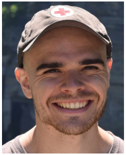
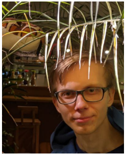
Low-Step Multi-Commodity Flow Emulators

Bernhard[1,2] Haeupler



D Ellis[3] Hershkowitz

Jason[4] Li



Antti[2] Roeyskoe

Thatchaphol[5] Saranurak

1: INSAIT, 2: ETH Zürich, 3: Brown University, 4: CMU, 5: U of Michigan

Multi-Commodity Flow Problem

Multi-Commodity Flow Problem

Input:

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Output:

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Output:

- $F = \sum_{(a,b)\in \mathrm{supp}(D)} F_{a,b}$

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Output:

- $F = \sum_{(a,b) \in \mathrm{supp}(D)} F_{a,b}$
- F must be *capacity-respecting*

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Output:

- $F = \sum_{(a,b)\in\text{supp}(D)} F_{a,b}$
- F must be *capacity-respecting*
- Variants:

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Output:

- $F = \sum_{(a,b)\in \mathrm{supp}(D)} F_{a,b}$
- F must be *capacity-respecting*
- Variants:

$$
\begin{aligned}
|F_{a,b}| &= \lambda D(a, b) && \text{(Concurrent)} \\
|F_{a,b}| &\leq D(a, b) && \text{(Non-Concurrent)}
\end{aligned}
$$

Multi-Commodity Flow Problem

Input:

- Undirected graph G with edge capacities u
- Demand D: "send $D(a, b)$ between $a, b \in V$"

Output:

- $F = \sum_{(a,b) \in \mathrm{supp}(D)} F_{a,b}$
- F must be *capacity-respecting*
- Variants:

$$|F_{a,b}| = \lambda D(a, b) \qquad \text{(Concurrent)}$$
$$|F_{a,b}| \leq D(a, b) \qquad \text{(Non-Concurrent)}$$

- Goal: maximize $|F|$

MCF Algorithms

$$n = |V| \qquad m = |E| \qquad k = |\mathrm{supp}(D)|$$

MCF Algorithms

$$n = |V| \quad m = |E| \quad k = |\mathrm{supp}(D)|$$

	Approx.	Time	C/NC
[Mad10]	$(1+\delta)$	$\mathcal{O}(mn/\delta)$	Both

MCF Algorithms

$$n = |V| \quad m = |E| \quad k = |\mathrm{supp}(D)|$$

	Approx.	Time	C/NC
[Mad10]	$(1+\delta)$	$\mathcal{O}(mn/\delta)$	Both
[KMP12]	$(1+\delta)$	$\mathcal{O}(m^{4/3}\mathrm{poly}(k/\delta))$	NC
[She17]	$(1+\delta)$	$\mathcal{O}(mk/\delta)$	C

Flow-Decomposition Barrier

Recall: the output is k single-commodity flows

Flow-Decomposition Barrier

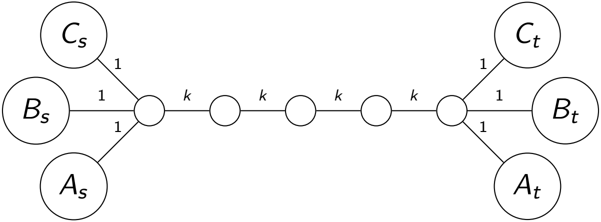
Recall: the output is k single-commodity flows

$\rightarrow$ $\Omega(mk)$ lower bound?

Flow-Decomposition Barrier

Recall: the output is k single-commodity flows

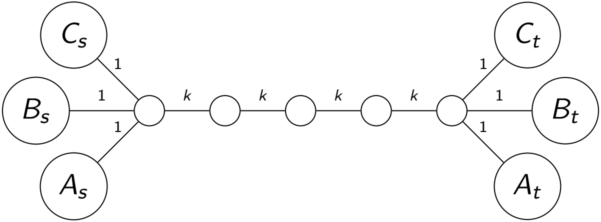
$\rightarrow$ $\Omega(mk)$ lower bound?



Flow-Decomposition Barrier

Recall: the output is k single-commodity flows

$\rightarrow$ $\Omega(mk)$ lower bound?

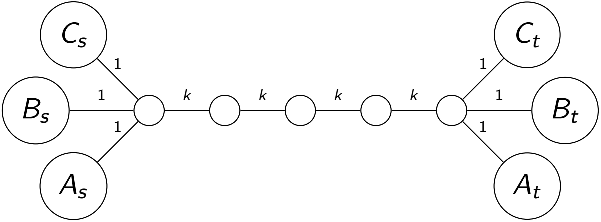


Solutions?

Flow-Decomposition Barrier

Recall: the output is k single-commodity flows

$\rightarrow$ $\Omega(mk)$ lower bound?



