Compact Oblivious Routing in Weighted Graphs

Philipp Czerner, Harald Räcke
Fakultät für Informatik, TU München

August 25, 2020

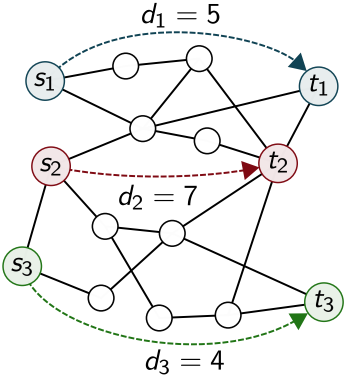
Compact Oblivious Routing

Input:

- undirected graph
- source-target pairs (s_i, t_i)
- demands d_i

Output:

- s_i-t_i flows f_i with value d_i



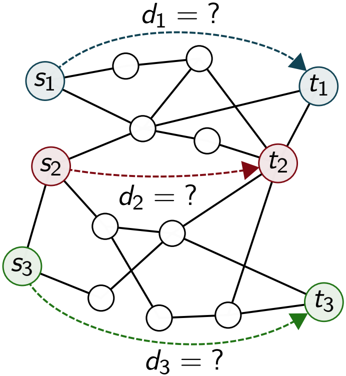
Compact Oblivious Routing

Input:

- undirected graph
- source-target pairs (s_i, t_i)
- ~~demands d_i~~

Output:

- s_i-t_i flows f_i with value 1



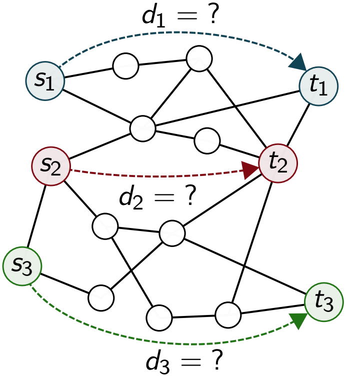
Compact Oblivious Routing

Input:

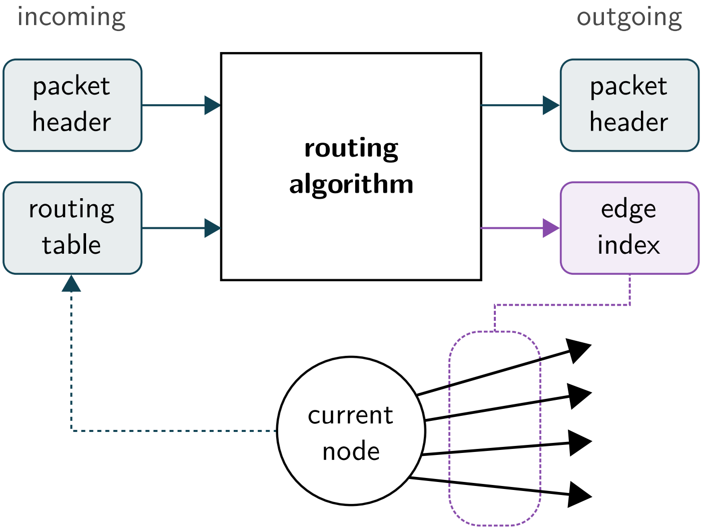
- undirected graph
- source-target pairs (s_i, t_i)
- ~~demands d_i~~

Output:

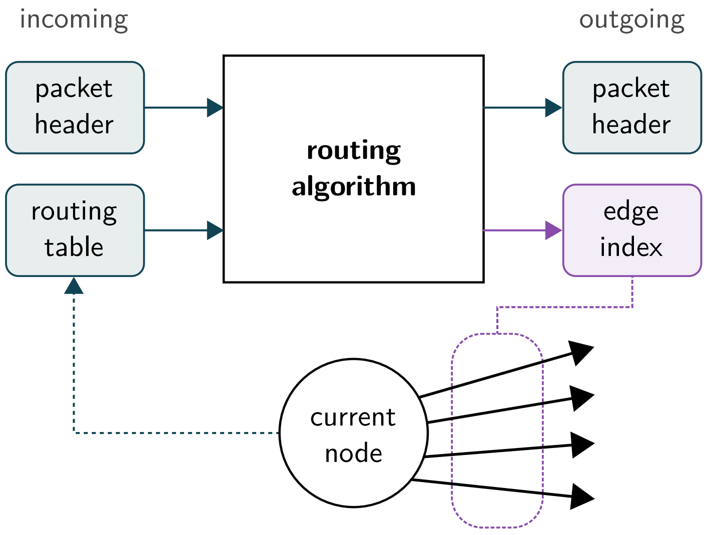
- s_i-t_i flows with value 1
- specified implicitly by a routing algorithm



Routing Algorithm



Routing Algorithm



- packet header initialised to target's node label

Cost Measures

- competitive ratio
  1. total load
  $$\sum_e \sum_i \frac{d_i f_i(e)}{w(e)}$$
  2. maximum load (congestion)
  $$\max_e \sum_i \frac{d_i f_i(e)}{w(e)}$$
- space complexity
  - node labels
  - routing tables
  - packet headers

Cost Measures

- competitive ratio
  1. total load

$$\sum_e \sum_i \frac{d_i f_i(e)}{w(e)}$$

  2. maximum load (congestion)

$$\max_e \sum_i \frac{d_i f_i(e)}{w(e)}$$

shortest path routing already oblivious
many results on compact routing schemes

- space complexity
  - node labels
  - routing tables
  - packet headers

Cost Measures

- competitive ratio
  1. total load

$$\sum_e \sum_i \frac{d_i f_i(e)}{w(e)}$$

  2. maximum load (congestion)

$$\max_e \sum_i \frac{d_i f_i(e)}{w(e)}$$

- space complexity
  - node labels
  - routing tables
  - packet headers

shortest path routing already oblivious
many results on compact routing schemes, see e.g.

