Efficient Algorithms for Geometric Partial Matching

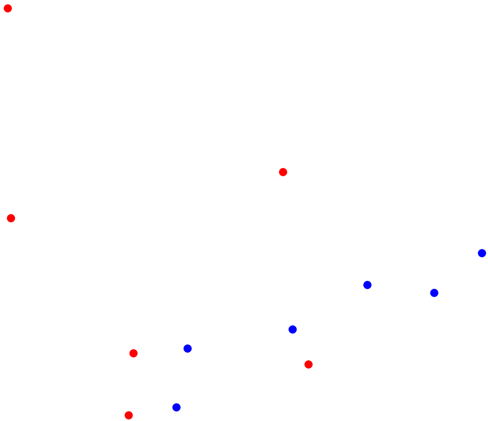
Pankaj K. Agarwal Hsien-Chih Chang Allen Xiao

Department of Computer Science, Duke University

June 2019

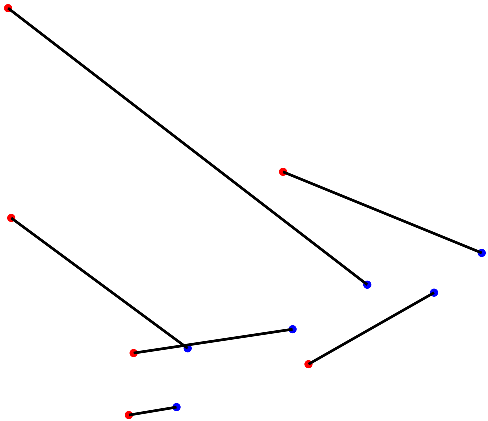
Geometric (bipartite) matching

A, B



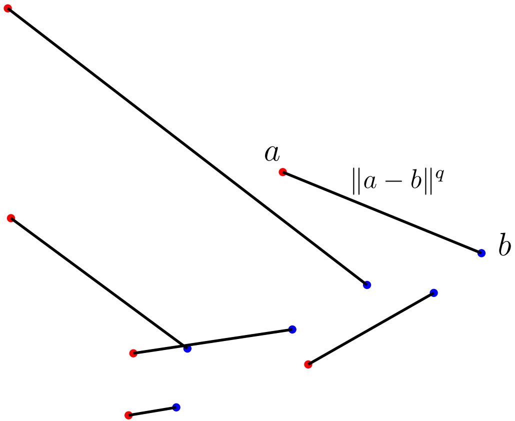
Geometric (bipartite) matching

A, B



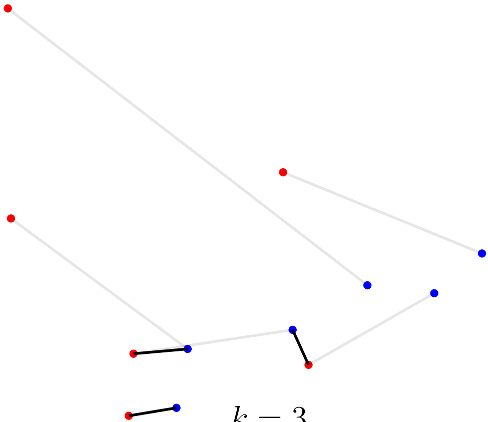
Geometric (bipartite) matching

A, B



Geometric (bipartite) partial matching

A, B



$k = 3$

Prior work and our results

	approx.	time	valid q
Hungarian algorithm (Kuhn)	exact	$O(km + k^2 \log n)$	$q \geq 1$
		$O(kn \operatorname{polylog} n)$	
Ramshaw, Tarjan 2012	exact[1]	$O(m\sqrt{k} \log(kC))$	$q \geq 1$
	$(1+\varepsilon)$	$O(n\sqrt{k} \operatorname{polylog} n \log(1/\varepsilon))$	
Sharathkumar, Agarwal 2012	$(1+\varepsilon)$	$O(n \operatorname{poly}(\log n, 1/\varepsilon))$	$q = 1$
new (Hungarian)	exact	$O((n + k^2) \operatorname{polylog} n)$	$q \geq 1$
new (cost-scaling)	$(1+\varepsilon)$	$O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$	$q \geq 1$

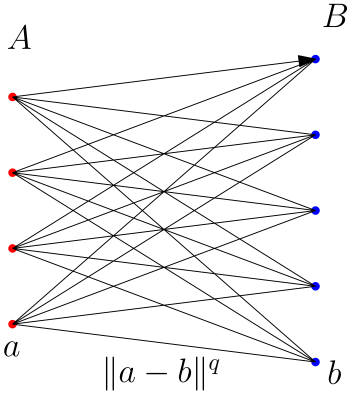
[1] Assuming integer costs $\leq C$.

Prior work and our results

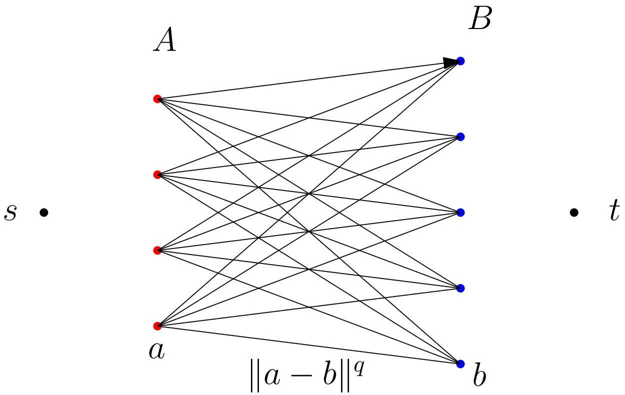
	approx.	time	valid q
Hungarian algorithm (Kuhn)	exact	$O(km + k^2 \log n)$	$q \geq 1$
		$O(kn \operatorname{polylog} n)$	
Ramshaw, Tarjan 2012	exact[1]	$O(m\sqrt{k} \log(kC))$	$q \geq 1$
	$(1+\varepsilon)$	$O(n\sqrt{k} \operatorname{polylog} n \log(1/\varepsilon))$	
Sharathkumar, Agarwal 2012	$(1+\varepsilon)$	$O(n \operatorname{poly}(\log n, 1/\varepsilon))$	$q = 1$
new (Hungarian)	exact	$O((n + k^2) \operatorname{polylog} n)$	$q \geq 1$
new (cost-scaling)	$(1+\varepsilon)$	$O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$	$q \geq 1$

[1] Assuming integer costs $\leq C$.