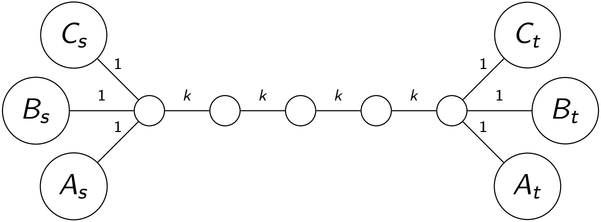
Solutions?

- Assume unit-capacity? (non-concurrent only)

Flow-Decomposition Barrier

Recall: the output is k single-commodity flows

$\rightarrow$ $\Omega(mk)$ lower bound?



Solutions?

- Assume unit-capacity? (non-concurrent only)
- Output *value only*?

MCF Algorithms

$$n = |V| \quad m = |E| \quad k = |\mathrm{supp}(D)|$$

	Approx.	Time	C/NC	Output
[Mad10]	$(1+\delta)$	$\Omega(mn)$	Both	Explicit
[KMP12]	$(1+\delta)$	$\Omega(mk)$	NC	Explicit
[She17]	$(1+\delta)$	$\Omega(mk)$	C	Explicit

MCF Algorithms

$$n = |V| \qquad m = |E| \qquad k = |\mathrm{supp}(D)|$$

	Approx.	Time	C/NC	Output
[Mad10]	$(1+\delta)$	$\Omega(mn)$	Both	Explicit
[KMP12]	$(1+\delta)$	$\Omega(mk)$	NC	Explicit
[She17]	$(1+\delta)$	$\Omega(mk)$	C	Explicit
[RST14]	$\log^4(n)$	$(m+k)\tilde{O}(1)$	Both	Value

MCF Algorithms

$$n = |V| \quad m = |E| \quad k = |\mathrm{supp}(D)|$$

	Approx.	Time	C/NC	Output
[Mad10]	$(1+\delta)$	$\Omega(mn)$	Both	Explicit
[KMP12]	$(1+\delta)$	$\Omega(mk)$	NC	Explicit
[She17]	$(1+\delta)$	$\Omega(mk)$	C	Explicit
[RST14]	$\log^4(n)$	$(m+k)\tilde{O}(1)$	Both	Value
[Chu21]	$(\log n)^{\mathcal{O}_\epsilon(1)}$	$(m+k)n^\epsilon$	NC†	Explicit
[CZ23]	$(\log \log n)^{\mathcal{O}_\epsilon(1)}$	$(m+k)n^\epsilon$	NC†	Explicit

†: Unit-capacity graphs only

MCF Algorithms

$$n = |V| \quad m = |E| \quad k = |\mathrm{supp}(D)|$$

	Approx.	Time	C/NC	Output
[Mad10]	$(1+\delta)$	$\Omega(mn)$	Both	Explicit
[KMP12]	$(1+\delta)$	$\Omega(mk)$	NC	Explicit
[She17]	$(1+\delta)$	$\Omega(mk)$	C	Explicit
[RST14]	$\log^4(n)$	$(m+k)\tilde{O}(1)$	Both	Value
[Chu21]	$(\log n)^{\mathcal{O}_\epsilon(1)}$	$(m+k)n^\epsilon$	NC†	Explicit
[CZ23]	$(\log\log n)^{\mathcal{O}_\epsilon(1)}$	$(m+k)n^\epsilon$	NC†	Explicit
Ours	$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

†: Unit-capacity graphs only

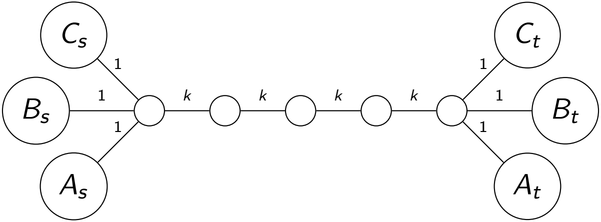
MCF Algorithms

$$n = |V| \quad m = |E| \quad k = |\mathrm{supp}(D)|$$

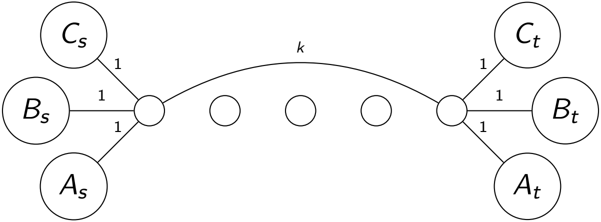
	Approx.	Time	C/NC	Output
[Mad10]	$(1+\delta)$	$\Omega(mn)$	Both	Explicit
[KMP12]	$(1+\delta)$	$\Omega(mk)$	NC	Explicit
[She17]	$(1+\delta)$	$\Omega(mk)$	C	Explicit
[RST14]	$\log^4(n)$	$(m+k)\tilde{O}(1)$	Both	Value
[Chu21]	$(\log n)^{\mathcal{O}_\epsilon(1)}$	$(m+k)n^\epsilon$	NC†	Explicit
[CZ23]	$(\log\log n)^{\mathcal{O}_\epsilon(1)}$	$(m+k)n^\epsilon$	NC†	Explicit
Ours	$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	**Implicit**

