Universally-Optimal $(1+\varepsilon)$-Approximate Shortest Path and Transshipment in the Distributed Setting

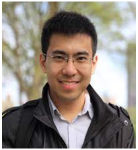
Goran Zuzic

ETH Zurich

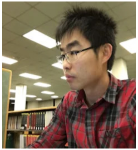
21 Oct 2021

Gramoz Goranci

Bernhard Haeupler



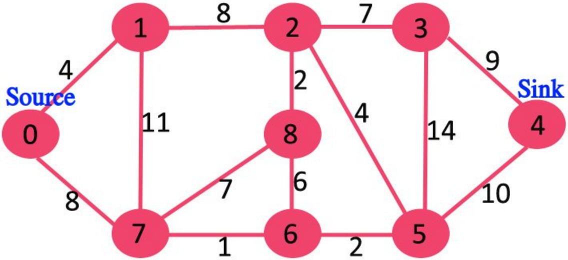
Xiaorui Sun



Mingquan Ye

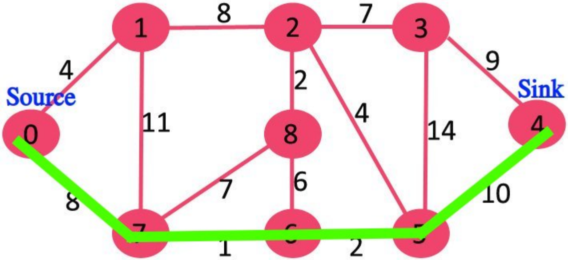
We want to solve the **single-source shortest path** problem (**SSSP**).

- Given a n-vertex undirected graph where edges have weights in the set $\{1, 2, \ldots, n^{O(1)}\}$. Compute shortest path from source to all other nodes.



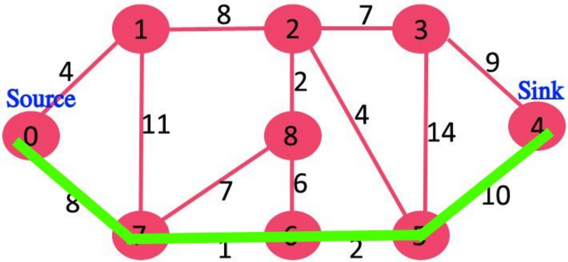
We want to solve the **single-source shortest path** problem (**SSSP**).

- Given a n-vertex undirected graph where edges have weights in the set $\{1, 2, \ldots, n^{O(1)}\}$. Compute shortest path from source to all other nodes.



We want to solve the **single-source shortest path** problem (**SSSP**).

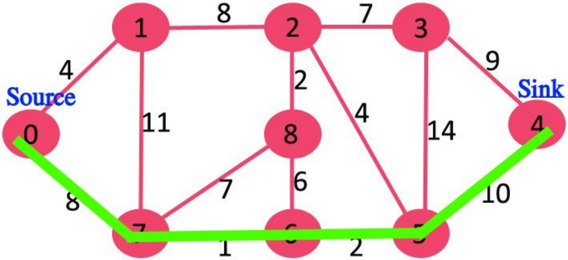
- Given a n-vertex undirected graph where edges have weights in the set $\{1, 2, \ldots, n^{O(1)}\}$. Compute shortest path from source to all other nodes.



One of the oldest problems in computer science.
Sequential setting .. easy! Dijkstra's famous $\tilde{O}(m+n)$-time algorithm is optimal.

We want to solve the **single-source shortest path** problem (**SSSP**).

- Given a n-vertex undirected graph where edges have weights in the set $\{1, 2, \ldots, n^{O(1)}\}$. Compute shortest path from source to all other nodes.



One of the oldest problems in computer science.
Sequential setting .. easy! Dijkstra's famous $\tilde{O}(m+n)$-time algorithm is optimal.

What about parallel or distributed settings? The problem seems much harder, processing long paths that fork and merge seems inherently sequential at first glance.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

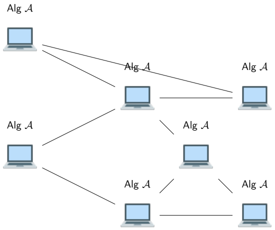


Figure: Network G.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

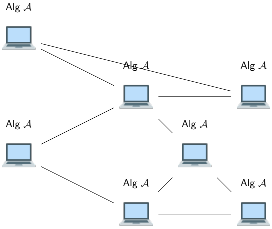


Figure: Network G.

- Communication in synchronous rounds. Local computation inbetween.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

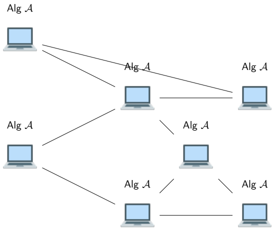


Figure: Network G.

