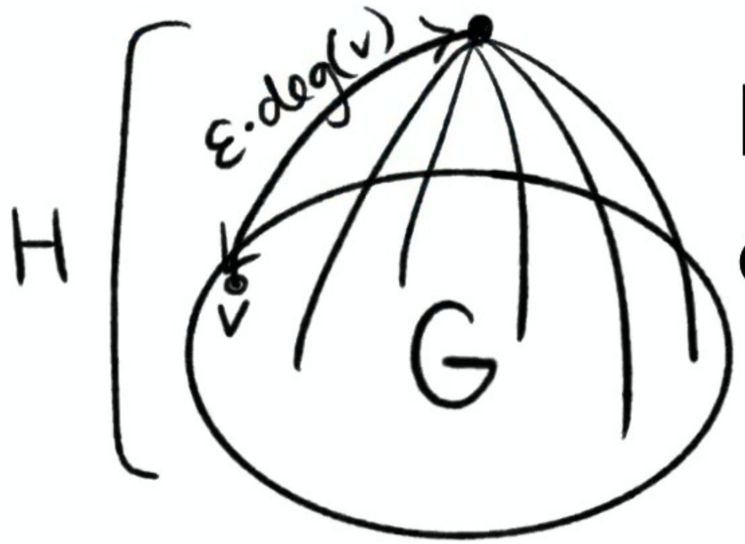
+ F*h/2 (max-flow)

= 3F/4*h.

So average path length <= h on the residual graph =><=

Combinatorial Preconditioning

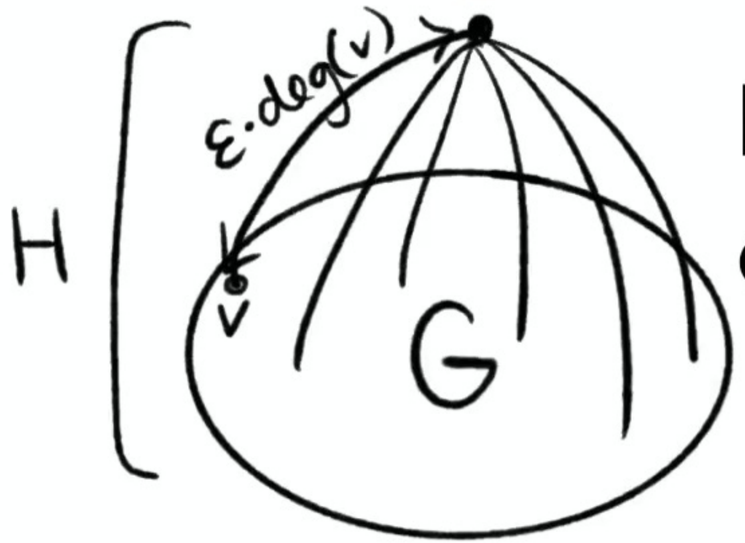
Consider adding a star on graph G:



bidirectional edge to each v
of capacity ε*deg(v)

Combinatorial Preconditioning

Consider adding a star on graph G:



bidirectional edge to each v
of capacity ε*deg(v)

Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= 1/ε

Combinatorial Preconditioning

Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= 1/ε

Combinatorial Preconditioning

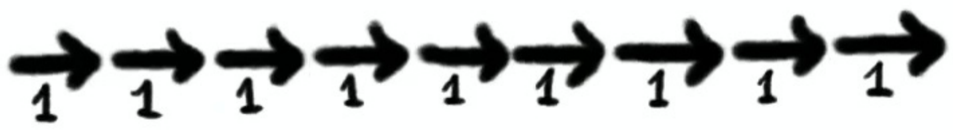
Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= $1/\varepsilon$

Proof: consider max-flow in G. Gradually "leak" the flow to the new vertex.