†: Unit-capacity graphs only

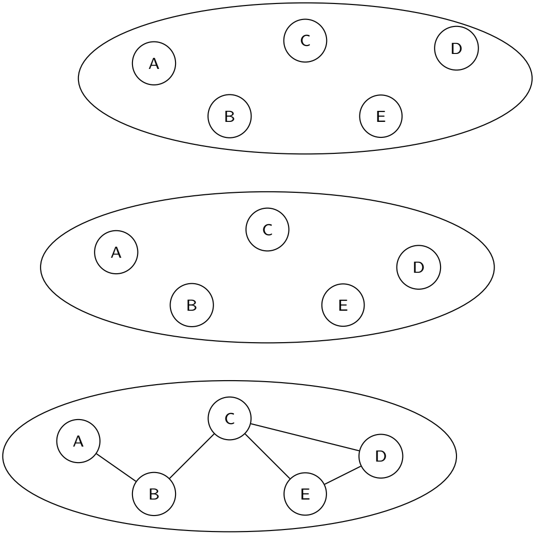
Implicit Representation



Implicit Representation

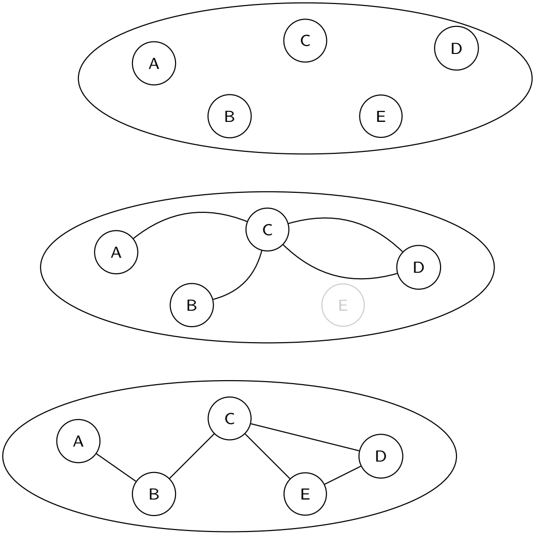


Implicit Representation



Implicit Representation

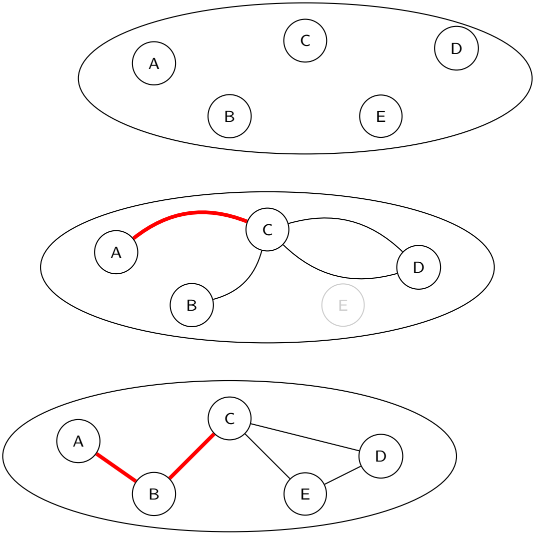
Stacked copies of vertex set on top of original graph



Implicit Representation

Stacked copies of vertex set on top of original graph

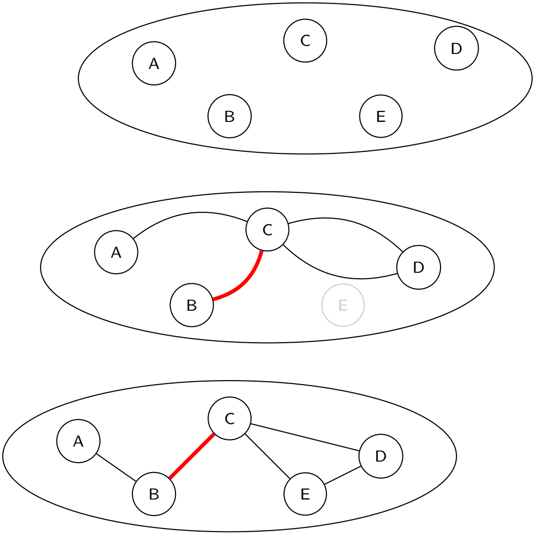
Edges correspond to *short* paths on previous level