- Communication in synchronous rounds. Local computation inbetween.
- Each round neighbors exchange $\tilde{O}(1)$-bit msgs.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

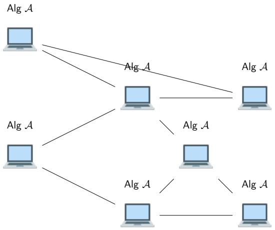


Figure: Network G.

- Communication in synchronous rounds. Local computation inbetween.
- Each round neighbors exchange $\tilde{O}(1)$-bit msgs.
- Initially: nodes know only their neighbors' IDs and incident weights.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

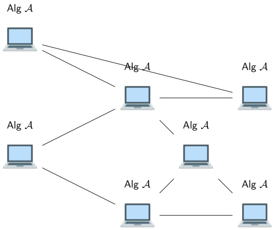


Figure: Network G.

- Communication in synchronous rounds. Local computation inbetween.
- Each round neighbors exchange $\tilde{O}(1)$-bit msgs.
- Initially: nodes know only their neighbors' IDs and incident weights.
- Objective: minimize # rounds.

What is the distributed setting?

- Communication: Network topology (read: undirected graph) $G = (V, E)$ with $|V| = n$ nodes and **hop-diameter** D.
- Problem-specific input: A set of weights (SSSP inputs) w and a source $s \in V$ (and $t \in V$).

Communication model: CONGEST [Peleg; 2000]

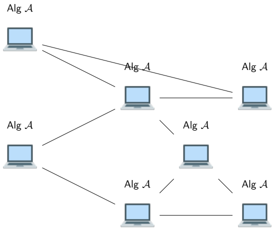
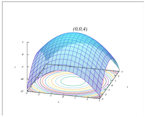


Figure: Network G.

- Communication in synchronous rounds. Local computation inbetween.
- Each round neighbors exchange $\tilde{O}(1)$-bit msgs.
- Initially: nodes know only their neighbors' IDs and incident weights.
- Objective: minimize # rounds.
- Note: D does not depend on the weights.

Today: SSSP that can be parallelized and distributed efficiently.

Recent results that develop fastest parallel SSSP algorithm use techniques from **continuous optimization**.

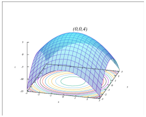


Parallel.
$(1+\varepsilon)$-apx with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. [Li; 2020] [ASZ; 2020]

Distributed.

Today: SSSP that can be parallelized and distributed efficiently.

Recent results that develop fastest parallel SSSP algorithm use techniques from **continuous optimization**.

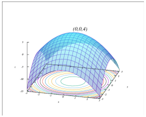


Parallel.
$(1+\varepsilon)$-apx with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. [Li; 2020] [ASZ; 2020]

Distributed.
$(1+\varepsilon)$-apx in $\tilde{O}(\sqrt{n}+D)$ rounds. [BFKL; 2016]

Today: SSSP that can be parallelized and distributed efficiently.

Recent results that develop fastest parallel SSSP algorithm use techniques from **continuous optimization**.



Parallel. [Li; 2020]

$(1+\varepsilon)$-apx with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. [ASZ; 2020]

Distributed.

$(1+\varepsilon)$-apx in $\tilde{O}(\sqrt{n}+D)$ rounds. [BFKL; 2016]

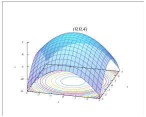
$n^{o(1)}$-apx in $OPT(G) \cdot n^{o(1)}$ rounds. [Haeupler, Li; 2018]

What is $OPT(G)$?

Def: Any correct SSSP algorithm on G requires $\geq OPT(G)$ rounds.

Today: SSSP that can be parallelized and distributed efficiently.

Recent results that develop fastest parallel SSSP algorithm use techniques from **continuous optimization**.



Parallel.

$(1+\varepsilon)$-apx with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. [Li; 2020] [ASZ; 2020]

Distributed.

$(1+\varepsilon)$-apx in $\tilde{O}(\sqrt{n}+D)$ rounds. [BFKL; 2016]

$n^{o(1)}$-apx in $OPT(G)\cdot n^{o(1)}$ rounds. [Haeupler, Li; 2018]

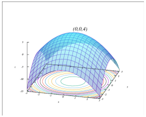
$(1+\varepsilon)$-apx in $OPT(G)\cdot n^{o(1)}$ rounds. **This talk.**

What is $OPT(G)$?

Def: Any correct SSSP algorithm on G requires $\geq OPT(G)$ rounds.

Today: SSSP that can be parallelized and distributed efficiently.

Recent results that develop fastest parallel SSSP algorithm use techniques from **continuous optimization**.



Parallel.

$(1+\varepsilon)$-apx with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. [Li; 2020] [ASZ; 2020]

Distributed.

$(1+\varepsilon)$-apx in $\tilde{O}(\sqrt{n}+D)$ rounds. [BFKL; 2016]

$n^{o(1)}$-apx in $OPT(G)\cdot n^{o(1)}$ rounds. [Haeupler, Li; 2018]

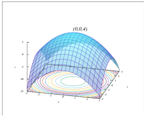
$(1+\varepsilon)$-apx in $OPT(G)\cdot n^{o(1)}$ rounds. **This talk.**

What is $OPT(G)$?