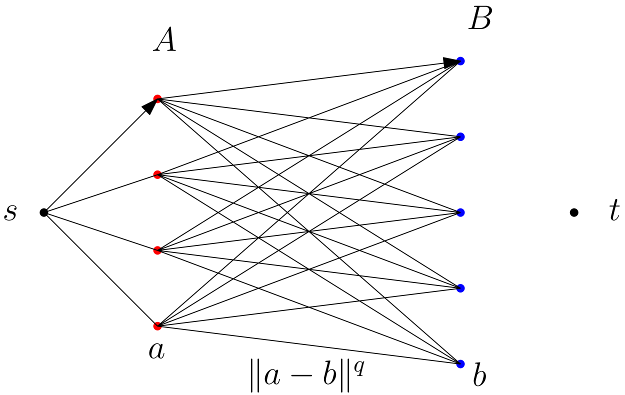
Unit-capacity min-cost flow formulation



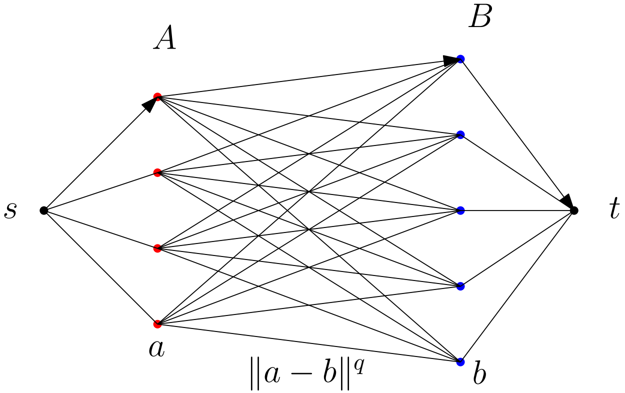
Unit-capacity min-cost flow formulation



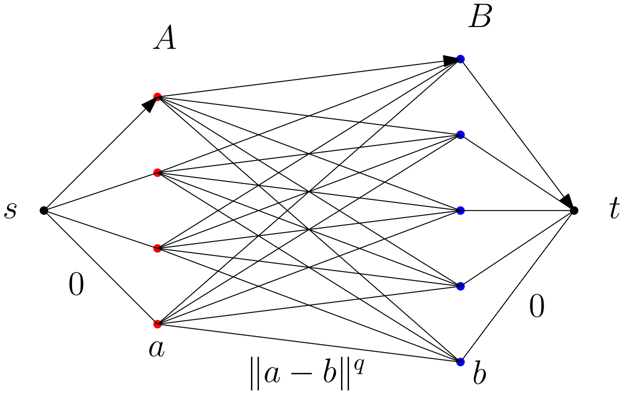
Unit-capacity min-cost flow formulation



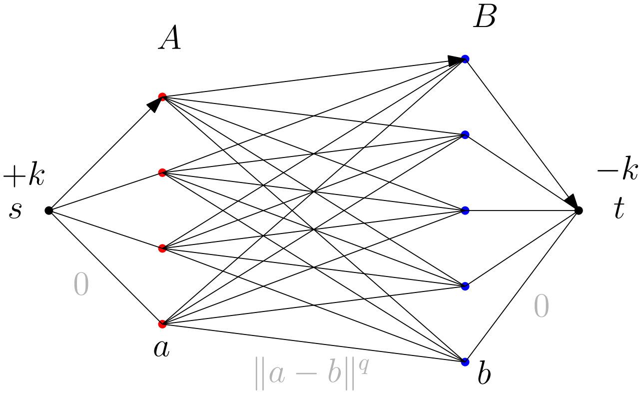
Unit-capacity min-cost flow formulation



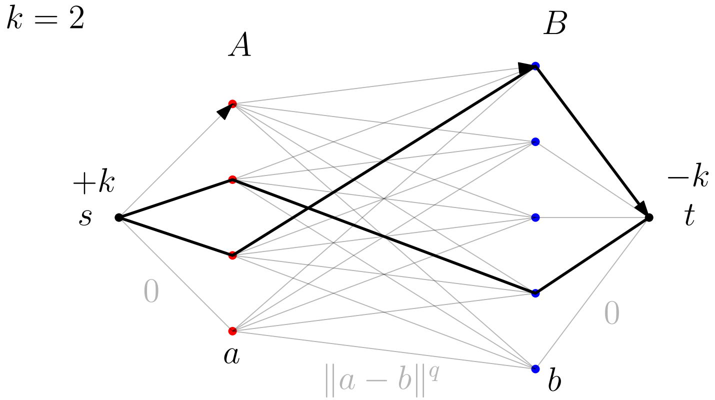
Unit-capacity min-cost flow formulation



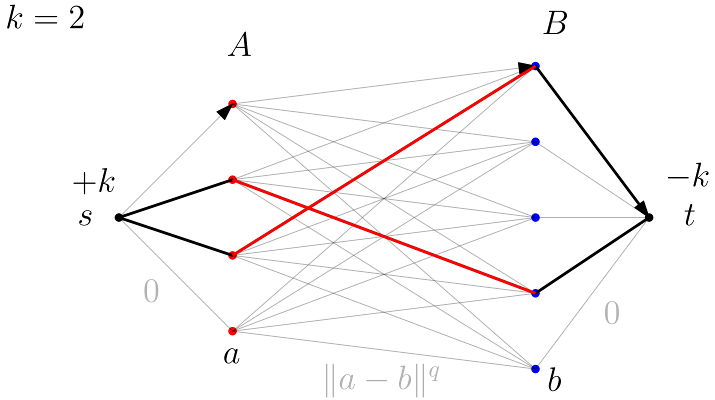
Unit-capacity min-cost flow formulation



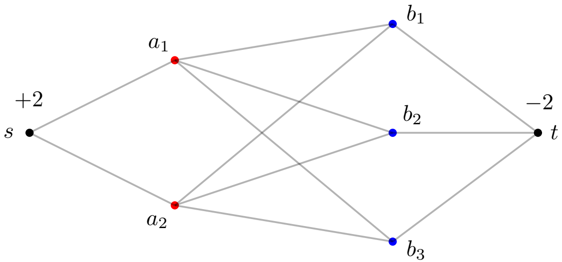
Unit-capacity min-cost flow formulation



Unit-capacity min-cost flow formulation

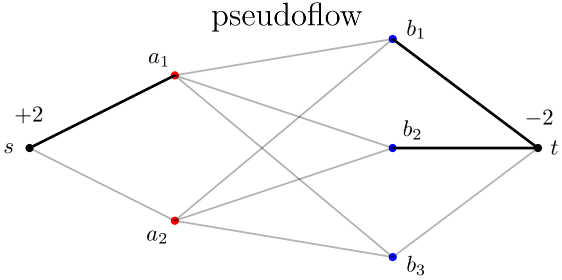


Flow terminology



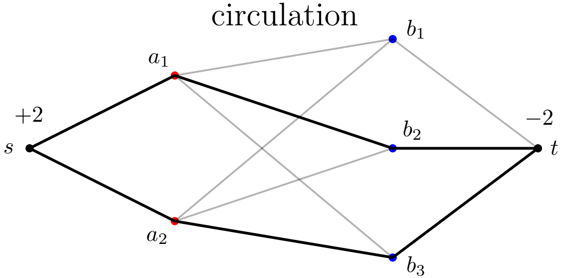
- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$
- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- admissible residual arcs: $c_\pi(v \to w) \leq 0$

Flow terminology



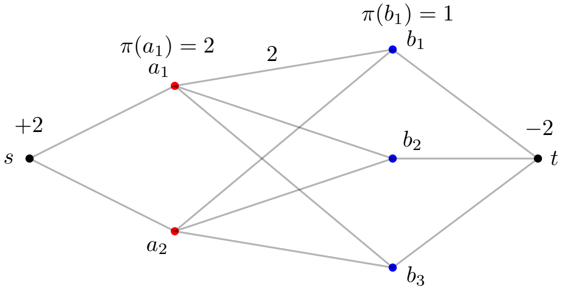
- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$
- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- admissible residual arcs: $c_\pi(v \to w) \leq 0$

Flow terminology



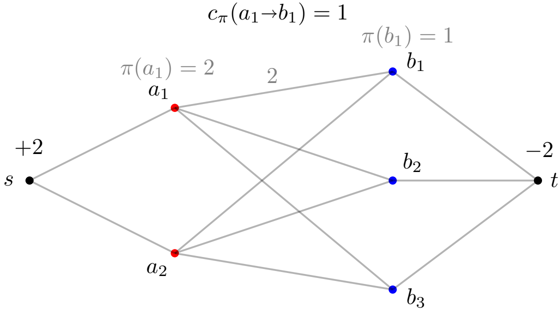
- Reduced cost: $c_\pi(v{\to}w) \coloneqq c(v{\to}w) - \pi(v) + \pi(w)$