Implicit Representation

Stacked copies of vertex set on top of original graph

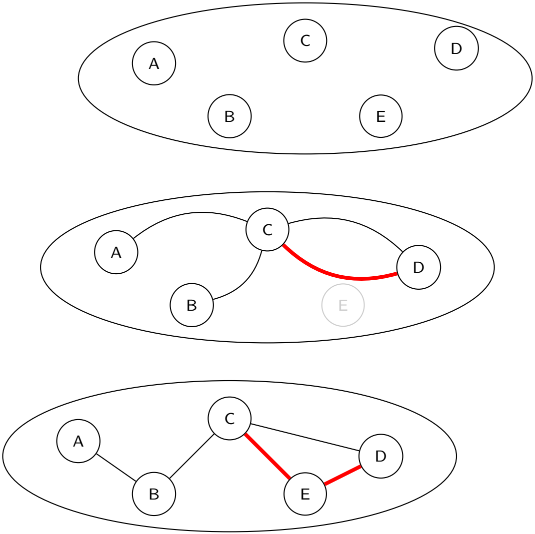
Edges correspond to *short* paths on previous level



Implicit Representation

Stacked copies of vertex set on top of original graph

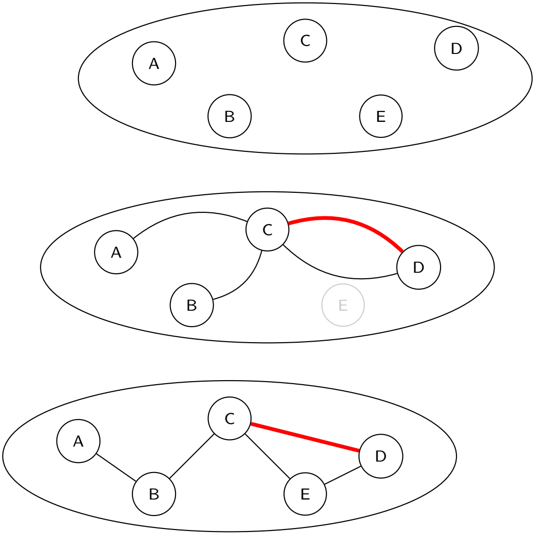
Edges correspond to *short* paths on previous level