Def: Any correct SSSP algorithm on G requires $\geq OPT(G)$ rounds.
Any $\left(OPT(G)\cdot n^{o(1)}\right)$-round algo is called (almost) **universally optimal**.

Today: SSSP that can be parallelized and distributed efficiently.

Recent results that develop fastest parallel SSSP algorithm use techniques from **continuous optimization**.



Parallel.

$(1+\varepsilon)$-apx with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. [Li; 2020] [ASZ; 2020]

Distributed.

$(1+\varepsilon)$-apx in $\tilde{O}(\sqrt{n}+D)$ rounds. [BFKL; 2016]

$n^{o(1)}$-apx in $OPT(G) \cdot n^{o(1)}$ rounds. [Haeupler, Li; 2018]

$(1+\varepsilon)$-apx in $OPT(G) \cdot n^{o(1)}$ rounds. **This talk.**

What is $OPT(G)$?

Def: Any correct SSSP algorithm on G requires $\geq OPT(G)$ rounds.
Any $\left(OPT(G) \cdot n^{o(1)}\right)$-round algo is called (almost) **universally optimal**.
Thm: $OPT(G) =_{\tilde{\Theta}(1)} ShortcutQuality(G)$ [Haeupler, Wajc, Zuzic; 2021]

1 Introduction

2 Main Ideas

- Idea 1: Transshipment generalizes shortest path
- Idea 2: Transshipment boosting
- Idea 3: Approximately Solving Transshipment
- Idea 4: Distributed Implementation

3 Conclusion

Idea 1: Transshipment generalizes shortest path

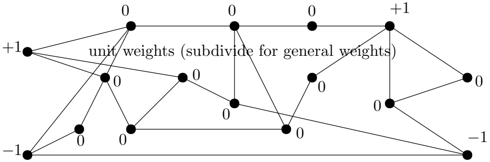
Transshipment.

Idea 1: Transshipment generalizes shortest path

Transshipment. Given a graph $G = (V, E)$ and a demand vector $d \in \mathbb{R}^V$ satisfying $\sum_v d(v) = 0$. Find a flow of minimum cost that satisfies the demands.

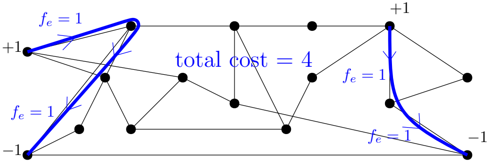
Idea 1: Transshipment generalizes shortest path

Transshipment. Given a graph $G = (V, E)$ and a demand vector $d \in \mathbb{R}^V$ satisfying $\sum_v d(v) = 0$. Find a flow of minimum cost that satisfies the demands.



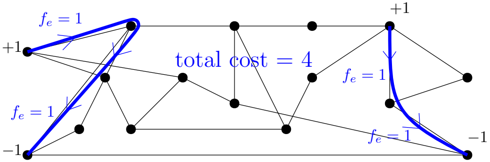
Idea 1: Transshipment generalizes shortest path

Transshipment. Given a graph $G = (V, E)$ and a demand vector $d \in \mathbb{R}^V$ satisfying $\sum_v d(v) = 0$. Find a flow of minimum cost that satisfies the demands.



Idea 1: Transshipment generalizes shortest path

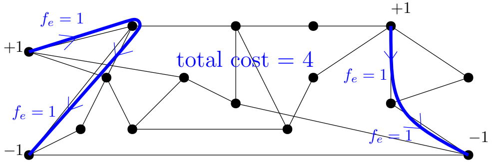
Transshipment. Given a graph $G = (V, E)$ and a demand vector $d \in \mathbb{R}^V$ satisfying $\sum_v d(v) = 0$. Find a flow of minimum cost that satisfies the demands.



Also known as: uncapacitated min-cost flow, Wasserstein metric, optimal transport, transshipment.

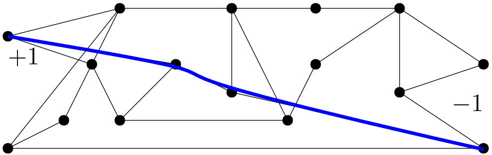
Idea 1: Transshipment generalizes shortest path

Transshipment. Given a graph $G = (V, E)$ and a demand vector $d \in \mathbb{R}^V$ satisfying $\sum_v d(v) = 0$. Find a flow of minimum cost that satisfies the demands.



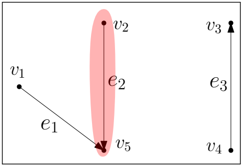
Also known as: uncapacitated min-cost flow, Wasserstein metric, optimal transport, transshipment.

Note. Generalizes $(s - t)$ shortest path. (Also generalizes SSSP.)