- θ-optimality: $c_\pi(v{\to}w) \geq -\theta$ on all residual arcs
- admissible residual arcs: $c_\pi(v{\to}w) \leq 0$

Flow terminology

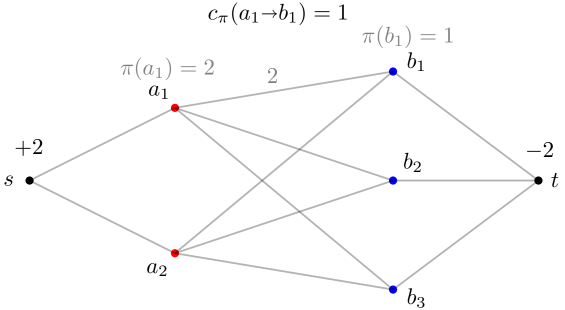


- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$
- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- admissible residual arcs: $c_\pi(v \to w) \leq 0$

Flow terminology



- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$
- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- admissible residual arcs: $c_\pi(v \to w) \leq 0$



- Reduced cost: $c_\pi(v \to w) \coloneqq c(v \to w) - \pi(v) + \pi(w)$
- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- admissible residual arcs: $c_\pi(v \to w) \leq 0$

Cost-scaling (Ramshaw-Tarjan)

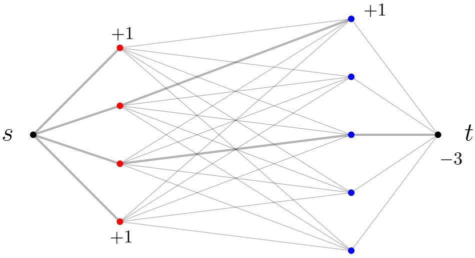
- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- θ-optimal circulation is $+n\theta$ approx. in general ($+6k\theta$ in our graph).
- Find θ-optimal circulations for geometrically decreasing values of θ:
  1. Reduce $\theta \leftarrow \theta/2$, while creating $O(k)$ excess.
  2. Refine this pseudoflow into a circulation, while preserving θ-optimality
- Reduction: $(1+\varepsilon)$-approx. geometric partial matching by executing $O(\log(n^q/\varepsilon))$ cost scales.