Implicit Representation

Stacked copies of vertex set on top of original graph

Edges correspond to *short* paths on previous level



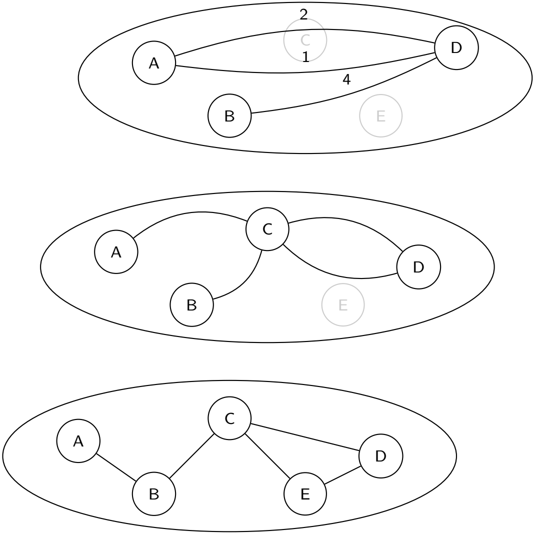
Implicit Representation

Stacked copies of vertex set on top of original graph

Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value



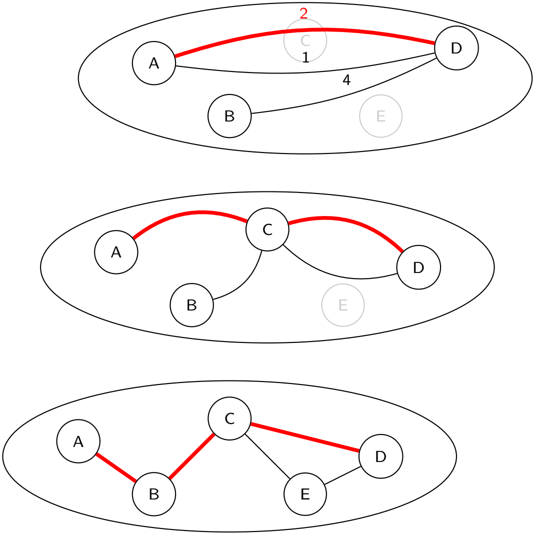
Implicit Representation

Stacked copies of vertex set on top of original graph

Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value



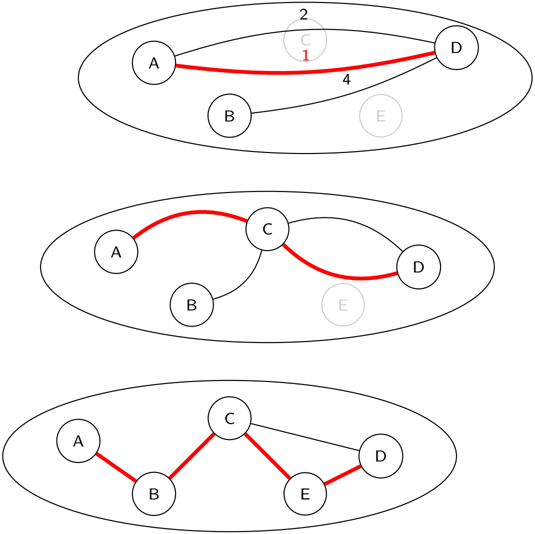
Implicit Representation

Stacked copies of vertex set on top of original graph

Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value



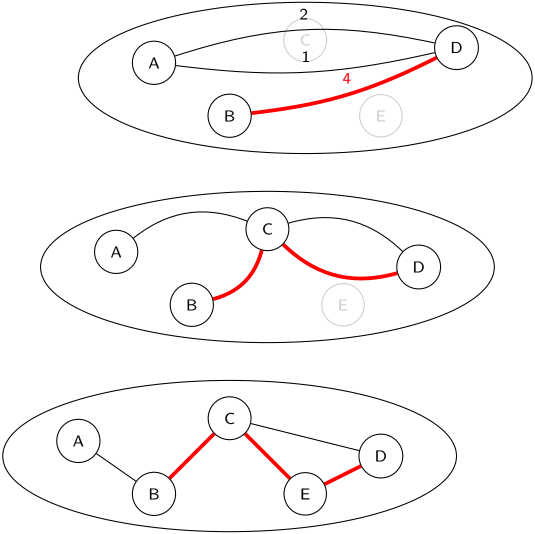
Implicit Representation

Stacked copies of vertex set on top of original graph

Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value



Implicit Representation

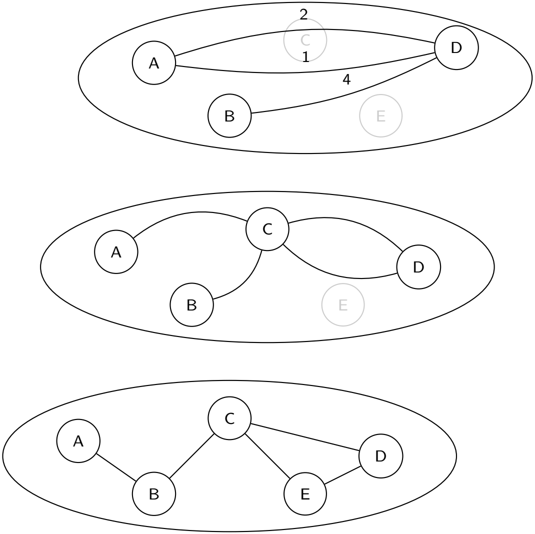
Stacked copies of vertex set on top of original graph

Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value

Can compute



Implicit Representation

Stacked copies of vertex set on top of original graph

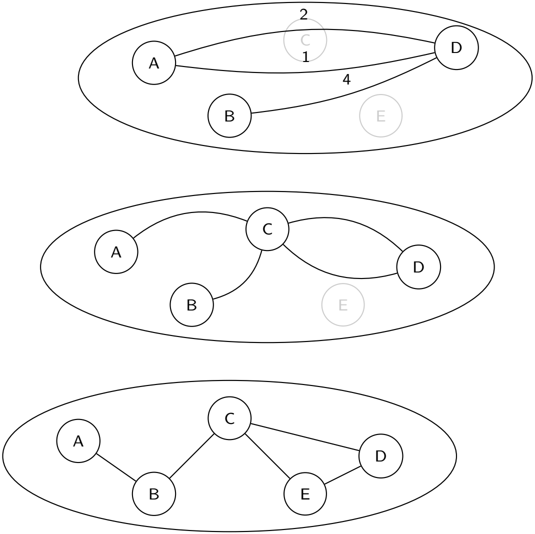
Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value

Can compute

- Flow value



Implicit Representation

Stacked copies of vertex set on top of original graph

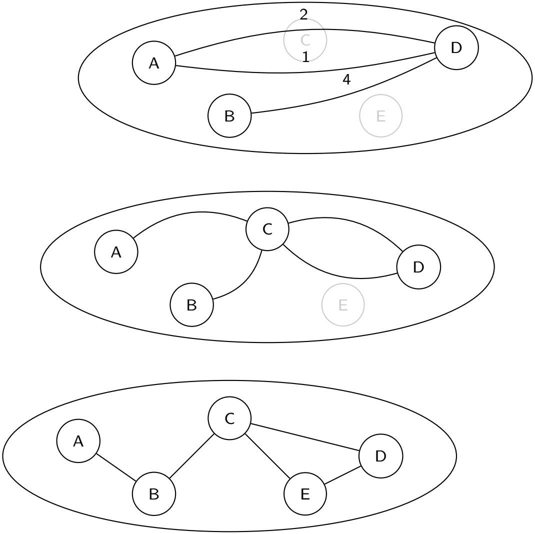
Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value