Combinatorial Preconditioning

Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= $1/\varepsilon$

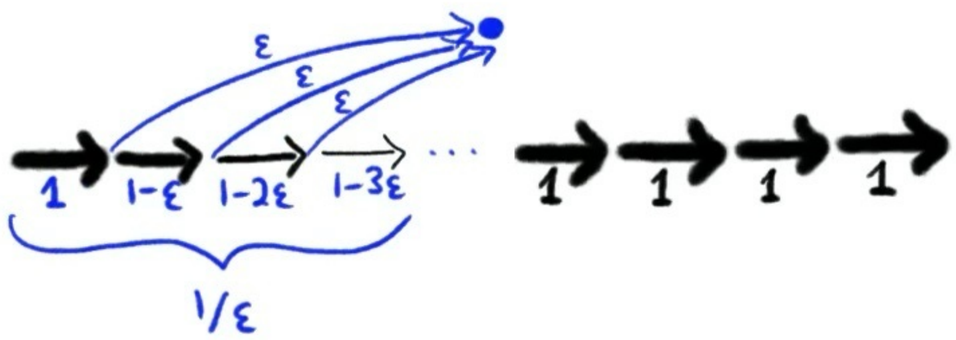
Proof: consider max-flow in G. Gradually "leak" the flow to the new vertex.



Combinatorial Preconditioning

Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= 1/ε

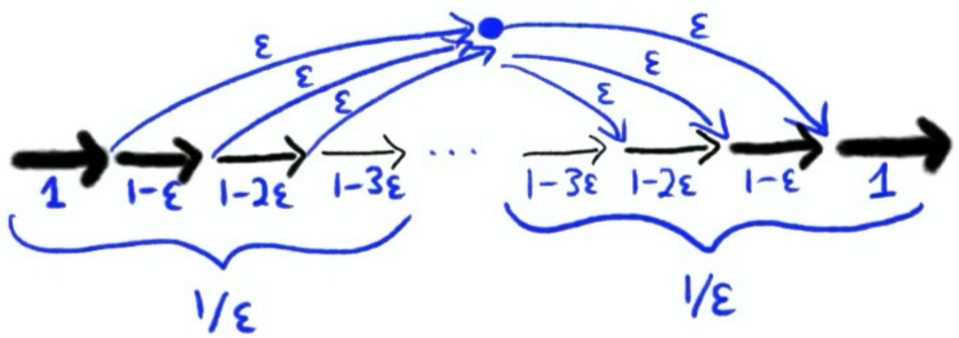
Proof: consider max-flow in G. Gradually "leak" the flow to the new vertex.



Combinatorial Preconditioning

Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= 1/ε

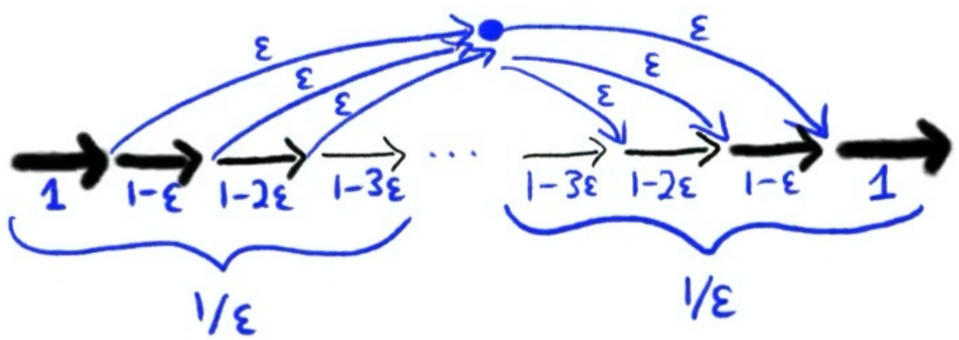
Proof: consider max-flow in G. Gradually "leak" the flow to the new vertex.



Combinatorial Preconditioning

Lemma (preconditioning --> short paths):
if s-t max-flow has value F in G, then there is a flow of value F in H with path length <= 1/ε

Proof: consider max-flow in G. Gradually "leak" the flow to the new vertex.



