

Efficient Algorithms for Geometric Partial Matching

Pankaj K. Agarwal Hsien-Chih Chang Allen Xiao

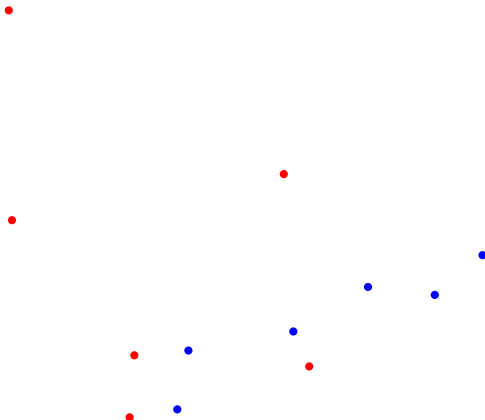
Department of Computer Science, Duke University

June 2019

Geometric (bipartite) matching



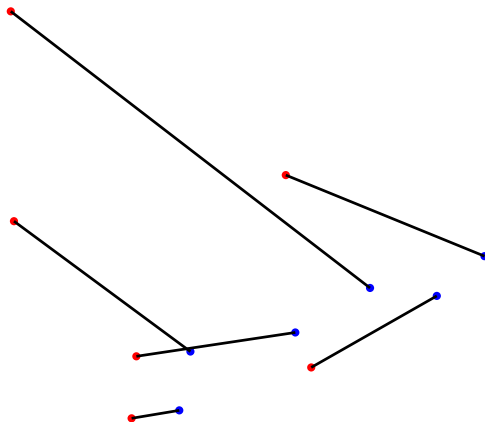
A, B



Geometric (bipartite) matching



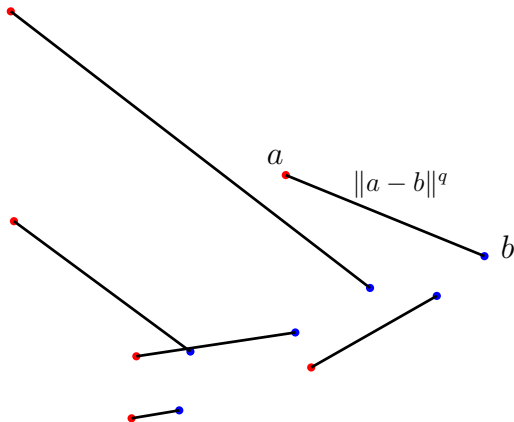
A, B



Geometric (bipartite) matching



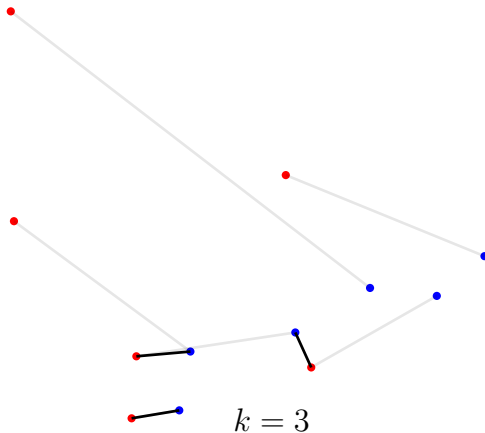
A, B



Geometric (bipartite) partial matching



A, B

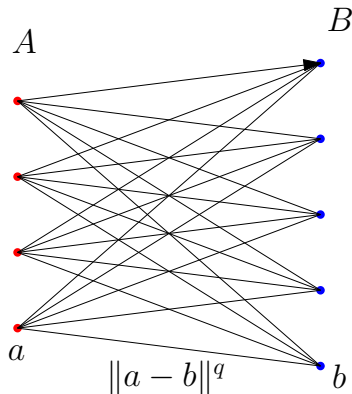




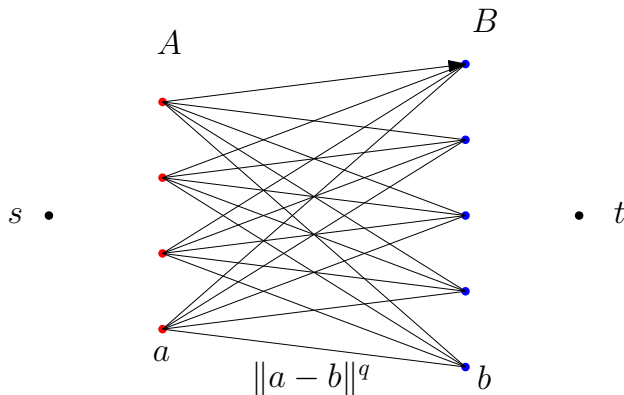
	approx.	time	valid q
Hungarian algorithm (Kuhn)	exact	$O(km + k^2 \log n)$	$q \geq 1$
Ramshaw, Tarjan 2012	exact ¹	$O(kn \text{ polylog } n)$	$q \geq 1$
		$O(m\sqrt{k} \log(kC))$	
	$(1 + \varepsilon)$	$O(n\sqrt{k} \text{ polylog } n \log(1/\varepsilon))$	
Sharathkumar, Agarwal 2012	$(1 + \varepsilon)$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

¹Assuming integer costs $\leq C$.