Can compute

- Flow value
- Congestion



Implicit Representation

Stacked copies of vertex set on top of original graph

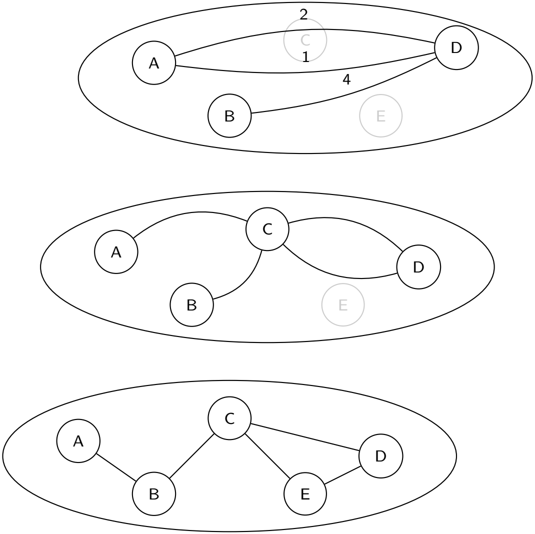
Edges correspond to *short* paths on previous level

Topmost layer defines flow

- Edge = flow path
- Capacity = flow value

Can compute

- Flow value
- Congestion
- i^{th} edge of j^{th} path



Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

- Near-linear time algorithm

Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

- Near-linear time algorithm
- Finds implicit representation of $\mathcal{O}_\epsilon(1)$-competitive flow

Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

- Near-linear time algorithm
- Finds implicit representation of $\mathcal{O}_\epsilon(1)$-competitive flow
  - Even *existence* of $o(mk)$-size, $\mathcal{O}(1)$-competitive implicit representations was open

Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

- Near-linear time algorithm
- Finds implicit representation of $\mathcal{O}_\epsilon(1)$-competitive flow
  - Even *existence* of $o(mk)$-size, $\mathcal{O}(1)$-competitive implicit representations was open
- Generalizes to min-cost multi-commodity flow

Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

- Near-linear time algorithm
- Finds implicit representation of $\mathcal{O}_\epsilon(1)$-competitive flow
  - Even *existence* of $o(mk)$-size, $\mathcal{O}(1)$-competitive implicit representations was open
- Generalizes to min-cost multi-commodity flow
- Works in parallel, with depth n^ϵ

Our Results

Approx.	Time	C/NC	Output
$\mathcal{O}_\epsilon(1)$	$(m+k)n^\epsilon$	Both	Implicit

- Near-linear time algorithm
- Finds implicit representation of $\mathcal{O}_\epsilon(1)$-competitive flow
  - Even *existence* of $o(mk)$-size, $\mathcal{O}(1)$-competitive implicit representations was open
- Generalizes to min-cost multi-commodity flow
- Works in parallel, with depth n^ϵ
- Key ingredient: **low-step MCMCF emulators**

Outline

So far:

- Problem definition
- Our results

Outline

So far:

- Problem definition
- Our results

Next:

Outline

So far:

- Problem definition
- Our results

Next:

- Setting with lengths

Outline

So far:

- Problem definition
- Our results

Next:

- Setting with lengths
- Emulators: definition and results

Outline

So far:

- Problem definition
- Our results

Next:

- Setting with lengths
- Emulators: definition and results
- Brief overview of algorithm

Setting with Lengths

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Flows: assign flow to *paths* in G

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Flows: assign flow to *paths* in G

- $\mathrm{cong}(F) := \max_e F(e)/u(e) := \max_e \sum_{P:e\in P} F(P)/u(e)$

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Flows: assign flow to *paths* in G

- $\mathrm{cong}(F) := \max_e F(e)/u(e) := \max_e \sum_{P: e \in P} F(P)/u(e)$
- $\mathrm{totlen}(F) := \sum_P F(P) \cdot \mathrm{len}(P)$

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Flows: assign flow to *paths* in G

- $\mathrm{cong}(F) := \max_e F(e)/u(e) := \max_e \sum_{P:e\in P} F(P)/u(e)$
- $\mathrm{totlen}(F) := \sum_P F(P) \cdot \mathrm{len}(P)$
- $\mathrm{step}(F) := \max_{P\in \mathrm{supp}(F)} |P|$

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Flows: assign flow to *paths* in G