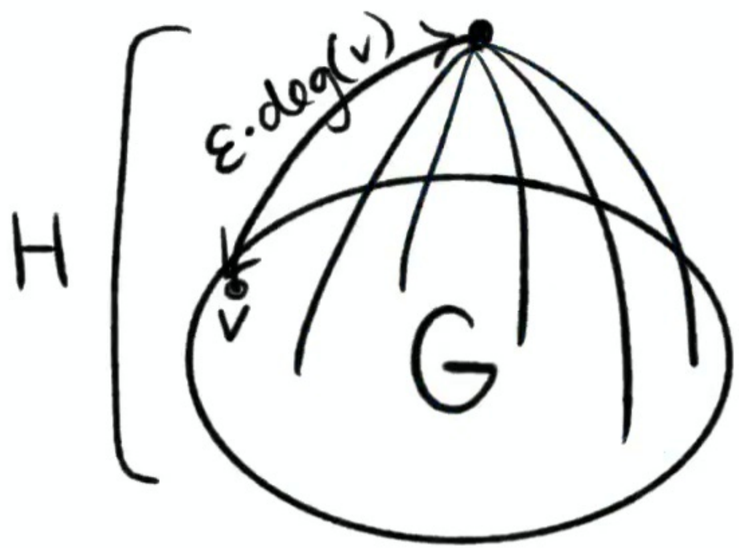
Each vertex participates in <= deg(v) paths, each leaking ε => ε*deg(v) total

Preconditioning on an Expander

Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.

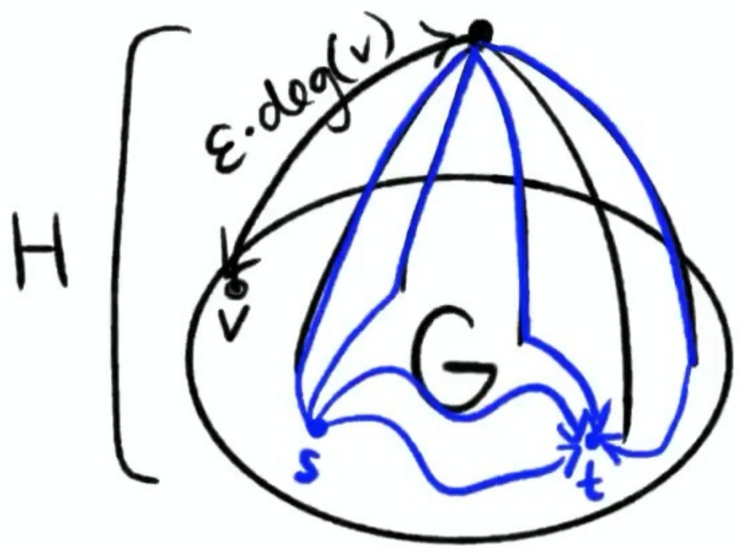
Preconditioning on an Expander

Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.



Preconditioning on an Expander

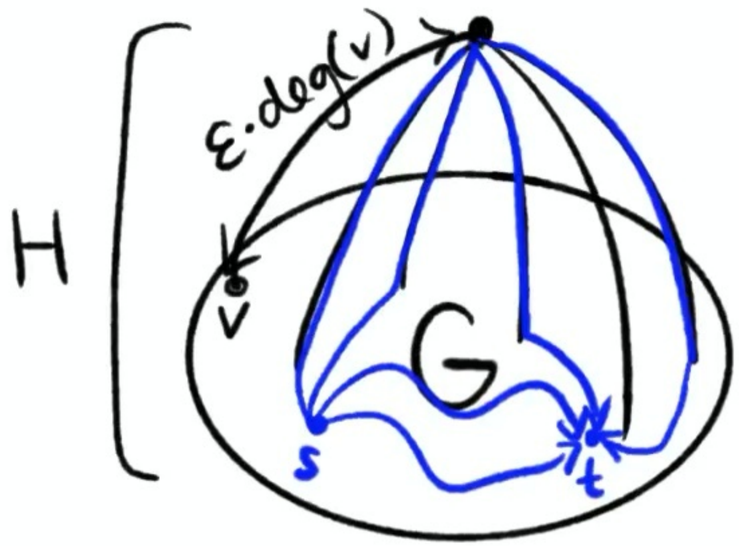
Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.



Preconditioning on an Expander

Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.