Transshipment: a primal-dual formulation

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



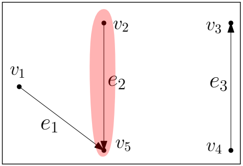
$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \dots \end{array} \\ \left[\begin{array}{cccc} +1 & & & \dots \\ & +1 & & \dots \\ & & -1 & \dots \\ & & +1 & \dots \\ -1 & -1 & & \dots \end{array}\right] \end{array}$$

Primal.

Dual.

Transshipment: a primal-dual formulation

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \dots \end{array} \\ \begin{bmatrix} +1 & & & \dots \\ & +1 & & \dots \\ & & -1 & \dots \\ & & +1 & \dots \\ -1 & -1 & & \dots \end{bmatrix} \end{array}$$

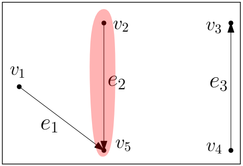
Primal.

$$\boxed{\min_f \quad \|f\|_1 : Bf = d}$$

Dual.

Transshipment: a primal-dual formulation

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \cdots \end{array} \\ \begin{bmatrix} +1 & & & \cdots \\ & +1 & & \cdots \\ & & -1 & \cdots \\ & & +1 & \cdots \\ -1 & -1 & & \cdots \end{bmatrix} \end{array}$$

Primal.

$$\boxed{\min_f \ \ \|f\|_1 : Bf = d}$$

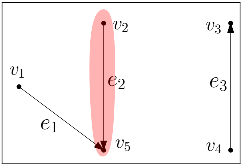
$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

Dual.

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \cdots \end{array} \\ \left[\begin{array}{cccc} +1 & & & \cdots \\ & +1 & & \cdots \\ & & -1 & \cdots \\ & & +1 & \cdots \\ -1 & -1 & & \cdots \end{array} \right] \end{array}$$

Primal.

$$\boxed{\min_f \quad \|f\|_1 : Bf = d}$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

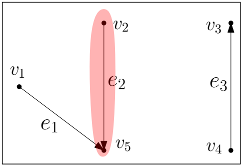
$f_e < 0$ if flow in opposite direction

$(Bf)_v = 0$ if flow conserved at v

Dual.

Transshipment: a primal-dual formulation

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \cdots \end{array} \\ \begin{bmatrix} +1 & & & \cdots \\ & +1 & & \cdots \\ & & -1 & \cdots \\ & & +1 & \cdots \\ -1 & -1 & & \cdots \end{bmatrix} \end{array}$$

Primal.

$$\boxed{\min_f \ \ \|f\|_1 : Bf = d}$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

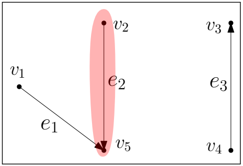
$(Bf)_v = 0$ if flow conserved at v

SP .. $f^* =$ shortest path from s to t

Dual.

Transshipment: a primal-dual formulation

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & {\color{red} e_2} & e_3 & \cdots \end{array} \\ \left[\begin{array}{cccc} +1 & & & \cdots \\ & {\color{red}+1} & & \cdots \\ & & -1 & \cdots \\ & & +1 & \cdots \\ -1 & {\color{red}-1} & & \cdots \end{array}\right] \end{array}$$

Primal.

$$\min_f \quad \|f\|_1 : Bf = d$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

$(Bf)_v = 0$ if flow conserved at v

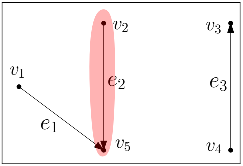
SP .. $f^* =$ shortest path from s to t

Dual.

$$\max_\phi \quad \langle d, \phi \rangle : \left\| B^\top \phi \right\|_\infty \leq 1.$$

Transshipment: a primal-dual formulation

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \dots \end{array} \\ \begin{bmatrix} +1 & & & \dots \\ & +1 & & \dots \\ & & -1 & \dots \\ & & +1 & \dots \\ -1 & -1 & & \dots \end{bmatrix} \end{array}$$

Primal.

$$\boxed{\min_f \ \ \|f\|_1 : Bf = d}$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

$(Bf)_v = 0$ if flow conserved at v

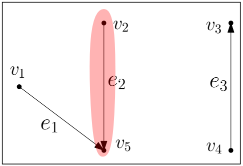
SP .. $f^* =$ shortest path from s to t

Dual.

$$\boxed{\max_\phi \ \ \langle d, \phi \rangle : \left\| B^\top \phi \right\|_\infty \leq 1.}$$

$\phi_v =$ potential (height) of v

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & \dots \end{array} \\ \begin{bmatrix} +1 & & & \dots \\ & +1 & & \dots \\ & & -1 & \dots \\ & & +1 & \dots \\ -1 & -1 & & \dots \end{bmatrix} \end{array}$$

Primal.

$$\boxed{\min_f \ \|f\|_1 : Bf = d}$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

$(Bf)_v = 0$ if flow conserved at v

SP .. $f^* =$ shortest path from s to t

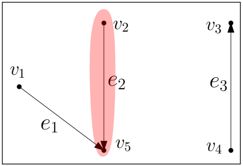
Dual.