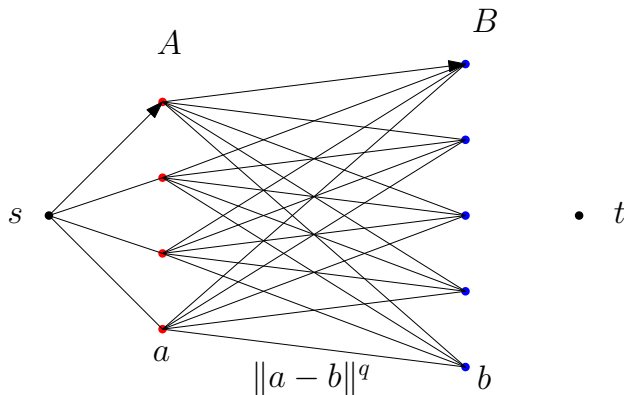
Unit-capacity min-cost flow formulation



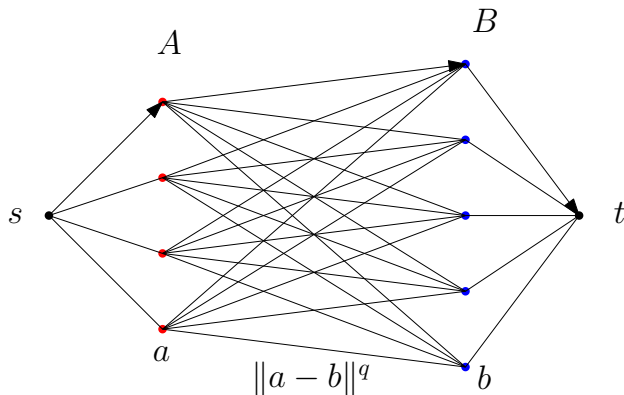
Unit-capacity min-cost flow formulation



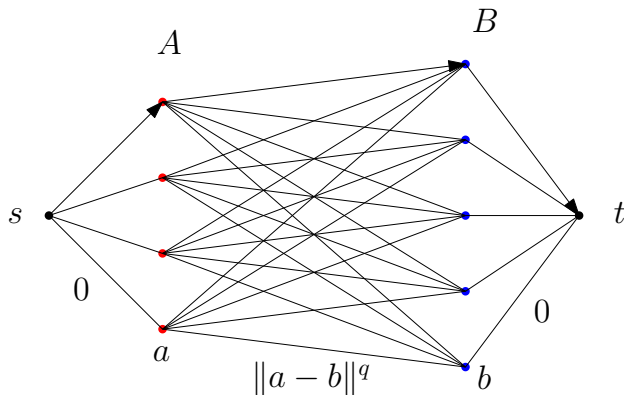
Unit-capacity min-cost flow formulation



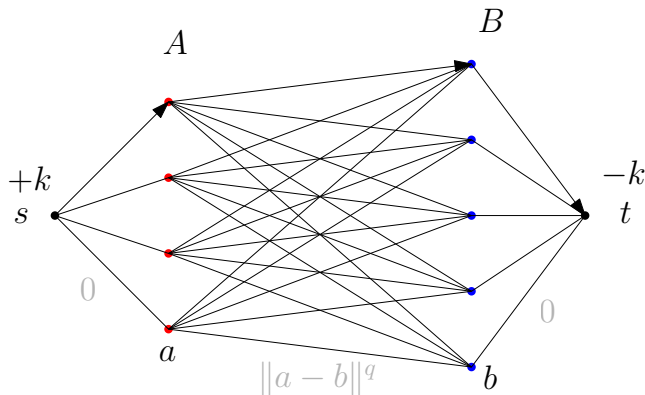
Unit-capacity min-cost flow formulation

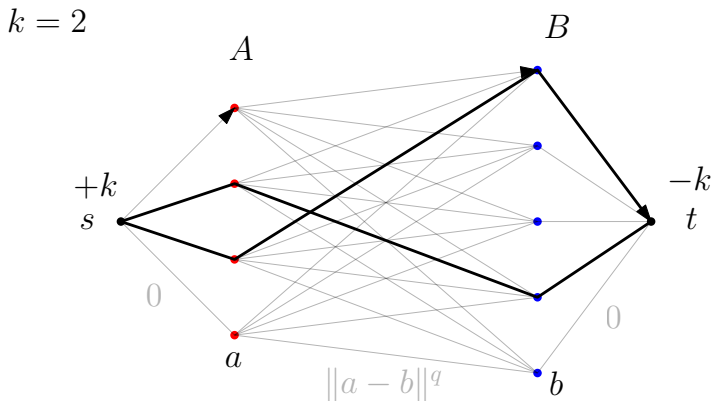


Unit-capacity min-cost flow formulation