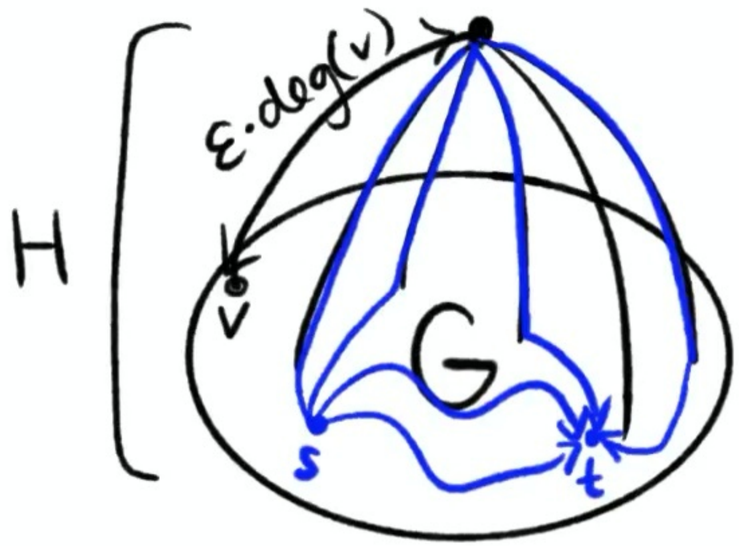
How to convert flow back to G?



Preconditioning on an Expander

Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.

How to convert flow back to G?

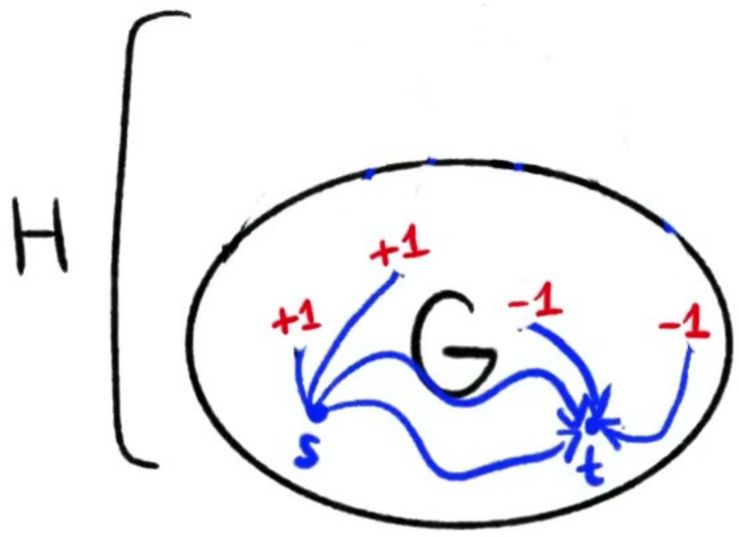


Remove star edges?

Preconditioning on an Expander

Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.

How to convert flow back to G?

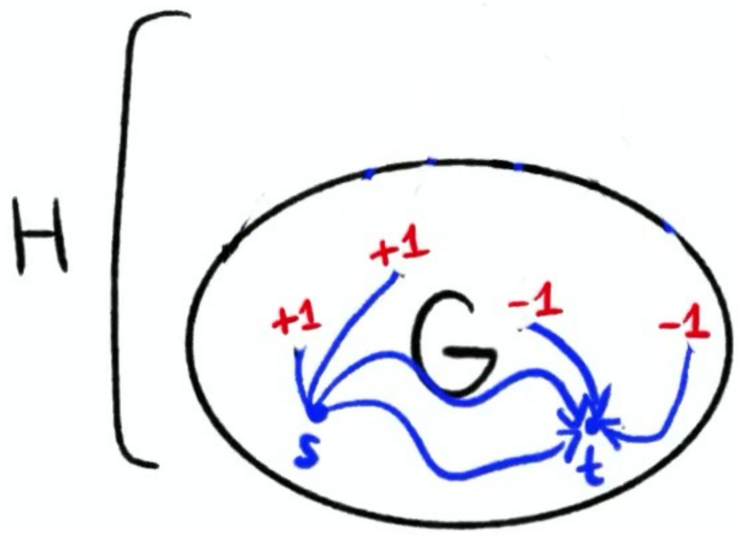


Remove star edges?

Preconditioning on an Expander

Recall that preconditioning => short paths, and short paths algorithm finds flow on H of value >= F/4.

How to convert flow back to G?