$$\boxed{\max_\phi \ \langle d, \phi \rangle : \left\|B^\top \phi\right\|_\infty \le 1.}$$

$\phi_v =$ potential (height) of v

$(B^T\phi)_e = \phi_a - \phi_b$ is height difference

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & \color{red}{e_2} & e_3 & \cdots \end{array} \\ \begin{bmatrix} +1 & & & \cdots \\ & \color{red}{+1} & & \cdots \\ & & -1 & \cdots \\ & & +1 & \cdots \\ -1 & \color{red}{-1} & & \cdots \end{bmatrix} \end{array}$$

Primal.

$$\min_f \quad \|f\|_1 : Bf = d$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

$(Bf)_v = 0$ if flow conserved at v

SP .. $f^* =$ shortest path from s to t

Dual.

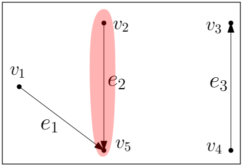
$$\max_\phi \quad \langle d, \phi \rangle : \left\|B^\top \phi\right\|_\infty \leq 1.$$

$\phi_v =$ potential (height) of v

$(B^T\phi)_e = \phi_a - \phi_b$ is height difference

$\left\|B^\top \phi\right\|_\infty \leq 1$ height diff must be small

Write the graph $G = (V, E)$ using the node-edge incidence matrix B.
Note: we orient edges arbitrarily.



$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & \color{red}{e_2} & e_3 & \cdots \end{array} \\ \begin{bmatrix} +1 & & & \dots \\ & \color{red}{+1} & & \dots \\ & & -1 & \dots \\ & & +1 & \dots \\ -1 & \color{red}{-1} & & \dots \end{bmatrix} \end{array}$$

Primal.

$$\min_f \quad \|f\|_1 : Bf = d$$

$f_e = 0$ if no flow along e

$f_e > 0$ if flow in same direction as e

$f_e < 0$ if flow in opposite direction

$(Bf)_v = 0$ if flow conserved at v

SP .. $f^* =$ shortest path from s to t

Dual.

$$\max_\phi \quad \langle d, \phi \rangle : \left\|B^\top \phi\right\|_\infty \leq 1.$$

$\phi_v =$ potential (height) of v

$(B^T\phi)_e = \phi_a - \phi_b$ is height difference

$\left\|B^\top \phi\right\|_\infty \leq 1$ height diff must be small

SP .. $\phi_v^* =$ distance of v from source

Idea 2: Transshipment boosting

Why transshipment? Isn't it harder?

Amazing property: we can boost a bad approximation to a good approximation.

Idea 2: Transshipment boosting

Why transshipment? Isn't it harder?

Amazing property: we can boost a bad approximation to a good approximation.

Primal.

$\min_f \ \|f\|_1 : Bf = d$

Dual.

$\max_\phi \ \langle d, \phi \rangle : \left\|B^\top \phi\right\|_\infty \leq 1.$

Idea 2: Transshipment boosting

Why transshipment? Isn't it harder?

Amazing property: we can boost a bad approximation to a good approximation.

Primal.

$\min_f \ \|f\|_1 : Bf = d$

Dual.

$\max_\phi \ \langle d, \phi \rangle : \left\|B^\top \phi\right\|_\infty \leq 1.$

Theorem ([Sherman; 2013], [BFKL; 2016], [Zuzic; unpublished])

Fix G. Suppose we are given an oracle $O_G(\cdot)$ which, given a demand d, outputs an α-approximate feasible dual $O_G(d)$. There is an algorithm that produces a $(1+\varepsilon)$-approximate feasible dual by calling $O_G(\cdot)$ at most $\mathrm{poly}(\alpha, \varepsilon^{-1}, \log n)$ times.

Idea 2: Transshipment boosting

Why transshipment? Isn't it harder?

Amazing property: we can boost a bad approximation to a good approximation.

Primal.

$\min_f \ \|f\|_1 : Bf = d$

Dual.

$\max_\phi \ \langle d, \phi \rangle : \|B^\top \phi\|_\infty \leq 1.$

Theorem ([Sherman; 2013], [BFKL; 2016], [Zuzic; unpublished])

Fix G. Suppose we are given an oracle $O_G(\cdot)$ which, given a demand d, outputs an α-approximate feasible dual $O_G(d)$. There is an algorithm that produces a $(1+\varepsilon)$-approximate feasible dual by calling $O_G(\cdot)$ at most $\mathrm{poly}(\alpha, \varepsilon^{-1}, \log n)$ times.

Corollary

Given such a (dual) $n^{o(1)}$-approximation oracle, we can solve $(1+\frac{1}{n^{o(1)}})$-approximate transshipment in $n^{o(1)}$ oracle calls.

Idea 3: Approximately Solving Transshipment

Goal: find an approximate ~~dual~~ solution

Prerequisite: **Low-diameter decomposition (LDD)**.