[Frederickson, Janardan 1988]
[Fraigniaud, Gavoille 1995]
[Cowen 2001]
[Thorup, Zwick 2001]
[Krioukov, Fall, Yang 2004]
[Abraham et al. 2006]
[Rétvári et al. 2013]

Cost Measures

- competitive ratio
  1. total load

$$\sum_e \sum_i \frac{d_i f_i(e)}{w(e)}$$

  2. maximum load (congestion)

$$\max_e \sum_i \frac{d_i f_i(e)}{w(e)}$$

shortest path routing already oblivious

many results on compact routing schemes

only one result on compact oblivious routing w.r.t. congestion!

- space complexity
  - node labels
  - routing tables
  - packet headers

[Räcke, Schmidt 2019]

- uses a hierarchical decomposition
- competitive ratio $\tilde{\mathcal{O}}(1)$
- node labels $\tilde{\mathcal{O}}(1)$
- packet headers $\tilde{\mathcal{O}}(1)$
- routing tables $\tilde{\mathcal{O}}(\deg(v))$

[1] and graphs where the decomposition tree has bounded degree

[Räcke, Schmidt 2019]

- uses a hierarchical decomposition
- competitive ratio $\tilde{\mathcal{O}}(1)$
- node labels $\tilde{\mathcal{O}}(1)$
- packet headers $\tilde{\mathcal{O}}(1)$
- routing tables $\tilde{\mathcal{O}}(\deg(v))$

But:

- only in unweighted graphs! [1]

Our contribution: extend this to graphs with polynomial weights.

[1] and graphs where the decomposition tree has bounded degree

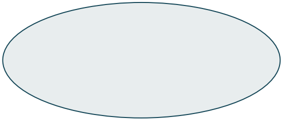
Our Result

Theorem 1. For any undirected graph with polynomial weights and n nodes there exists a compact oblivious routing scheme with:

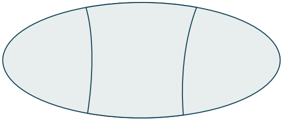
- competitive ratio $\mathcal{O}(\log^9 n)$
- node labels $\mathcal{O}(\log^2 n)$
- packet headers $\mathcal{O}(\log^3 n)$
- routing tables $\mathcal{O}(\deg(v) \log^9 n)$