Cost-scaling (Ramshaw-Tarjan)

- θ-optimality: $c_\pi(v \to w) \geq -\theta$ on all residual arcs
- θ-optimal circulation is $+n\theta$ approx. in general ($+6k\theta$ in our graph).
- Find θ-optimal circulations for geometrically decreasing values of θ:
  1. Reduce $\theta \leftarrow \theta/2$, while creating $O(k)$ excess.
  2. Refine this pseudoflow into a circulation, while preserving θ-optimality
- Reduction: $(1+\varepsilon)$-approx. geometric partial matching by executing $O(\log(n^q/\varepsilon))$ cost scales.

Refinement by blocking flows (Ramshaw-Tarjan)

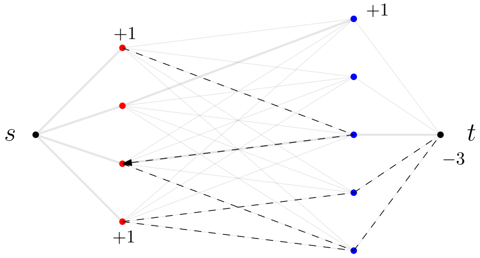
- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.
  3. Repeat until 0 excess.



- $O(\sqrt{k})$ blocking flows before f becomes a circulation.

Refinement by blocking flows (Ramshaw-Tarjan)

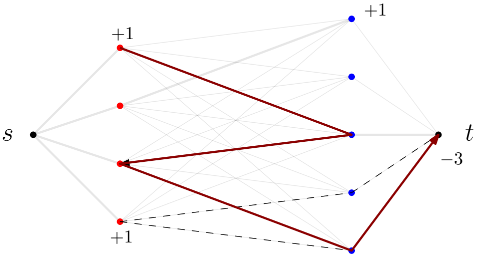
- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.
  3. Repeat until 0 excess.



- $O(\sqrt{k})$ blocking flows before f becomes a circulation.

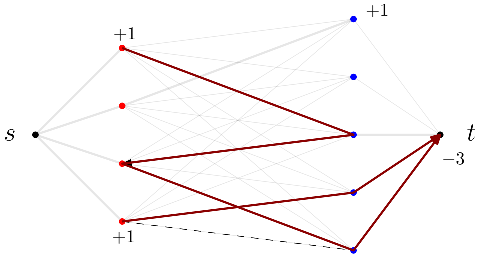
Refinement by blocking flows (Ramshaw-Tarjan)

- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.
  3. Repeat until 0 excess.



- $O(\sqrt{k})$ blocking flows before f becomes a circulation.

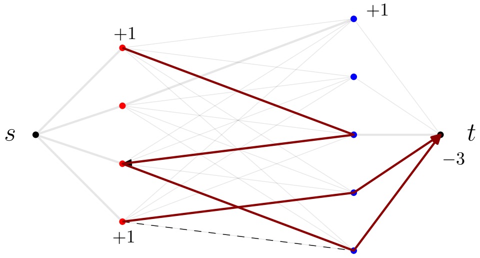
- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.
  3. Repeat until 0 excess.



- $O(\sqrt{k})$ blocking flows before f becomes a circulation.

Refinement by blocking flows (Ramshaw-Tarjan)

- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.
  3. Repeat until 0 excess.



- $O(\sqrt{k})$ blocking flows before f becomes a circulation.

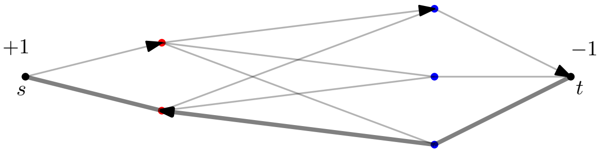
High-level goal per scale

- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.
  3. Repeat until 0 excess.
- After $O(n \operatorname{polylog} n)$-time preprocessing, perform Hungarian search and find each blocking flow in $O(k \operatorname{polylog} n)$ time.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$

X: admissible reachable from an excess node



- Dynamic 2D bichromatic closest pair with additive weights:

$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

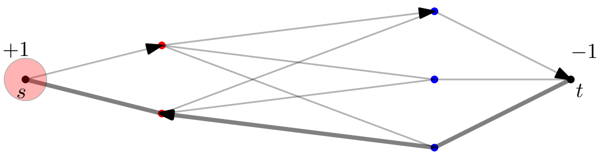
in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v{\to}w) \coloneqq c(v{\to}w) - \pi(v) + \pi(w)$

X: admissible reachable from an excess node



- Dynamic 2D bichromatic closest pair with additive weights:

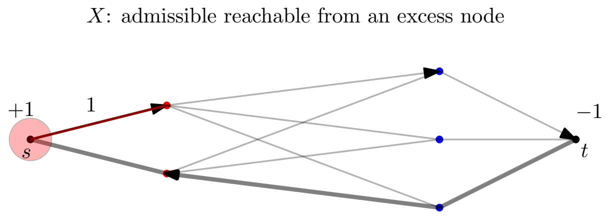
$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v \to w) \coloneqq c(v \to w) - \pi(v) + \pi(w)$



- Dynamic 2D bichromatic closest pair with additive weights:

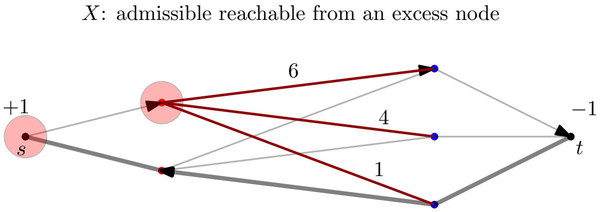
$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\text{polylog}\, n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v \to w) \coloneqq c(v \to w) - \pi(v) + \pi(w)$



- Dynamic 2D bichromatic closest pair with additive weights:

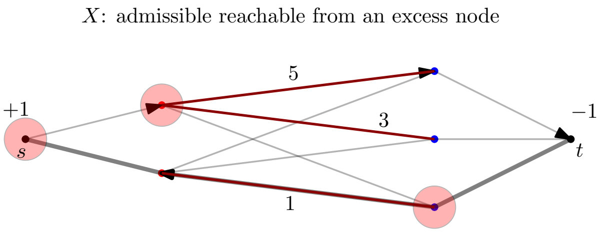
$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$



- Dynamic 2D bichromatic closest pair with additive weights:

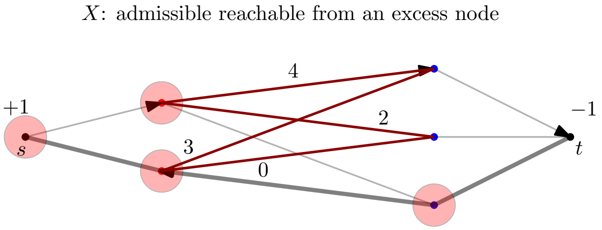
$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\text{polylog}\, n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$



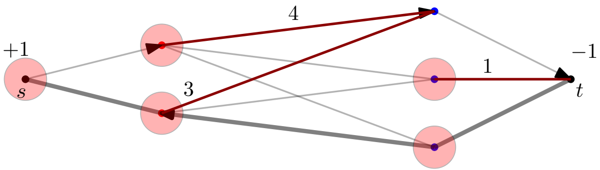
- Dynamic 2D bichromatic closest pair with additive weights:

$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

- Reduced cost: $c_\pi(v{\to}w) \coloneqq c(v{\to}w) - \pi(v) + \pi(w)$



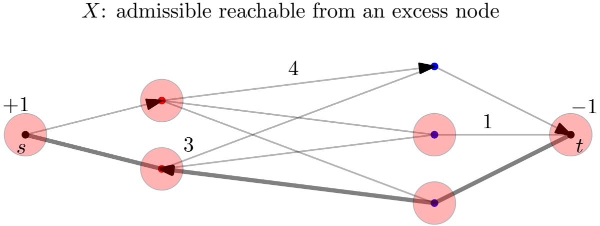
- Dynamic 2D bichromatic closest pair with additive weights:

$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

  in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)
- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$



- Dynamic 2D bichromatic closest pair with additive weights:

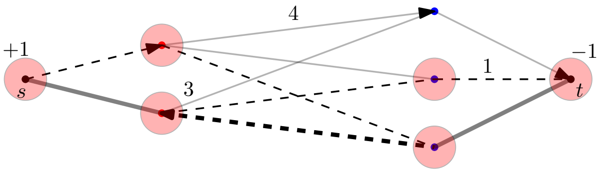
$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

- Reduced cost: $c_\pi(v \to w) := c(v \to w) - \pi(v) + \pi(w)$

X: admissible reachable from an excess node



- Dynamic 2D bichromatic closest pair with additive weights:

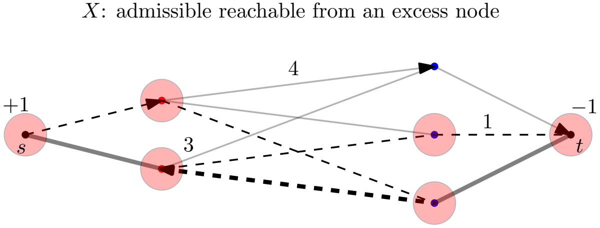
$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

in $O(\operatorname{polylog} n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

Hungarian search with BCP (Agarwal-Efrat-Sharir)

- Reduced cost: $c_\pi(v{\to}w) \coloneqq c(v{\to}w) - \pi(v) + \pi(w)$



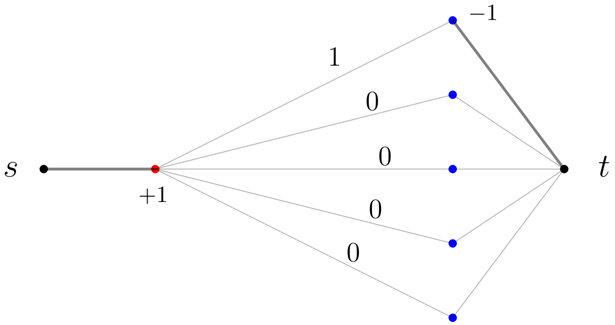
- Dynamic 2D bichromatic closest pair with additive weights:

$$\min_{S_1 \times S_2} \|p_1 - p_2\|^q + \omega(p_1) + \omega(p_2)$$

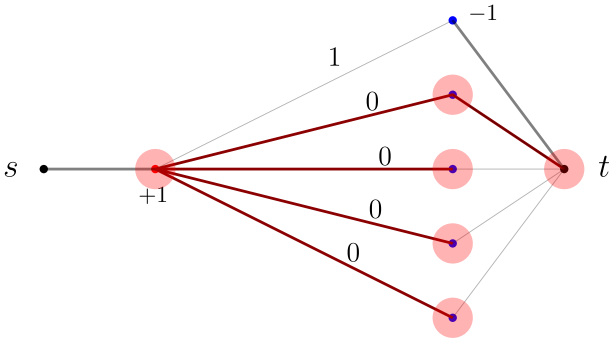
in $O(\text{polylog}\, n)$ update, $O(\log^2 n)$ query (Kaplan *et al.* SODA'17)

- Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

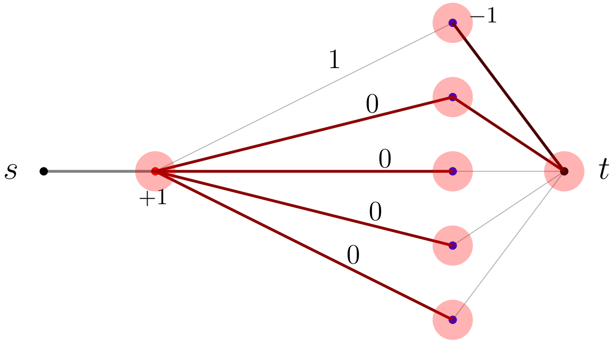
Problem: Hungarian search looks at too many nodes



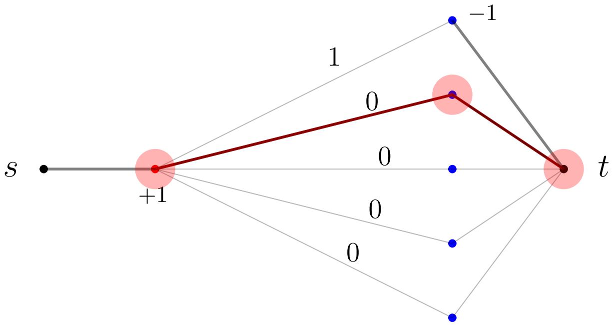
Problem: Hungarian search looks at too many nodes



Problem: Hungarian search looks at too many nodes

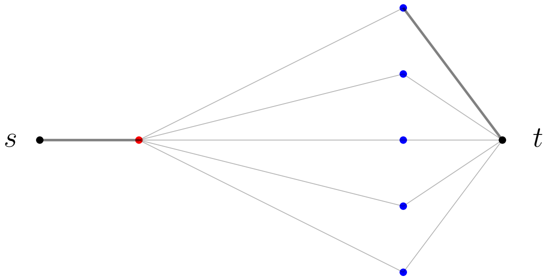


Problem: Hungarian search looks at too many nodes



Dead/alive nodes

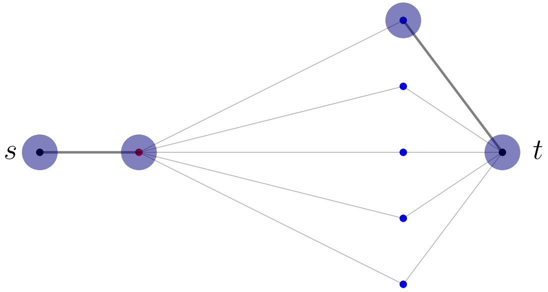
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.

Dead/alive nodes

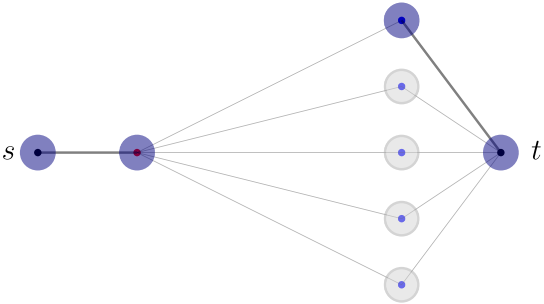
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.

Dead/alive nodes

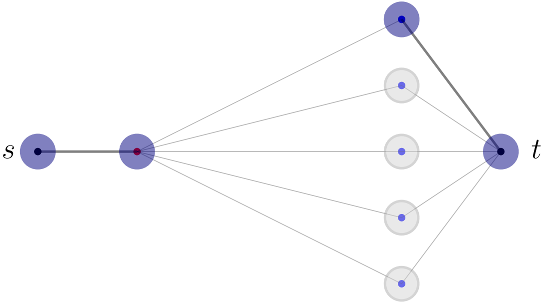
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.

Dead/alive nodes

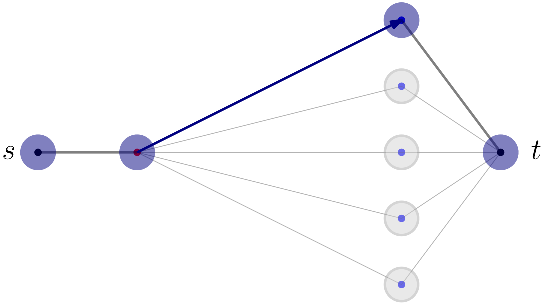
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.

Dead/alive nodes

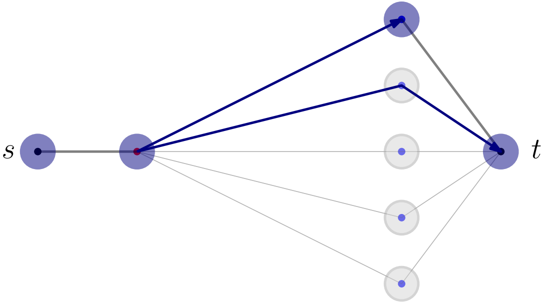
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.

Dead/alive nodes

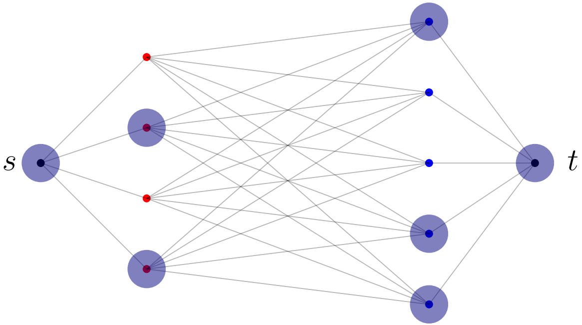
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.