Idea 3: Approximately Solving Transshipment

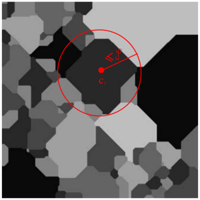
Goal: find an approximate ~~dual~~ solution

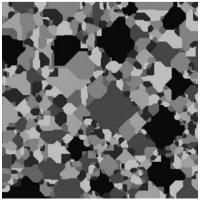
Prerequisite: **Low-diameter decomposition (LDD)**.

Definition

For a graph G, a low-diameter decomposition (LDD) of radius ρ is a distribution over node partitions into **clusters** $V = S_1 \sqcup \ldots \sqcup S_k$ along with centers $c_1 \in S_1, \ldots, c_k \in S_k$ such that:

1. For each i, the center c_i is within distance ρ of every other node in the induced subgraph $G[S_i]$, w.h.p.
2. For all $x, y \in V$, the probability they are in different clusters is at most $2^{\sqrt{\log n}} \cdot \frac{dist_G(u,v)}{\rho}$.





[Miller, Peng, Xu; 2013]

Idea 3: Approximately Solving Transshipment

Goal: find an approximate ~~dual~~ solution

Prerequisite: **Low-diameter decomposition (LDD)**.

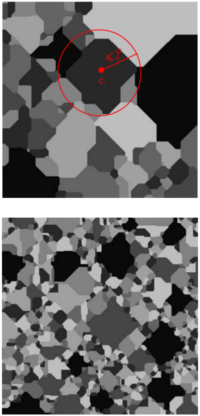
Definition

For a graph G, a low-diameter decomposition (LDD) of radius ρ is a distribution over node partitions into **clusters** $V = S_1 \sqcup \ldots \sqcup S_k$ along with centers $c_1 \in S_1, \ldots, c_k \in S_k$ such that:

1. For each i, the center c_i is within distance ρ of every other node in the induced subgraph $G[S_i]$, w.h.p.
2. For all $x, y \in V$, the probability they are in different clusters is at most $2^{\sqrt{\log n}} \cdot \frac{dist_G(u,v)}{\rho}$.

Theorem (Prior work [Haeupler, Li; 2018])

LDDs can be sampled in $OPT(G)n^{o(1)}$ CONGEST rounds.



[Miller, Peng, Xu; 2013]

Idea 3: Approximately Solving Transshipment

Algorithm 0: Oblivious routing for TS.

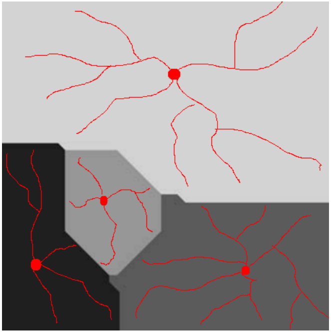
1. Let $\rho := 2^{(\log n)^{3/4}}$ (LDD radius).

   1. Sample an LDD with radius ρ^i.
   2. Each v sends $\frac{1}{g}$-fraction of its demend to the center of cluster containing v.

Algorithm 0: Oblivious routing for TS.

1. Let $\rho := 2^{(\log n)^{3/4}}$ (LDD radius).

   1. Sample an LDD with radius ρ^i.
   2. Each v sends $\frac{1}{g}$-fraction of its demend to the center of cluster containing v.



Idea 3: Approximately Solving Transshipment

Algorithm 0: Oblivious routing for TS.

1. Let $\rho := 2^{(\log n)^{3/4}}$ (LDD radius).

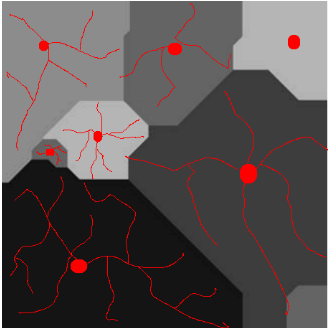
    1. For $j = 1, 2, \ldots, g := 2^{(\log n)^{3/4}}$.
        1. Sample an LDD with radius ρ^i.
        2. Each v sends $\frac{1}{g}$-fraction of its demend to the center of cluster containing v.
    2. Update the demand to reflect the transport.

Idea 3: Approximately Solving Transshipment

Algorithm 0: Oblivious routing for TS.

1. Let $\rho := 2^{(\log n)^{3/4}}$ (LDD radius).

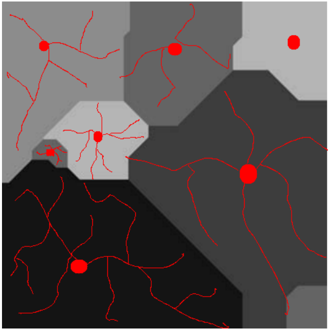
   1. For $j = 1, 2, \ldots, g := 2^{(\log n)^{3/4}}$.
      1. Sample an LDD with radius ρ^i.
      2. Each v sends $\frac{1}{g}$-fraction of its demend to the center of cluster containing v.
   2. Update the demand to reflect the transport.