Decomposition Tree

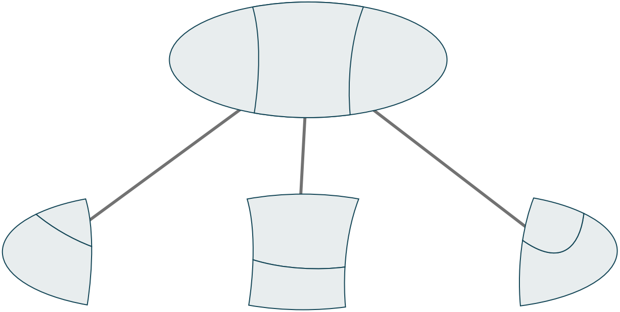
Decomposition Tree



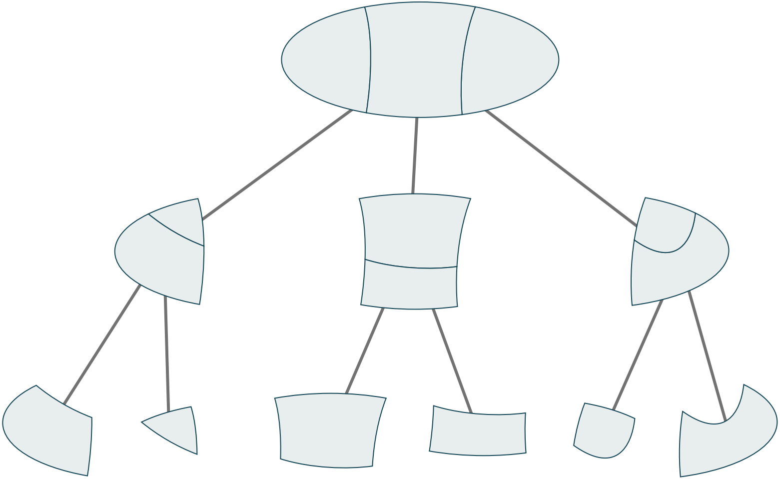
Decomposition Tree



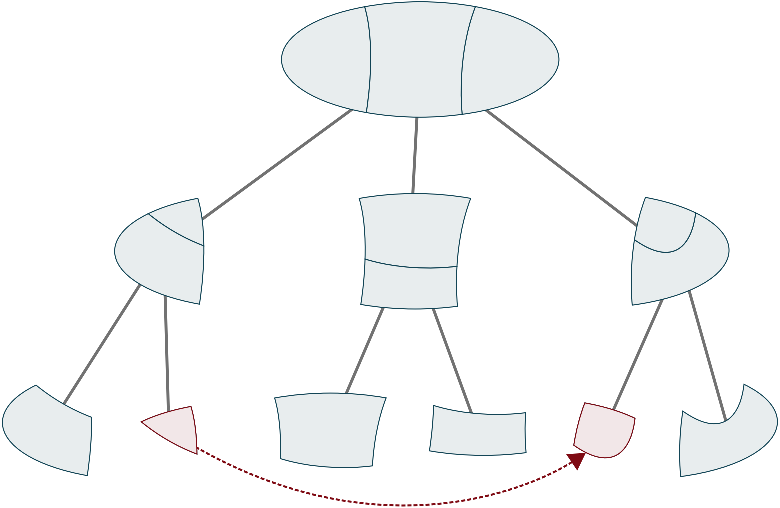
Decomposition Tree



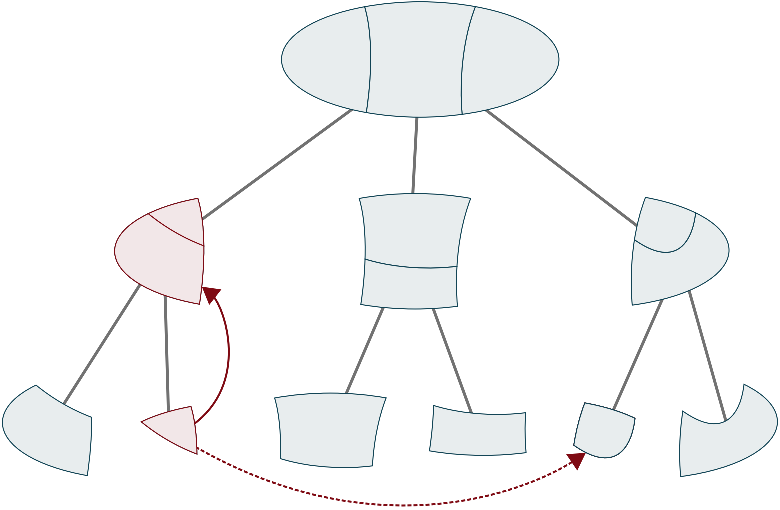
Decomposition Tree



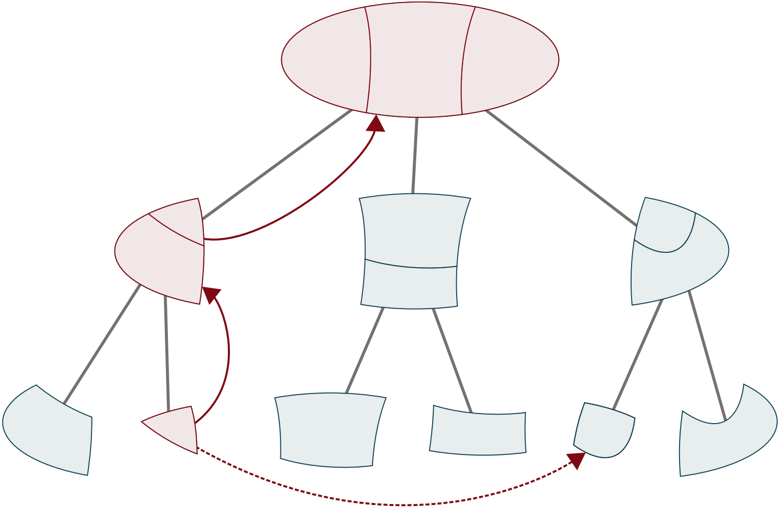
Decomposition Tree



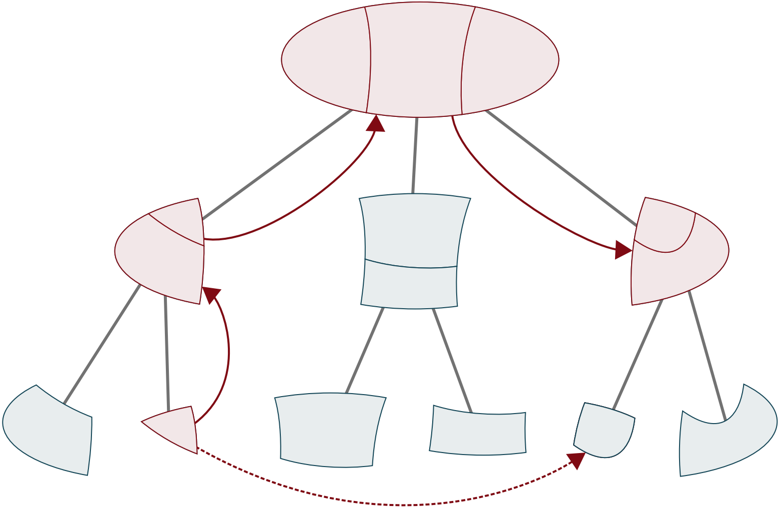
Decomposition Tree



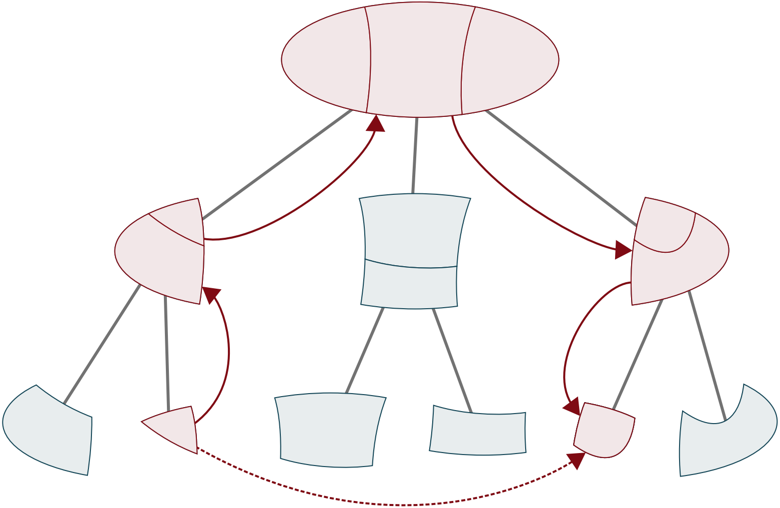
Decomposition Tree



Decomposition Tree

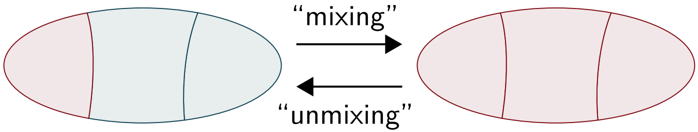


Decomposition Tree



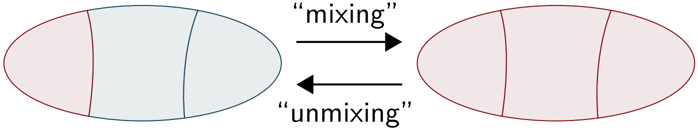
Two Directions

Our routing scheme needs to support two operations:



Two Directions

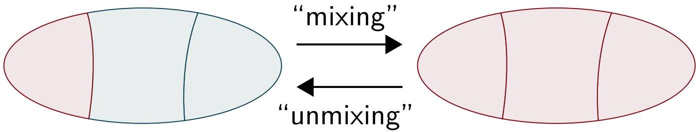
Our routing scheme needs to support two operations:



- “mixing” is easier, we start with that

Two Directions

Our routing scheme needs to support two operations:



- “mixing” is easier, we start with that

From now on, we think only about a single parent cluster.