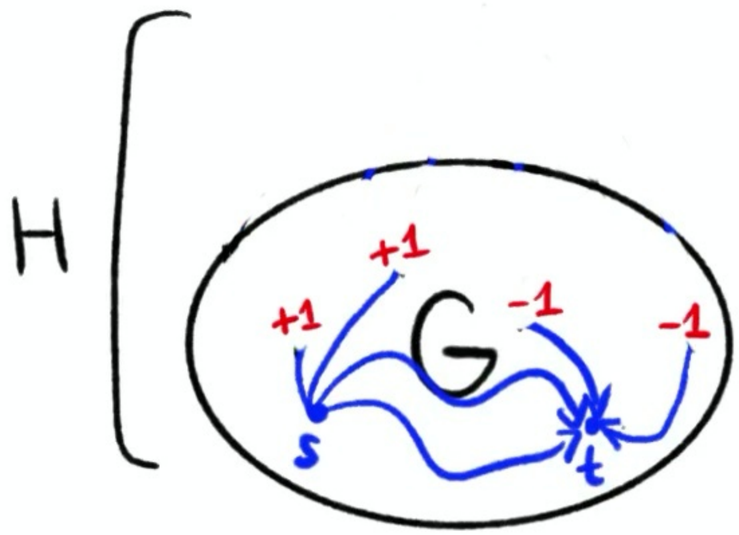
Remove star edges?

Induces vertex demands.

Each vertex demand <= $\varepsilon * \deg(v)$ in absolute value.

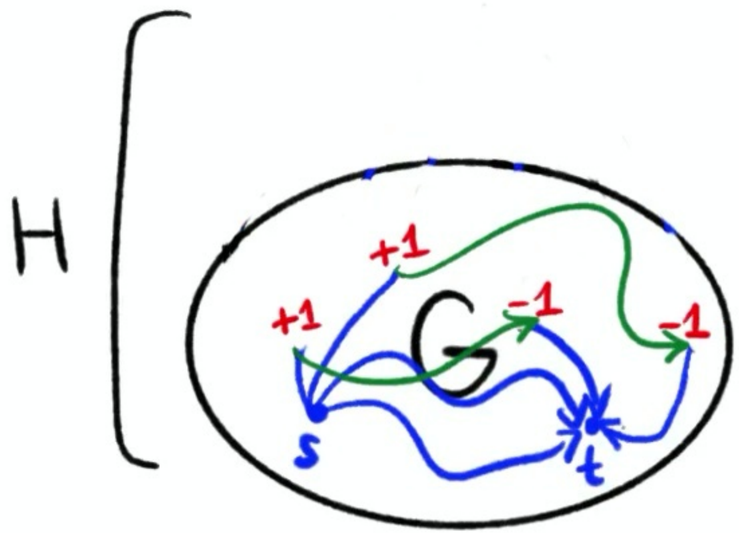
Preconditioning on an Expander

Flow definition of expander: G is a ϕ-expander if for any vertex demand with absolute value <= deg(v) at each vertex v, can be routed with congestion <= 1/ϕ