- $\mathrm{cong}(F) := \max_e F(e)/u(e) := \max_e \sum_{P:e\in P} F(P)/u(e)$
- $\mathrm{totlen}(F) := \sum_P F(P) \cdot \mathrm{len}(P)$
- $\mathrm{step}(F) := \max_{P\in \mathrm{supp}(F)} |P|$

Flow F *routes* demand D if

Setting with Lengths

Graphs: undirected, with capacities $u(e)$ and *lengths* $l(e)$

Flows: assign flow to *paths* in G

- $\mathrm{cong}(F) := \max_e F(e)/u(e) := \max_e \sum_{P:e\in P} F(P)/u(e)$
- $\mathrm{totlen}(F) := \sum_P F(P) \cdot \mathrm{len}(P)$
- $\mathrm{step}(F) := \max_{P\in\mathrm{supp}(F)} |P|$

Flow F *routes* demand D if $\sum_{(a,b)\text{-path } P} F(P) = D(a,b)$

Low-Step Flow Emulators (Informal)

Low-Step Flow Emulators (Informal)

A graph simplification that

Low-Step Flow Emulators (Informal)

A graph simplification that

- Preserves min-cost multi-commodity flows, and

Low-Step Flow Emulators (Informal)

A graph simplification that

- Preserves min-cost multi-commodity flows, and
- Allows us to focus only on flows with a *short step-length*

Low-Step Flow Emulators (Informal)

A graph simplification that

- Preserves min-cost multi-commodity flows, and
- Allows us to focus only on flows with a *short step-length*
    - i.e., all flow paths have few edges

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\text{cong}(F') \leq \kappa \cdot \text{cong}(F^*)$

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\mathrm{cong}(F') \leq \kappa \cdot \mathrm{cong}(F^*)$
- $\mathrm{totlen}(F') \leq s \cdot \mathrm{totlen}(F^*)$

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\mathrm{cong}(F') \leq \kappa \cdot \mathrm{cong}(F^*)$
- $\mathrm{totlen}(F') \leq s \cdot \mathrm{totlen}(F^*)$
- $\mathrm{step}(F') \leq t$

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\mathrm{cong}(F') \leq \kappa \cdot \mathrm{cong}(F^*)$
- $\mathrm{totlen}(F') \leq s \cdot \mathrm{totlen}(F^*)$
- $\mathrm{step}(F') \leq t$

$\Leftarrow$: For any F' on H, for $F = \Pi(F')$,

Low-Step Flow Emulators

A *t*-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\mathrm{cong}(F') \leq \kappa \cdot \mathrm{cong}(F^*)$
- $\mathrm{totlen}(F') \leq s \cdot \mathrm{totlen}(F^*)$
- $\mathrm{step}(F') \leq t$

$\Leftarrow$: For any F' on H, for $F = \Pi(F')$,

- $\mathrm{cong}(F) \leq \mathrm{cong}(F')$

Low-Step Flow Emulators

A *t*-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\mathrm{cong}(F') \leq \kappa \cdot \mathrm{cong}(F^*)$
- $\mathrm{totlen}(F') \leq s \cdot \mathrm{totlen}(F^*)$
- $\mathrm{step}(F') \leq t$

$\Leftarrow$: For any F' on H, for $F = \Pi(F')$,

- $\mathrm{cong}(F) \leq \mathrm{cong}(F')$
- $\mathrm{totlen}(F) \leq \mathrm{totlen}(F')$

Low-Step Flow Emulators

A t-**step emulator** of a graph G is a

- graph H on same vertex set
- with mapping Π from *edges* in H to *paths* in G,

Such that

$\Rightarrow$: For every flow F^* routing D in G, exists F' routing D in H, s.t.

- $\mathrm{cong}(F') \leq \kappa \cdot \mathrm{cong}(F^*)$
- $\mathrm{totlen}(F') \leq s \cdot \mathrm{totlen}(F^*)$
- $\mathrm{step}(F') \leq t$

$\Leftarrow$: For any F' on H, for $F = \Pi(F')$,

- $\mathrm{cong}(F) \leq \mathrm{cong}(F')$
- $\mathrm{totlen}(F) \leq \mathrm{totlen}(F')$

Parameters: congestion slack κ, length slack s

Emulators: Our Results

Emulators: Our Results

Emulators (Constructive)

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

- step bound $t = \mathcal{O}_{\epsilon}(1)$,

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

- step bound $t = \mathcal{O}_\epsilon(1)$,
- length slack $s = \mathcal{O}_\epsilon(1)$, and

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

- step bound $t = \mathcal{O}_\epsilon(1)$,
- length slack $s = \mathcal{O}_\epsilon(1)$, and
- congestion slack $\kappa = n^\epsilon$