Idea 3: Approximately Solving Transshipment

Algorithm 0: Oblivious routing for TS.

1. Let $\rho := 2^{(\log n)^{3/4}}$ (LDD radius).
2. For $i = 1, 2, \ldots, (\log n)^{1/4}$ repeat the following:
   1. For $j = 1, 2, \ldots, g := 2^{(\log n)^{3/4}}$.
      1. Sample an LDD with radius ρ^i.
      2. Each v sends $\frac{1}{g}$-fraction of its demend to the center of cluster containing v.
   2. Update the demand to reflect the transport.
3. Route all remaining demand to a common node along any spanning tree.



When $i = (\log n)^{1/4}$, radius is $\rho^i = \text{poly}(n)$ and LDD has a single cluster.

Analysis intuition.

Question: how does OPT change between steps?

Fix u, v at distance ℓ. Suppose at some step $d(u) = +1, d(v) = -1, d(\text{all else}) = 0$. Clearly, $\mathrm{OPT}_{before} = \ell$. Suppose we sampled an LDD of radius ρ. How does OPT change?

Analysis intuition.

Question: how does OPT change between steps?

Fix u, v at distance ℓ. Suppose at some step $d(u) = +1, d(v) = -1, d(\text{all else}) = 0$. Clearly, $\mathrm{OPT}_{before} = \ell$. Suppose we sampled an LDD of radius ρ. How does OPT change?

- If u, v in same cluster, we are happy (the demand cancels out).

Question: how does OPT change between steps?

Fix u, v at distance ℓ. Suppose at some step $d(u) = +1, d(v) = -1, d(\text{all else}) = 0$. Clearly, $\mathrm{OPT}_{before} = \ell$. Suppose we sampled an LDD of radius ρ. How does OPT change?

- If u, v in same cluster, we are happy (the demand cancels out).
- If u, v in different clusters, they are now at distance $\rho + \ell + \rho$ apart.
  - When $\ell \gg \rho$, this is still $O(\ell)$.

Analysis intuition.

Question: how does OPT change between steps?

Fix u, v at distance ℓ. Suppose at some step $d(u) = +1, d(v) = -1, d(\text{all else}) = 0$. Clearly, $\mathrm{OPT}_{before} = \ell$. Suppose we sampled an LDD of radius ρ. How does OPT change?

- If u, v in same cluster, we are happy (the demand cancels out).
- If u, v in different clusters, they are now at distance $\rho + \ell + \rho$ apart.
  - When $\ell \gg \rho$, this is still $O(\ell)$.
  - When $\rho \gg \ell$. Remember that separation happens with probability $2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho}$. In expectation:

$$2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho} \cdot (2\rho + \ell) = 2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho} \cdot O(\rho) = 2^{\sqrt{\log n}} \cdot O(\ell)$$

Analysis intuition.

Question: how does OPT change between steps?

Fix u, v at distance ℓ. Suppose at some step $d(u) = +1, d(v) = -1, d(\text{all else}) = 0$. Clearly, $\mathrm{OPT}_{before} = \ell$. Suppose we sampled an LDD of radius ρ. How does OPT change?

- If u, v in same cluster, we are happy (the demand cancels out).
- If u, v in different clusters, they are now at distance $\rho + \ell + \rho$ apart.
  - When $\ell \gg \rho$, this is still $O(\ell)$.
  - When $\rho \gg \ell$. Remember that separation happens with probability $2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho}$. In expectation:

$$2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho} \cdot (2\rho + \ell) = 2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho} \cdot O(\rho) = 2^{\sqrt{\log n}} \cdot O(\ell)$$

In both cases, OPT increases by at most $2^{\sqrt{\log n}}$ factor. Hence after $(\log n)^{1/4}$ steps, it only increases by

$$\left(2^{\sqrt{\log n}}\right)^{(\log n)^{1/4}} = 2^{(\log n)^{3/4}} = n^{o(1)}$$

.

Issue: On last slide, we proved the new OPT grows slowly .. BUT only looked at a fixed "pair" demand.

Claim

After repeating LDD sampling $2^{(\log n)^{3/4}}$ times, we get concentration and it holds for all demands.

Issue: On last slide, we proved the new OPT grows slowly .. BUT only looked at a fixed "pair" demand.

Claim

After repeating LDD sampling $2^{(\log n)^{3/4}}$ times, we get concentration and it holds for all demands.

Question: Why not simply start at $\rho = \text{poly}(n)$?

- New OPT is OK in expectation.

Issue: On last slide, we proved the new OPT grows slowly .. BUT only looked at a fixed "pair" demand.

Claim

After repeating LDD sampling $2^{(\log n)^{3/4}}$ times, we get concentration and it holds for all demands.

Question: Why not simply start at $\rho = \text{poly}(n)$?

- New OPT is OK in expectation.
- Reason 1: we would need $\text{poly}(n)$ repetitions to get concentration.