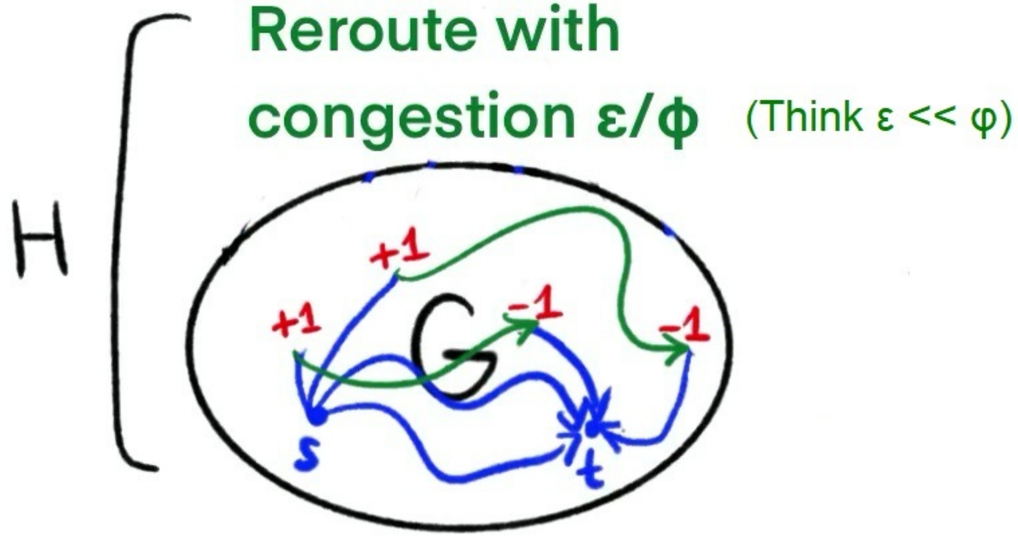
Preconditioning on an Expander

Flow definition of expander: G is a ϕ-expander if for any vertex demand with absolute value <= deg(v) at each vertex v, can be routed with congestion <= 1/ϕ



Preconditioning on an Expander

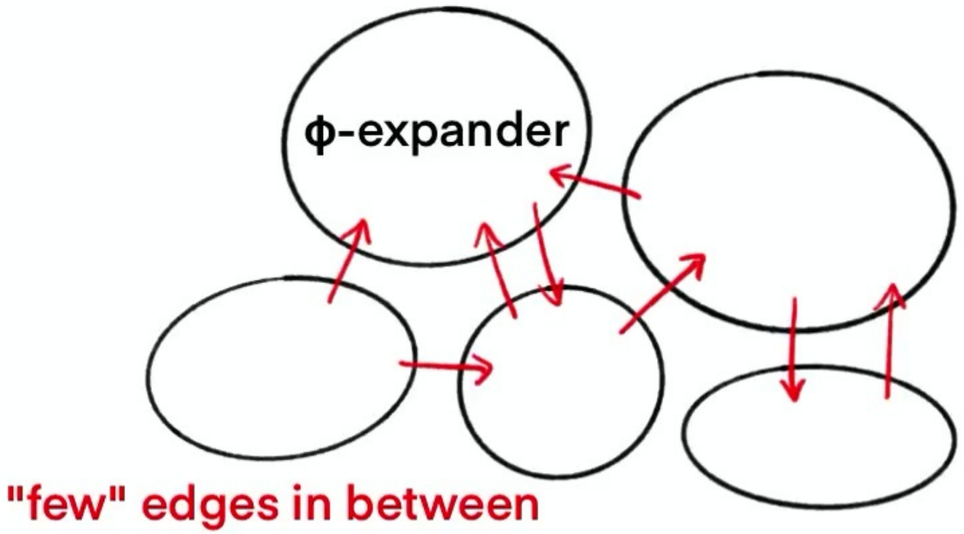
Flow definition of expander: G is a ϕ-expander if for any vertex demand with absolute value <= deg(v) at each vertex v, can be routed with congestion <= 1/ϕ



Expander Decomposition and Hierarchy

In general, G may not be an expander.

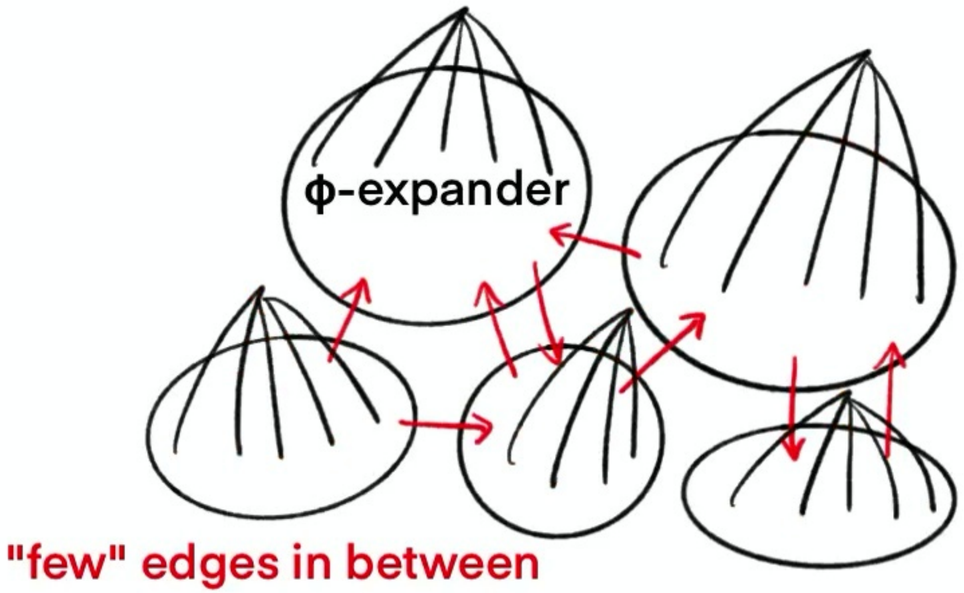
Expander decomposition:



Expander Decomposition and Hierarchy

In general, G may not be an expander.