Why are weighted graphs hard? (1)

Unweighted graph:

- Compute multi-commodity flow routing all children at once

Why are weighted graphs hard? (1)

Unweighted graph:

- Compute multi-commodity flow routing all children at once
- Possible with low congestion

Why are weighted graphs hard? (1)

Unweighted graph:

- Compute multi-commodity flow routing all children at once
- Possible with low congestion
- Use paths instead of flows (randomised rounding)

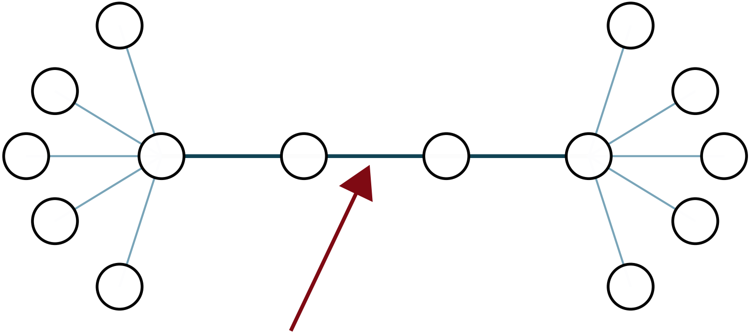
Why are weighted graphs hard? (1)

Unweighted graph:

- Compute multi-commodity flow routing all children at once
- Possible with low congestion
- Use paths instead of flows (randomised rounding)
- Due to low congestion, only few paths per edge
- Can store per-path routing information in the graph
  - $\tilde{\mathcal{O}}(\deg(v))$ per adjacent node

Why are weighted graphs hard? (1)

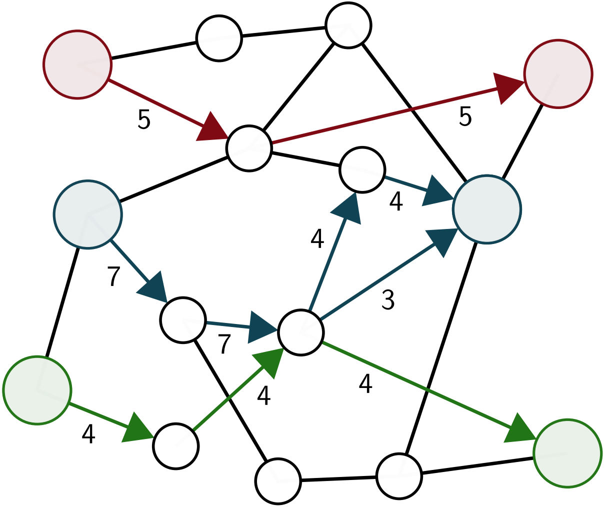
Weighted graph:



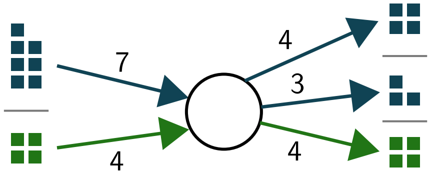
Many paths use the same edge!

Single-commodity Flows

Multi-commodity Flows?

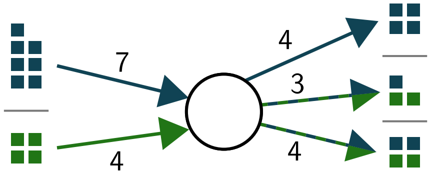


Multi-commodity Flows?



- assign each unit of flow a unique id
- determine thresholds s.t. each edge gets the right amount

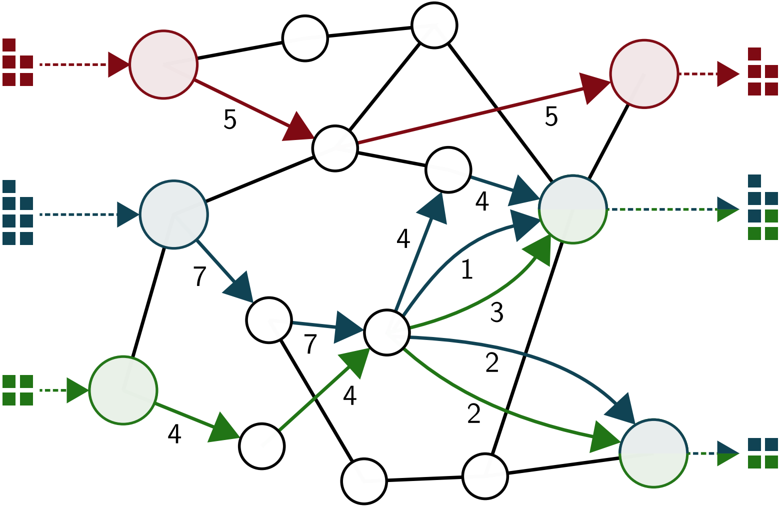
Multi-commodity Flows?



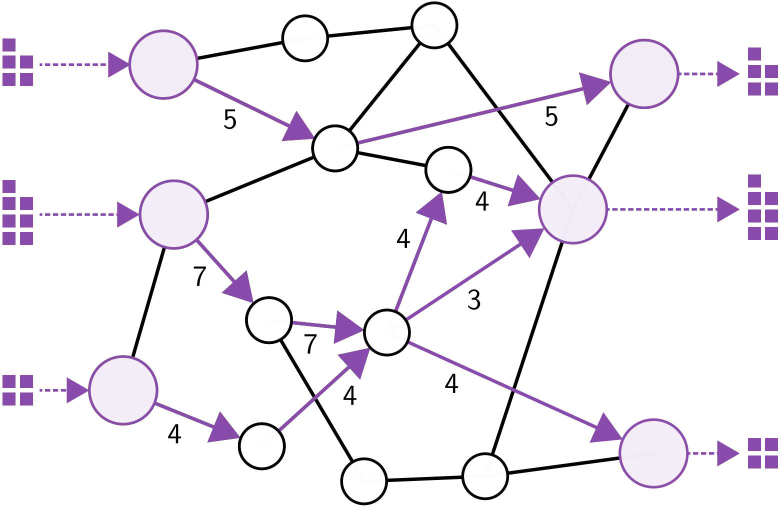
- assign each unit of flow a unique id
- determine thresholds s.t. each edge gets the right amount

We cannot keep flow of different sources separate!