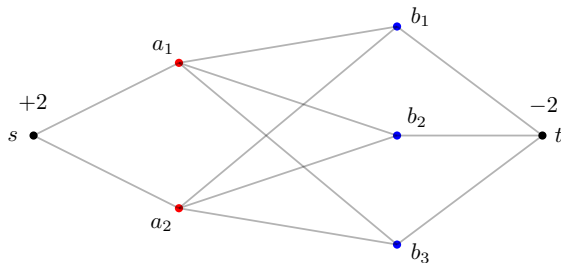


Unit-capacity min-cost flow formulation

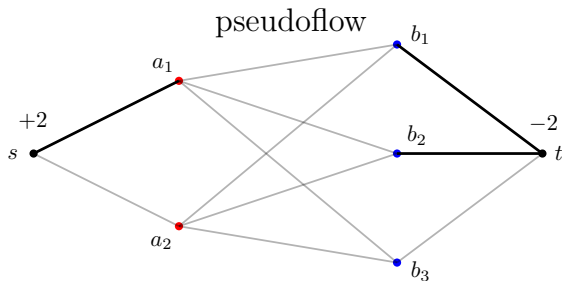




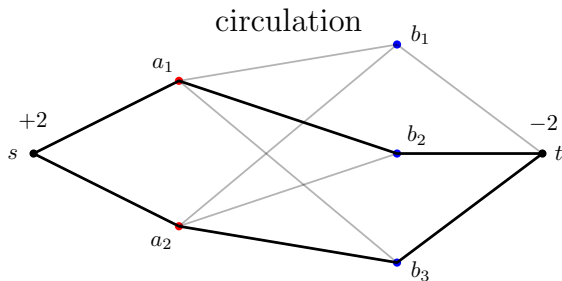




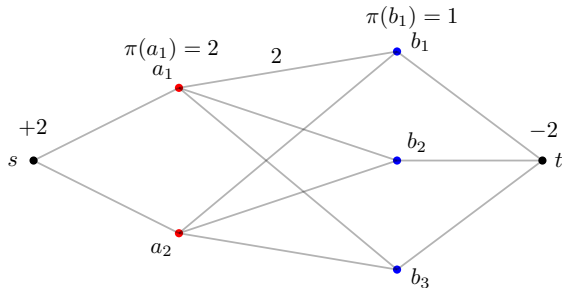
- ▶ Reduced cost: $c_\pi(v \rightarrow w) := c(v \rightarrow w) - \pi(v) + \pi(w)$
- ▶ θ -optimality: $c_\pi(v \rightarrow w) \geq -\theta$ on all residual arcs
- ▶ admissible residual arcs: $c_\pi(v \rightarrow w) \leq 0$