Issue: On last slide, we proved the new OPT grows slowly .. BUT only looked at a fixed "pair" demand.

Claim

After repeating LDD sampling $2^{(\log n)^{3/4}}$ times, we get concentration and it holds for all demands.

Question: Why not simply start at $\rho = \text{poly}(n)$?

- New OPT is OK in expectation.
- Reason 1: we would need poly(n) repetitions to get concentration.

Issue: On last slide, we proved the new OPT grows slowly .. BUT only looked at a fixed "pair" demand.

Claim

After repeating LDD sampling $2^{(\log n)^{3/4}}$ times, we get concentration and it holds for all demands.

Question: Why not simply start at $\rho = \text{poly}(n)$?

- New OPT is OK in expectation.
- Reason 1: we would need $\text{poly}(n)$ repetitions to get concentration.
- Reason 2: the transport cost in this single step would be too high.

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

1. Each node chooses a private $\tilde{O}(1)$-bit value x_v.

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

1. Each node chooses a private $\tilde{O}(1)$-bit value x_v.
2. We contract a subset of edges.

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

1. Each node chooses a private $\tilde{O}(1)$-bit value x_v.
2. We contract a subset of edges.
3. For each supernode $S \subseteq V$, define $x_S := \bigoplus_{x \in S} x_v$.

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

1. Each node chooses a private $\tilde{O}(1)$-bit value x_v.
2. We contract a subset of edges.
3. For each supernode $S \subseteq V$, define $x_S := \bigoplus_{x \in S} x_v$.
4. (Each node in) each supernode receives an aggregate of adjacent supernodes' values.

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

1. Each node chooses a private $\tilde{O}(1)$-bit value x_v.
2. We contract a subset of edges.
3. For each supernode $S \subseteq V$, define $x_S := \bigoplus_{x \in S} x_v$.
4. (Each node in) each supernode receives an aggregate of adjacent supernodes' values.

Theorem (Many prior and concurrent papers)

A Minor-Aggregation round can be simulated in $OPT(G) \cdot n^{o(1)}$ CONGEST rounds.

Idea 4: Distributed Implementation

Designing distributed algorithms in CONGEST is hard. We propose a new model.

Distributed Minor-Aggregation model. In each round:

1. Each node chooses a private $\tilde{O}(1)$-bit value x_v.
2. We contract a subset of edges.
3. For each supernode $S \subseteq V$, define $x_S := \bigoplus_{x \in S} x_v$.
4. (Each node in) each supernode receives an aggregate of adjacent supernodes' values.

Theorem (Many prior and concurrent papers)

A Minor-Aggregation round can be simulated in $OPT(G) \cdot n^{o(1)}$ CONGEST rounds.

Corollary

Transshipment can be solved in $n^{o(1)}$ Minor-Aggregation rounds. Hence it can be implemented in $OPT(G) \cdot n^{o(1)}$ CONGEST rounds.

Conclusion

Idea 1: don't solve SSSP. Solve transshipment.

Conclusion

Idea 1: don't solve SSSP. Solve transshipment.

Idea 2: transshipment can be boosted, hence we only need to compute a $n^{o(1)}$-approximation.

Conclusion

Idea 1: don't solve SSSP. Solve transshipment.

Idea 2: transshipment can be boosted, hence we only need to compute a $n^{o(1)}$-approximation.

Idea 3: oblivious routing for transshipment. Find LDD, send to center, repeat many times, increaase LDD radius until we consume the entire graph.

Conclusion

Idea 1: don't solve SSSP. Solve transshipment.

Idea 2: transshipment can be boosted, hence we only need to compute a $n^{o(1)}$-approximation.

Idea 3: oblivious routing for transshipment. Find LDD, send to center, repeat many times, increaase LDD radius until we consume the entire graph.

Idea 4: implement in distributed setting using minor-aggregations. Convert to a universally-optimal CONGEST algorithm.

Conclusion

Idea 1: don't solve SSSP. Solve transshipment.

Idea 2: transshipment can be boosted, hence we only need to compute a $n^{o(1)}$-approximation.

Idea 3: oblivious routing for transshipment. Find LDD, send to center, repeat many times, increaase LDD radius until we consume the entire graph.

Idea 4: implement in distributed setting using minor-aggregations. Convert to a universally-optimal CONGEST algorithm.

Future directions: (1) lose polylog factors instead of $n^{o(1)}$, (2) make deterministic.

Conclusion

Idea 1: don't solve SSSP. Solve transshipment.

Idea 2: transshipment can be boosted, hence we only need to compute a $n^{o(1)}$-approximation.

Idea 3: oblivious routing for transshipment. Find LDD, send to center, repeat many times, increaase LDD radius until we consume the entire graph.

Idea 4: implement in distributed setting using minor-aggregations. Convert to a universally-optimal CONGEST algorithm.

Future directions: (1) lose polylog factors instead of $n^{o(1)}$, (2) make deterministic.

Thank you!