Single-commodity Flows

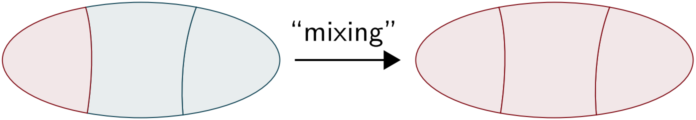


Single-commodity Flows



Allows us to use single-commodity flows as building block.

Are we done?



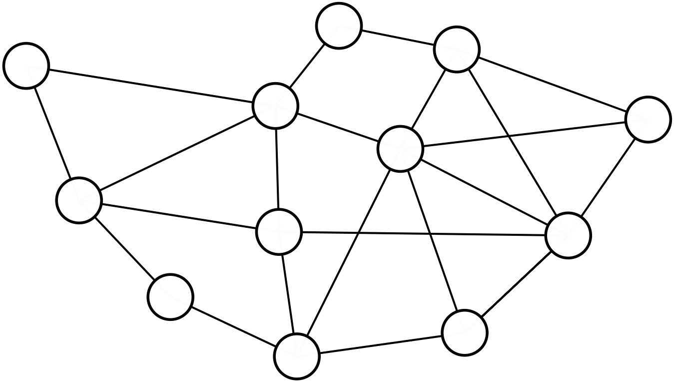
- We could embed a single-commodity flow for *each* child

Are we done?

- We could embed a single-commodity flow for *each* child
- Uses $\tilde{\mathcal{O}}(\deg(v))$ space *per child cluster* $\rightarrow$ too much

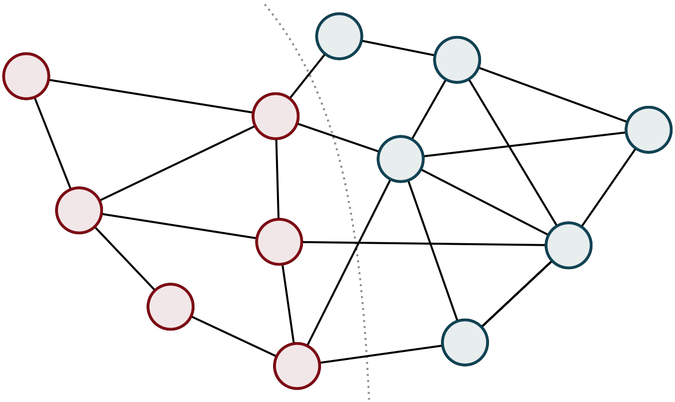
Random Walks

Cut-matching Games [Khandekar, Rao, Vazirani 2006]



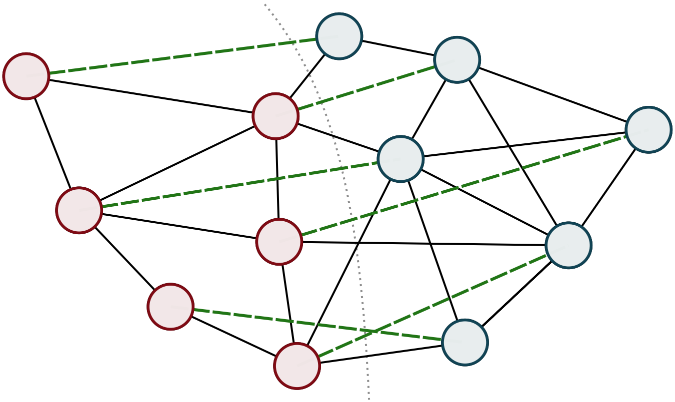
- $\tilde{\mathcal{O}}(1)$ iterations

Cut-matching Games [Khandekar, Rao, Vazirani 2006]



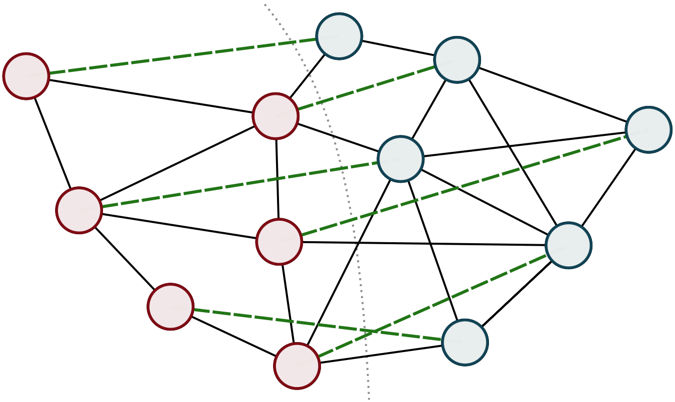
- $\tilde{\mathcal{O}}(1)$ iterations, in each we
  - are given a cut $(M, V \setminus M)$, with $|M| = |V \setminus M|$

Cut-matching Games [Khandekar, Rao, Vazirani 2006]



- $\tilde{\mathcal{O}}(1)$ iterations, in each we
  - are given a cut $(M, V \setminus M)$, with $|M| = |V \setminus M|$
  - output *any* bipartite matching on this cut

Cut-matching Games [Khandekar, Rao, Vazirani 2006]



Now we can do a random walk:

- Each iteration we throw a coin
- Either change to the matching partner or not
- Result: Uniformly distributed

Routing a Random Walk

- Idea: Use single-commodity flow instead of matching

Routing a Random Walk

- Idea: Use single-commodity flow instead of matching
- two flows, one in each direction
- the procedure still works!

We need only $\tilde{\mathcal{O}}(1)$ flows!