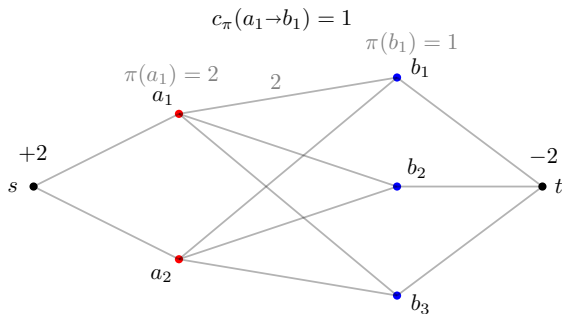
- ▶ Reduced cost: $c_{\pi}(v \rightarrow w) := c(v \rightarrow w) - \pi(v) + \pi(w)$
- ▶ θ -optimality: $c_{\pi}(v \rightarrow w) \geq -\theta$ on all residual arcs
- ▶ admissible residual arcs: $c_{\pi}(v \rightarrow w) \leq 0$



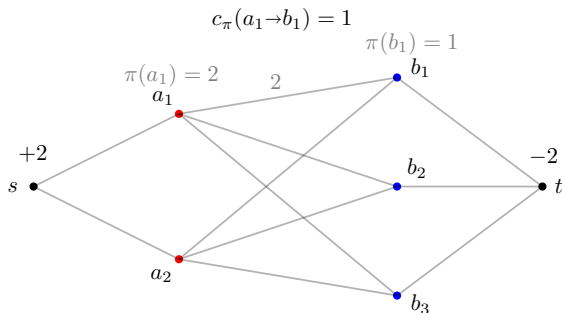
- ▶ Reduced cost: $c_{\pi}(v \rightarrow w) := c(v \rightarrow w) - \pi(v) + \pi(w)$
- ▶ θ -optimality: $c_{\pi}(v \rightarrow w) \geq -\theta$ on all residual arcs
- ▶ admissible residual arcs: $c_{\pi}(v \rightarrow w) \leq 0$



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



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



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

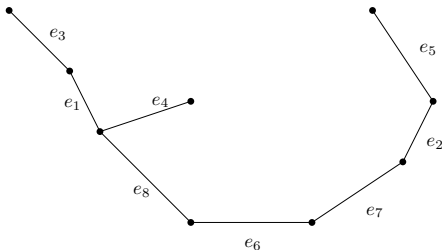


- ▶ **θ -optimality**: $c_\pi(v \rightarrow w) \geq -\theta$ on all residual arcs
- ▶ θ -optimal circulation is $+n\theta$ approx. in general ($+6k\theta$ in our graph).
- ▶ Find θ -optimal circulations for geometrically decreasing values of θ :
 1. Reduce $\theta \leftarrow \theta/2$, while creating $O(k)$ excess.
 2. **Refine** this pseudoflow into a circulation, while preserving θ -optimality



► Compute $\alpha \geq 0$ satisfying:

1. there exists a k -matching whose longest edge has cost $\leq n^q \cdot \alpha$
2. an $(\varepsilon\alpha/6k)$ -optimal circulation is $(1 + \varepsilon)$ -approx.

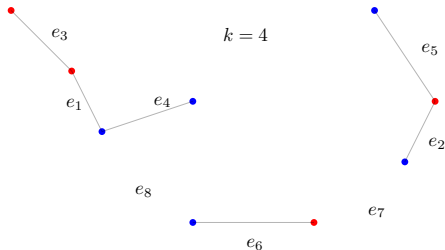


- $(1 + \varepsilon)$ -approx. geometric partial matching by executing $O(\log(n^q/\varepsilon))$ cost scales.