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

- step bound $t = \mathcal{O}_\epsilon(1)$,
- length slack $s = \mathcal{O}_\epsilon(1)$, and
- congestion slack $\kappa = n^\epsilon$

can be computed in time $|E(G)|^{1+\epsilon}$

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

- step bound $t = \mathcal{O}_\epsilon(1)$,
- length slack $s = \mathcal{O}_\epsilon(1)$, and
- congestion slack $\kappa = n^\epsilon$

can be computed in time $|E(G)|^{1+\epsilon}$

- The mapping Π is *implicit*

Emulators: Our Results

Emulators (Constructive)

For any $\epsilon > 0$ and G, an emulator H with

- step bound $t = \mathcal{O}_\epsilon(1)$,
- length slack $s = \mathcal{O}_\epsilon(1)$, and
- congestion slack $\kappa = n^\epsilon$

can be computed in time $|E(G)|^{1+\epsilon}$

- The mapping Π is *implicit*
- We also give an existential result with a tighter tradeoff

The Flow Algorithm

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
    - Recall: want constant approximation

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

Solutions:

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

Solutions:

1. Boosting: (κ, s)-approx to $\mathcal{O}(s)$-approx

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

Solutions:

1. Boosting: (κ, s)-approx to $\mathcal{O}(s)$-approx
   - Recall: $s = \mathcal{O}_{\epsilon}(1)$

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

Solutions:

1. Boosting: (κ, s)-approx to $\mathcal{O}(s)$-approx
   - Recall: $s = \mathcal{O}_{\epsilon}(1)$
   - Similar to [GK98]; enabled by implicit representation!

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

Solutions:

1. Boosting: (κ, s)-approx to $\mathcal{O}(s)$-approx
   - Recall: $s = \mathcal{O}_{\epsilon}(1)$
   - Similar to [GK98]; enabled by implicit representation!
2. Novel high-commodity short flow algorithm

The Flow Algorithm

Emulators reduce MCMC-flow problems into *short* MCMCF-problems, but

1. "Congestion slack $\kappa = n^{\epsilon}$"
   - Recall: want constant approximation
2. How to even solve a *short* MCMC-flow problem?

Solutions:

1. Boosting: (κ, s)-approx to $\mathcal{O}(s)$-approx
   - Recall: $s = \mathcal{O}_{\epsilon}(1)$
   - Similar to [GK98]; enabled by implicit representation!
2. Novel high-commodity short flow algorithm
   - Built on length-constrained expander routing

Length-Constrained Expander Workshop
- **Here at STOC!**

Tuesday 25th - Introduction and overview

- Length-constrained expanders
- **Low-step emulators**

Wednesday 26th - Fast algorithms

- $\mathcal{O}(1)$-**approx min cost multicommodity flow** via emulators
- Algorithmics of LC expander decomposition

Thursday 27th - Dynamic algorithms

- Dynamic emulators $\rightarrow$ $\mathcal{O}(1)$-approx fully dynamic distance oracles!
- Open directions of research

End of Talk

Questions?

References

[Chu21] Julia Chuzhoy. "Decremental all-pairs shortest paths in deterministic near-linear time". In: *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM, 2021, pp. 626–639. DOI: 10.1145/3406325.3451025. URL: https://doi.org/10.1145/3406325.3451025.

[CZ23] Julia Chuzhoy and Ruimin Zhang. "A New Deterministic Algorithm for Fully Dynamic All-Pairs Shortest Paths". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*. Ed. by Barna Saha and Rocco A. Servedio. ACM, 2023, pp. 1159–1172. DOI: 10.1145/3564246.3585196. URL: https://doi.org/10.1145/3564246.3585196.

[GK98] Naveen Garg and Jochen Könemann. "Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems". In: *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*. IEEE Computer Society, 1998, pp. 300–309. DOI: 10.1109/SFCS.1998.743463. URL: https://doi.org/10.1109/SFCS.1998.743463.

[KMP12] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. "Faster approximate multicommodity flow using quadratically coupled flows". In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*. Ed. by Howard J. Karloff and Toniann Pitassi. ACM, 2012, pp. 1–18. DOI: 10.1145/2213977.2213979. URL: https://doi.org/10.1145/2213977.2213979.

[Mad10] Aleksander Madry. "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 121–130. DOI: 10.1145/1806689.1806708. URL: https://doi.org/10.1145/1806689.1806708.

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. "Computing Cut-Based Hierarchical Decompositions in Almost Linear Time". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. Ed. by Chandra Chekuri. SIAM, 2014, pp. 227–238. DOI: 10.1137/1.9781611973402.17. URL: https://doi.org/10.1137/1.9781611973402.17.

[She17] Jonah Sherman. "Area-convexity, l_∞ regularization, and undirected multicommodity flow". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM, 2017, pp. 452–460. DOI: 10.1145/3055399.3055501. URL: https://doi.org/10.1145/3055399.3055501.