Relaxing alive paths

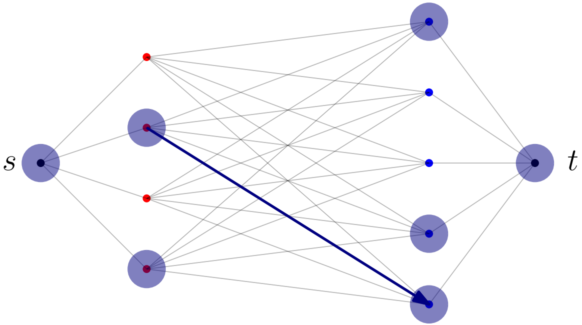
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

Relaxing alive paths

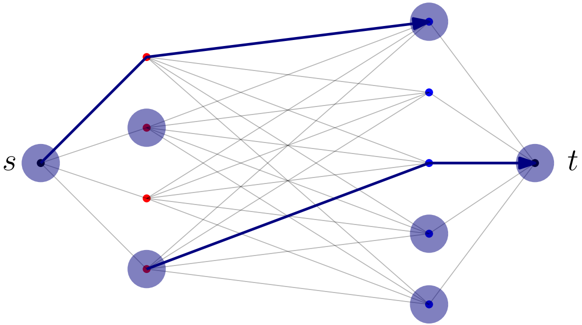
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

Relaxing alive paths

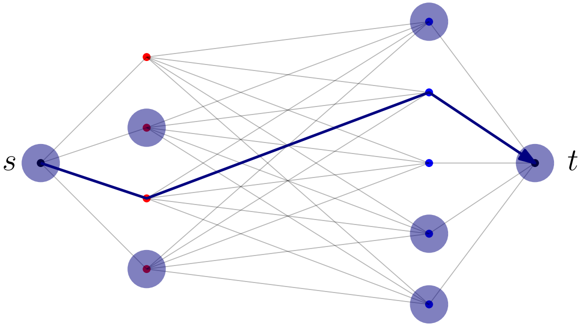
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

Relaxing alive paths

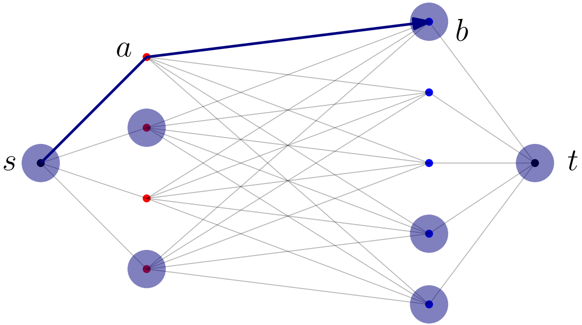
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

Relaxing alive paths

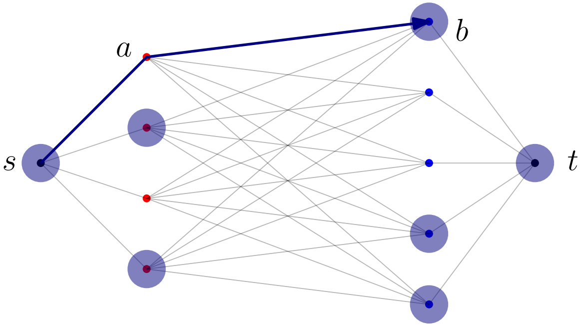
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

Relaxing alive paths

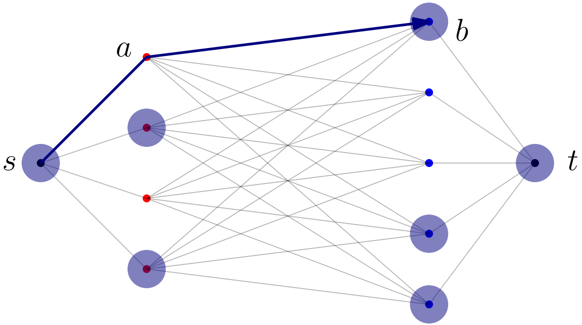
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

Relaxing alive paths

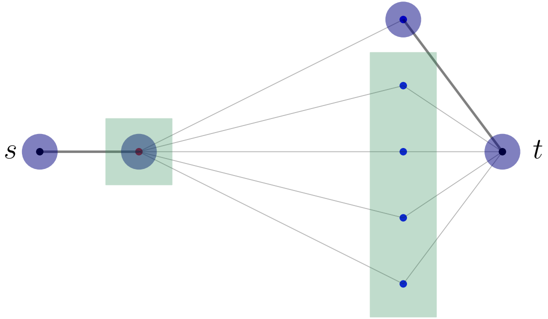
- Alive paths have length 1, 2, or 3.



- Telescoping: $c_\pi(s \to a \to b) = c(a \to b) - \pi(s) + \pi(b)$ (use BCP)
- Only $O(k)$ relaxations per Hungarian search.
- Also: find a blocking flow in $O(k)$ relaxations (DFS).

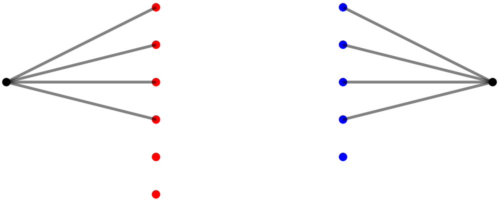
Problem: BCP initialization

- Dynamic 2D BCP with $O(\text{polylog}\, n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)



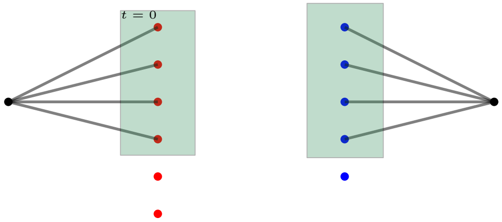
- Some BCP may begin a Hungarian search with $\Theta(n)$ vertices.
- Can't afford to construct from scratch for every Hungarian search.