Mixing works!

- embed a random walk for the parent
- go from child to parent by executing the random walk

Unmixing

Hypercube Embedding

Problem:

- there can be $\Omega(n)$ child clusters
- so we must compress the routing information

Hypercube Embedding

Problem:

- there can be $\Omega(n)$ child clusters
- so we must compress the routing information

[Räcke, Schmidt 2019] use a hypercube:

Hypercube Embedding

Problem:

- there can be $\Omega(n)$ child clusters
- so we must compress the routing information

[Räcke, Schmidt 2019] use a hypercube:

- each child cluster gets a (contiguous!) range of hypercube ids

Hypercube Embedding

Problem:

- there can be $\Omega(n)$ child clusters
- so we must compress the routing information

[Räcke, Schmidt 2019] use a hypercube:

- each child cluster gets a (contiguous!) range of hypercube ids
- each node gets roughly $\mathcal{O}(\deg(v))$ hypercube ids

Hypercube Embedding

Problem:

- there can be $\Omega(n)$ child clusters
- so we must compress the routing information

[Räcke, Schmidt 2019] use a hypercube:

- each child cluster gets a (contiguous!) range of hypercube ids
- each node gets roughly $\mathcal{O}(\deg(v))$ hypercube ids
- each hypercube id has $\mathcal{O}(\log n)$ neighbours

Hypercube Embedding

Problem:

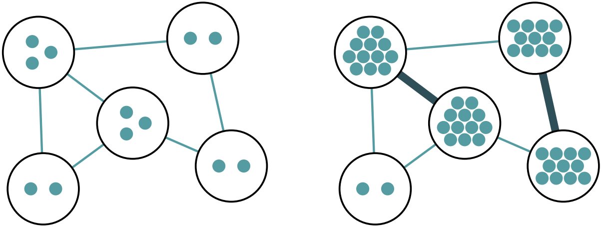
- there can be $\Omega(n)$ child clusters
- so we must compress the routing information

[Räcke, Schmidt 2019] use a hypercube:

- each child cluster gets a (contiguous!) range of hypercube ids
- each node gets roughly $\mathcal{O}(\deg(v))$ hypercube ids
- each hypercube id has $\mathcal{O}(\log n)$ neighbours

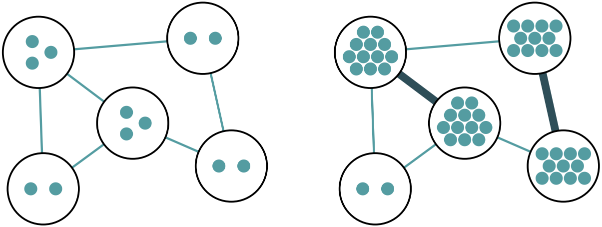
Only need to store routing information for $\tilde{\mathcal{O}}(\deg(v))$ hypercube edges in each node.

Why are weighted graphs hard? (2)

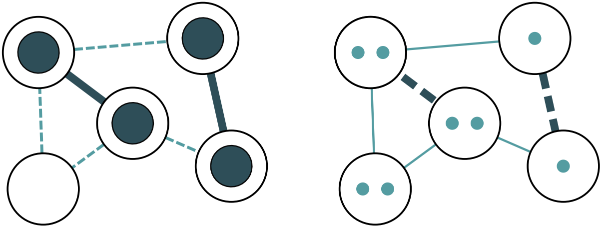


Hypercube ids are not bounded by deg(v) but *weighted* degree.

Why are weighted graphs hard? (2)



Hypercube ids are not bounded by $\deg(v)$ but *weighted* degree.

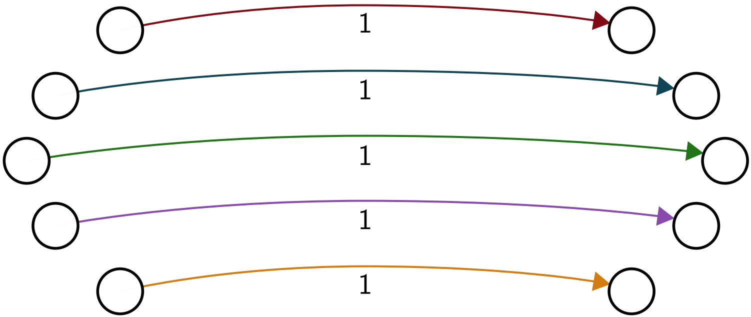


So we use $\mathcal{O}(\log n)$ hypercubes with geometric weights.

For now, consider a hypercube with weight 1.

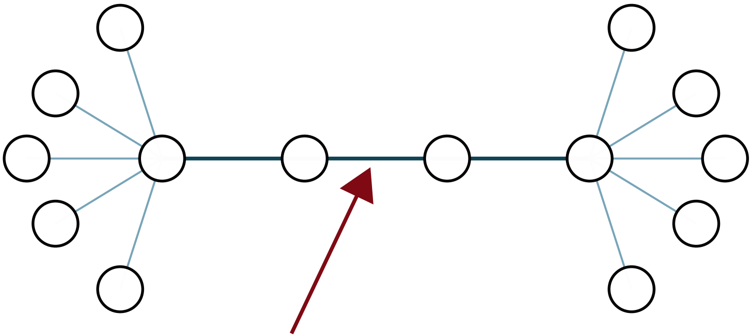
What do we need to route?

Why are weighted graphs hard? (3)



Use randomised rounding to get a single path for each demand.

Why are weighted graphs hard? (3)

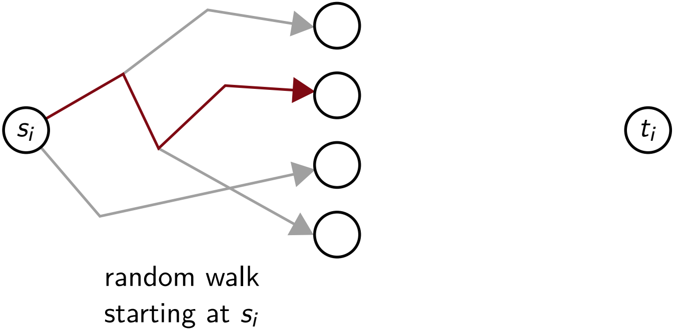


Same problem we had earlier!