► Compute $\alpha \geq 0$ satisfying:

1. there exists a k -matching whose longest edge has cost $\leq n^q \cdot \alpha$
2. an $(\varepsilon\alpha/6k)$ -optimal circulation is $(1 + \varepsilon)$ -approx.

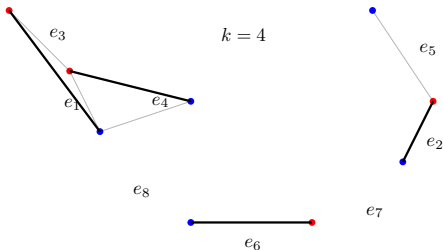


- $(1 + \varepsilon)$ -approx. geometric partial matching by executing $O(\log(n^q/\varepsilon))$ cost scales.



► Compute $\alpha \geq 0$ satisfying:

1. there exists a k -matching whose longest edge has cost $\leq n^q \cdot \alpha$
2. an $(\varepsilon\alpha/6k)$ -optimal circulation is $(1 + \varepsilon)$ -approx.

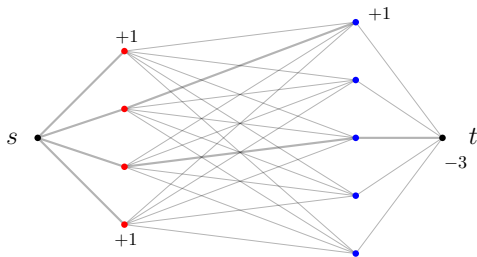


- $(1 + \varepsilon)$ -approx. geometric partial matching by executing $O(\log(n^q/\varepsilon))$ cost scales.



► Inside Refine:

1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
2. Augment by an admissible **blocking flow**.

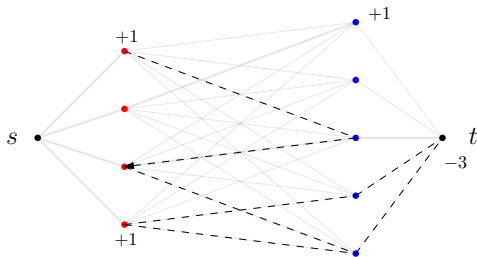


► $O(\sqrt{k})$ blocking flows before f becomes a circulation.



► Inside Refine:

1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
2. Augment by an admissible **blocking flow**.

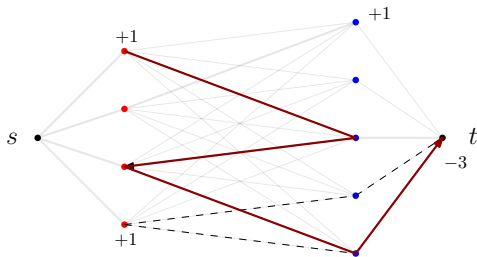


► $O(\sqrt{k})$ blocking flows before f becomes a circulation.



► Inside Refine:

1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
2. Augment by an admissible **blocking flow**.

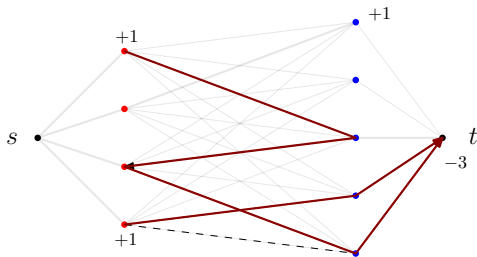


► $O(\sqrt{k})$ blocking flows before f becomes a circulation.



► Inside Refine:

1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
2. Augment by an admissible **blocking flow**.

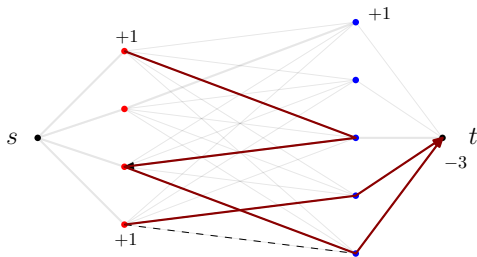


► $O(\sqrt{k})$ blocking flows before f becomes a circulation.