Expander decomposition:

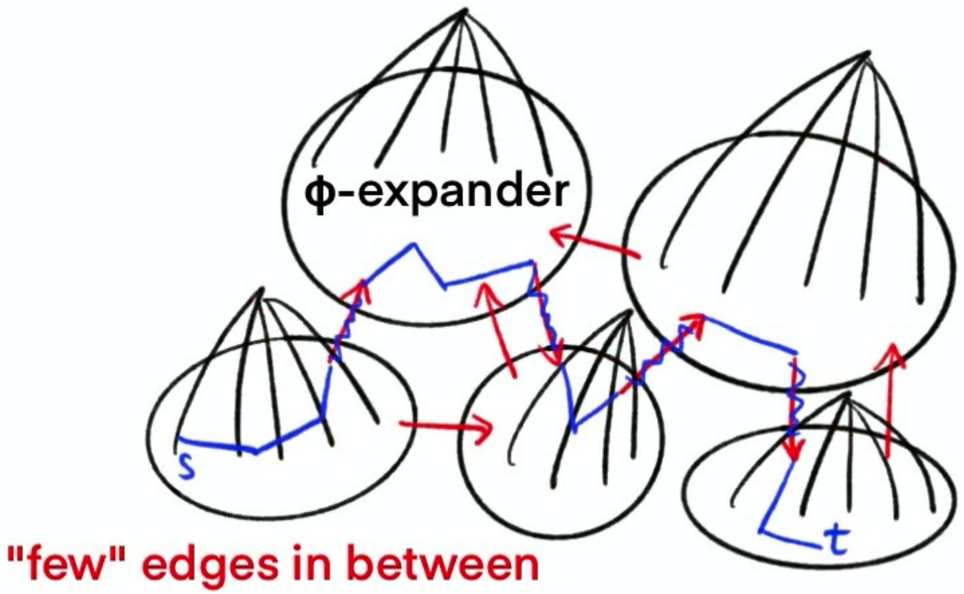


Precondition each expander. Can assume flow paths use few edges from each expander.

Expander Decomposition and Hierarchy

In general, G may not be an expander.

Expander decomposition:

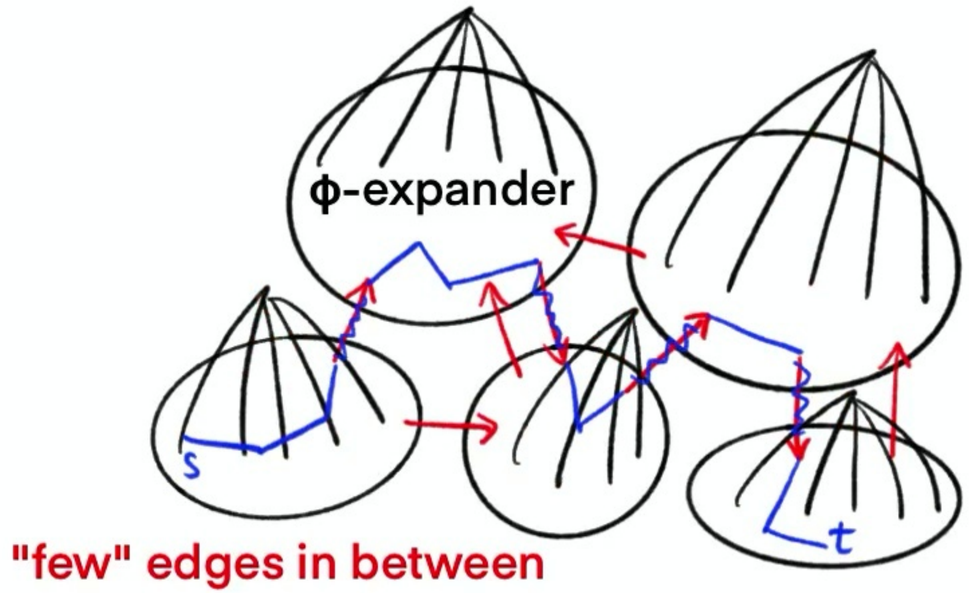


Precondition each expander. Can assume flow paths use few edges from each expander.