We cannot store the paths.

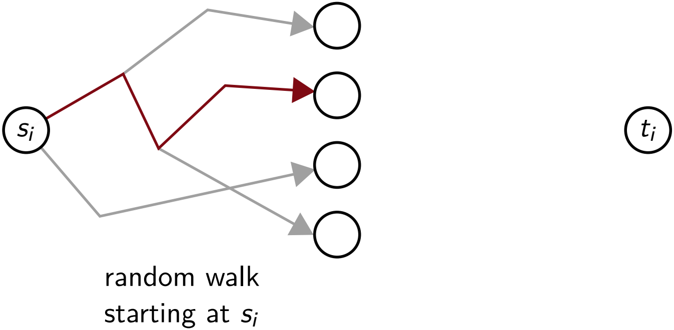
Compact Path Encoding

Solution: random walk + Valiant's trick



Compact Path Encoding

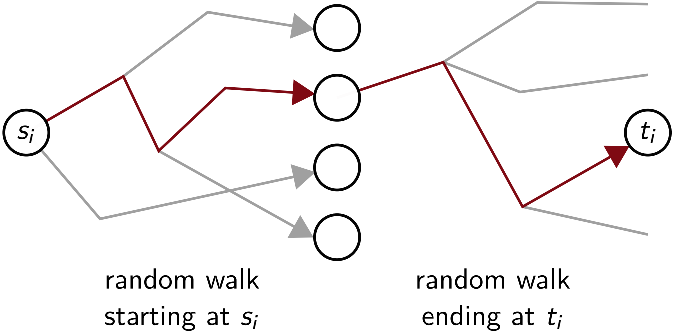
Solution: random walk + Valiant's trick



- do a random walk to (random) intermediate node

Compact Path Encoding

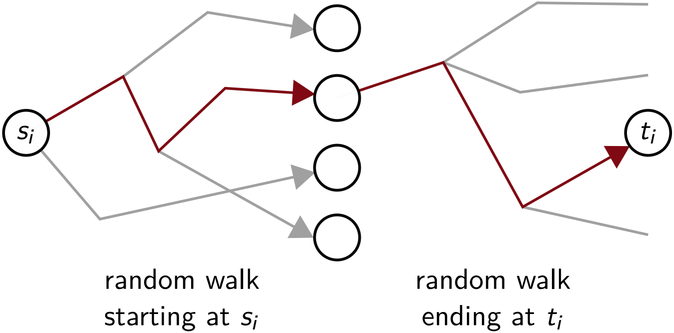
Solution: random walk + Valiant's trick



- do a random walk to (random) intermediate node
- then do random walk, but condition on ending up at the target

Compact Path Encoding

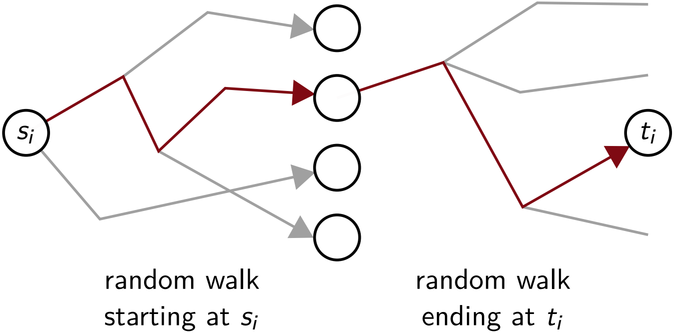
Solution: random walk + Valiant's trick



- do a random walk to (random) intermediate node
- then do random walk, but condition on ending up at the target
- this is Valiant's trick, so good congestion

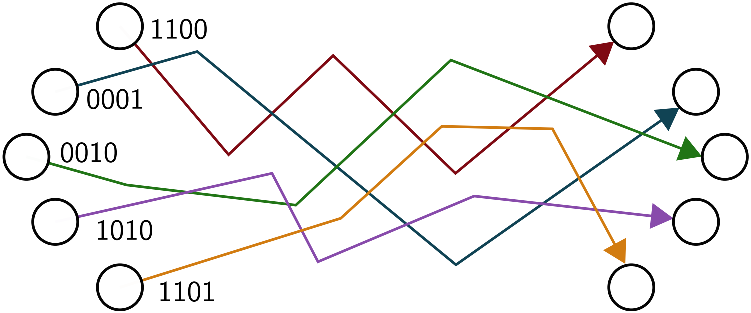
Compact Path Encoding

Solution: random walk + Valiant's trick



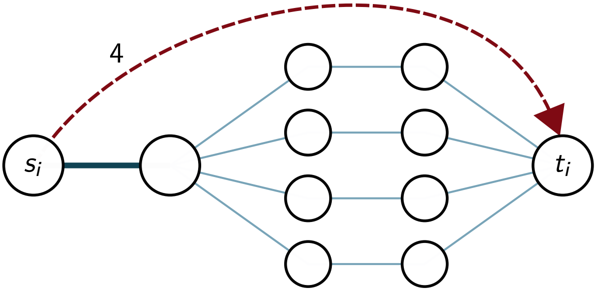
- do a random walk to (random) intermediate node
- then do random walk, but condition on ending up at the target
- this is Valiant's trick, so good congestion
- only need $\tilde{\mathcal{O}}(1)$ coin-flips to store the path!

Compact Path Encoding



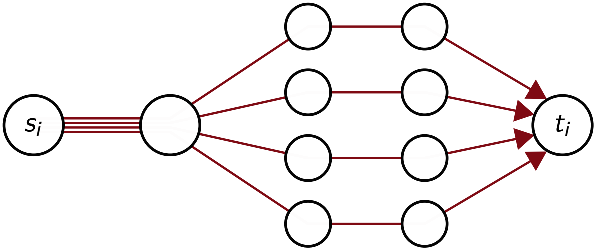
Now we can store the path information at the source.