Initial BCP by rewinding



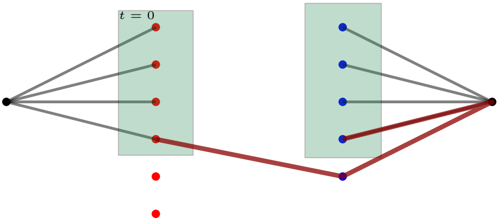
- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
  1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
  2. Apply the few changes (newly matched, dead/alive).
- Persistence?

Initial BCP by rewinding



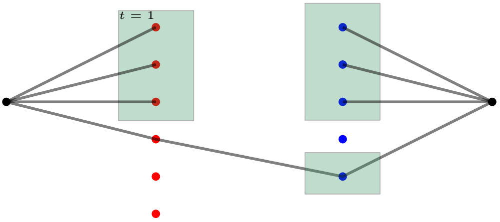
- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
  1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
  2. Apply the few changes (newly matched, dead/alive).
- Persistence?

Initial BCP by rewinding

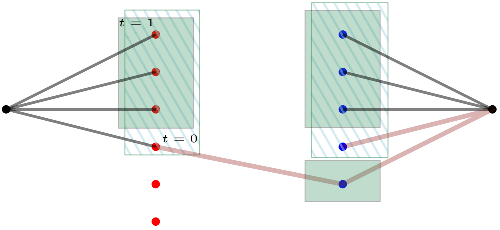


- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
    1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
    2. Apply the few changes (newly matched, dead/alive).
- Persistence?

Initial BCP by rewinding

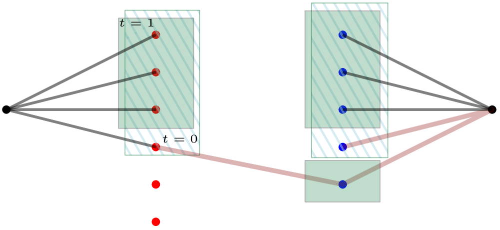


- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
  1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
  2. Apply the few changes (newly matched, dead/alive).
- Persistence?



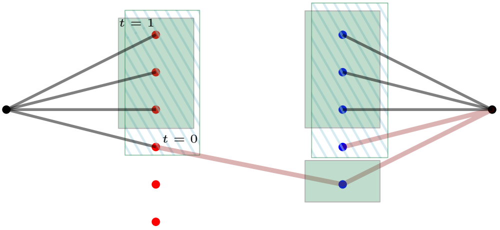
- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
    1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
    2. Apply the few changes (newly matched, dead/alive).
- Persistence?

Initial BCP by rewinding



- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
  1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
  2. Apply the few changes (newly matched, dead/alive).
- Persistence?

Initial BCP by rewinding



- Let $\mathcal{D}_t$ be the BCP at the start of the t-th Hungarian search.
- $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ differ by only a few nodes.
- To generate $\mathcal{D}_{t+1}$:
  1. Rewind the BCP updates of the last Hungarian search to obtain $\mathcal{D}_t$.
  2. Apply the few changes (newly matched, dead/alive).
- Persistence?

Summary

- $O(\log(n^q/\varepsilon))$ scales, $O(\sqrt{k})$ blocking flows per scale.
- Each blocking flow's Hungarian search uses $O(k)$ relaxations (alive paths).
- Each blocking flow's Hungarian search BCP can be initialized using the previous one in $O(k \operatorname{polylog} n)$ time (rewinding). Need to spend $O(n \operatorname{polylog} n)$ once per scale to build data structures for $t = 0$.
- $O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$
- Thank you.

Summary

- $O(\log(n^q/\varepsilon))$ scales, $O(\sqrt{k})$ blocking flows per scale.
- Each blocking flow's Hungarian search uses $O(k)$ relaxations (alive paths).
- Each blocking flow's Hungarian search BCP can be initialized using the previous one in $O(k \operatorname{polylog} n)$ time (rewinding). Need to spend $O(n \operatorname{polylog} n)$ once per scale to build data structures for $t = 0$.
- $O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$
- Thank you.

Summary

- $O(\log(n^q/\varepsilon))$ scales, $O(\sqrt{k})$ blocking flows per scale.
- Each blocking flow's Hungarian search uses $O(k)$ relaxations (alive paths).
- Each blocking flow's Hungarian search BCP can be initialized using the previous one in $O(k \operatorname{polylog} n)$ time (rewinding). Need to spend $O(n \operatorname{polylog} n)$ once per scale to build data structures for $t = 0$.
- $O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$
- Thank you.

Summary

- $O(\log(n^q/\varepsilon))$ scales, $O(\sqrt{k})$ blocking flows per scale.
- Each blocking flow's Hungarian search uses $O(k)$ relaxations (alive paths).
- Each blocking flow's Hungarian search BCP can be initialized using the previous one in $O(k \operatorname{polylog} n)$ time (rewinding). Need to spend $O(n \operatorname{polylog} n)$ once per scale to build data structures for $t = 0$.

- $O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$

- Thank you.

Summary

- $O(\log(n^q/\varepsilon))$ scales, $O(\sqrt{k})$ blocking flows per scale.
- Each blocking flow's Hungarian search uses $O(k)$ relaxations (alive paths).
- Each blocking flow's Hungarian search BCP can be initialized using the previous one in $O(k \operatorname{polylog} n)$ time (rewinding). Need to spend $O(n \operatorname{polylog} n)$ once per scale to build data structures for $t = 0$.
- $O((n + k\sqrt{k}) \operatorname{polylog} n \log(1/\varepsilon))$
- Thank you.