Expander Decomposition and Hierarchy

In general, G may not be an expander.

Expander decomposition:



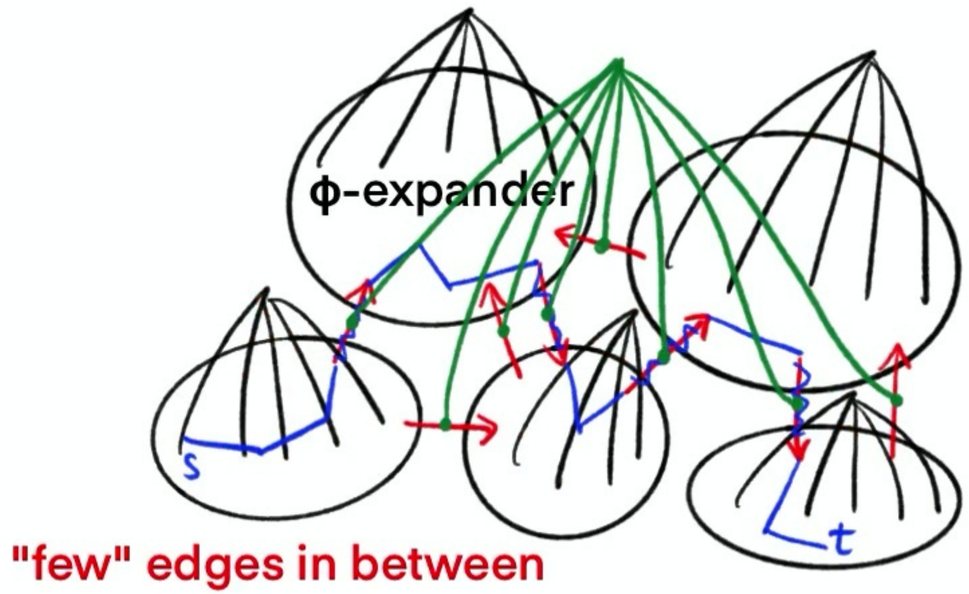
Precondition each expander. Can assume flow paths use few edges from each expander.

What if it uses too many edges in between?

Expander Decomposition and Hierarchy

In general, G may not be an expander.

Expander decomposition:



Precondition each expander. Can assume flow paths use few edges from each expander.

Add another star. If these edges "expand", then done. Otherwise, next-level decomposition.

Directed Acyclic Graphs (DAGs)

Directed Acyclic Graphs (DAGs)

A DAG cannot be decomposed into expanders: any subgraph is still a DAG and hence not an expander.

Directed Acyclic Graphs (DAGs)

A DAG cannot be decomposed into expanders: any subgraph is still a DAG and hence not an expander.

Treat DAGs as a base case handled by weighted push-relabel.

Directed Acyclic Graphs (DAGs)

A DAG cannot be decomposed into expanders: any subgraph is still a DAG and hence not an expander.

Treat DAGs as a base case handled by weighted push-relabel.

DAG instances are quite general, includes bipartite matching.

Missing Details

Constructing the expander hierarchy

- itself uses max-flow calls

- use lower levels of hierarchy for these flows

Missing Details

Constructing the expander hierarchy

- itself uses max-flow calls
- use lower levels of hierarchy for these flows

Constant-approximation is sufficient

- re-run algorithm on residual graph to halve the remaining flow
- repeat O(log n) times for 1/poly(n) error

Conclusion

Combinatorial structure for max-flow: short paths, star preconditioning, expanders and DAGs

Conclusion

Combinatorial structure for max-flow: short paths, star preconditioning, expanders and DAGs

Open: combinatorial beyond n^2? (And Goldberg-Rao's $m^{1.5}$)
- already hard on DAGs!