Why are weighted graphs hard? (4)



This does not work for hypercubes with large weight!

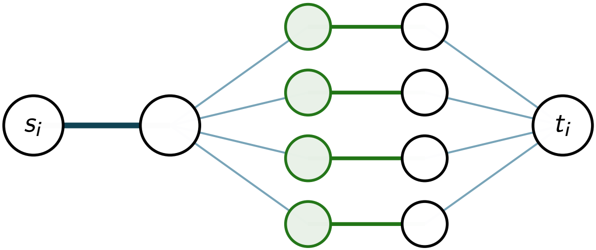
Why are weighted graphs hard? (4)



This does not work for hypercubes with large weight!

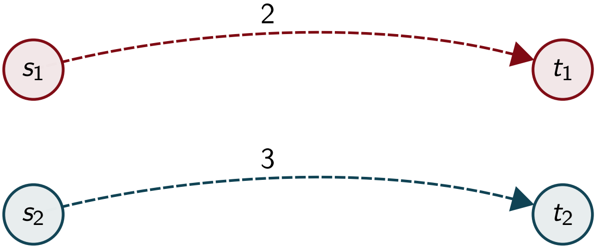
s_i would have to store too many paths.

Distribute Routing Information



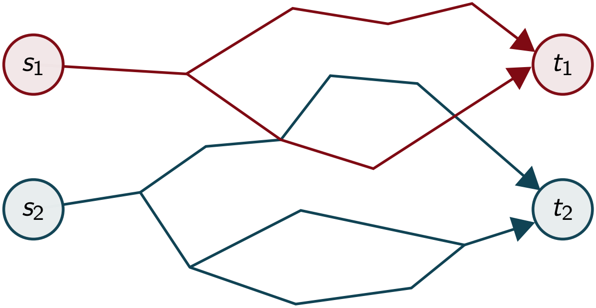
Idea: This happens only if there are many small edges between s_i and t_i $\rightarrow$ We can use adjacent nodes for storage!

Distribute Routing Information



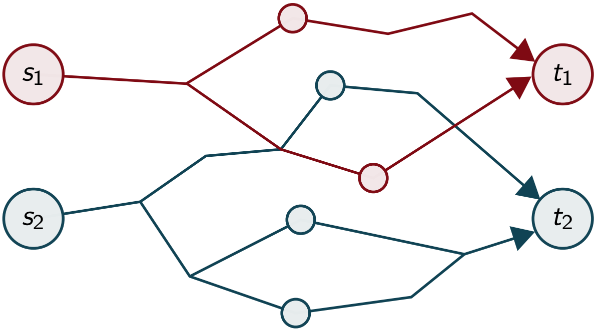
- construct paths

Distribute Routing Information



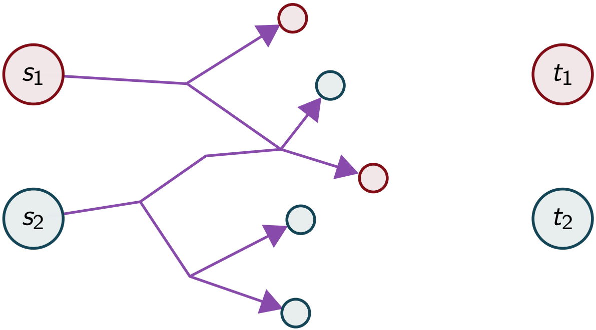
- construct paths
- identify intermediate nodes

Distribute Routing Information



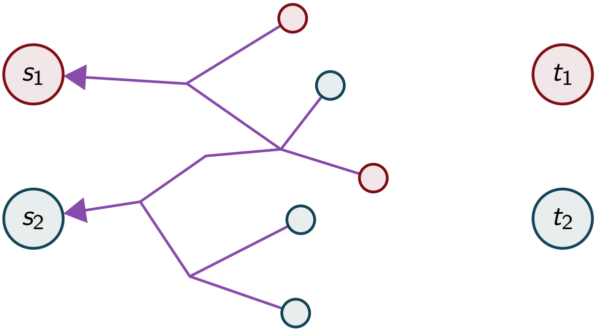
- construct paths
- identify intermediate nodes
- group demands

Distribute Routing Information



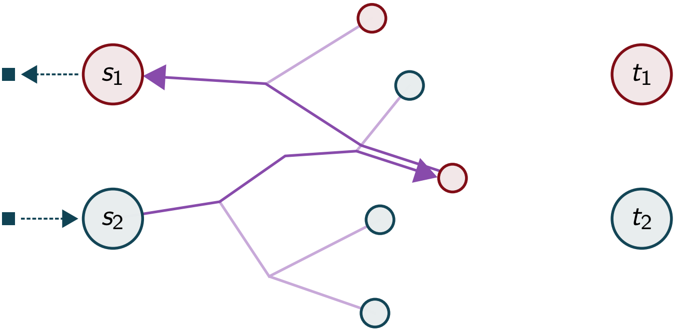
- construct paths
- identify intermediate nodes
- group demands
- route a single-commodity flow

Distribute Routing Information



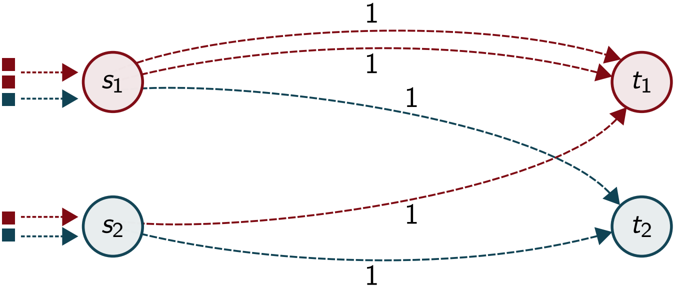
- route backwards

Distribute Routing Information



- route backwards
- packets might get mixed up

Distribute Routing Information



- route backwards
- packets might get mixed up
- new demands
  - smaller
  - use storage of intermediate nodes

Thank you for
your attention!