► Inside Refine:

1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
2. Augment by an admissible **blocking flow**.



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



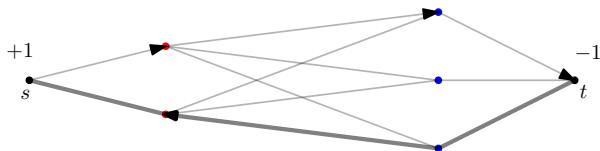
► Inside Refine:

1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
2. Augment by an admissible **blocking flow**.

- After $O(n \text{ polylog } n)$ -time preprocessing, perform Hungarian search and find each blocking flow in $O(k \text{ polylog } n)$ time.



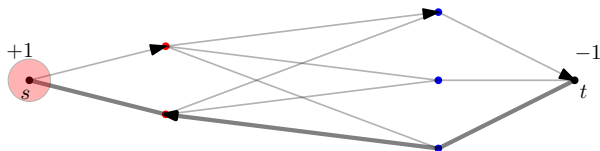
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



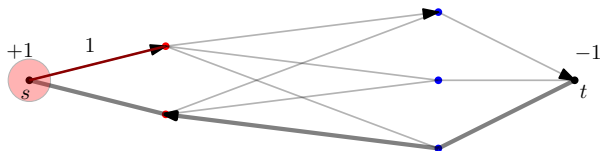
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



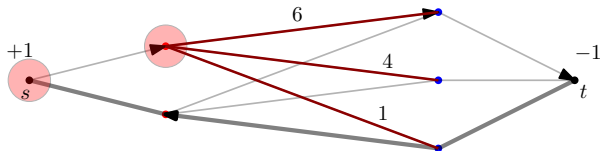
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



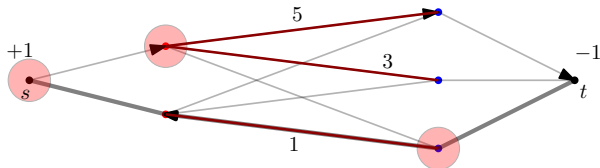
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



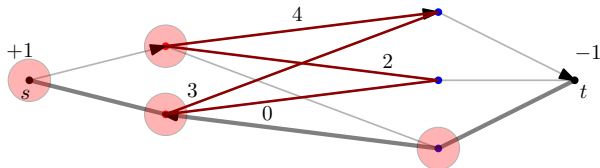
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



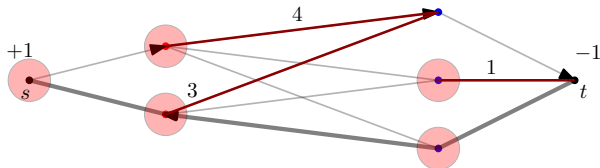
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



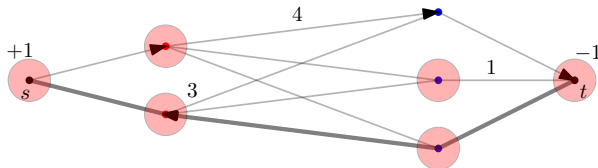
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



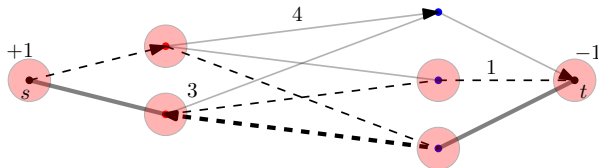
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.



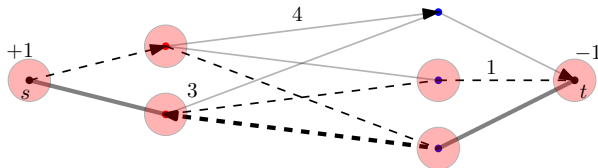
X : admissible reachable from an excess node



- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

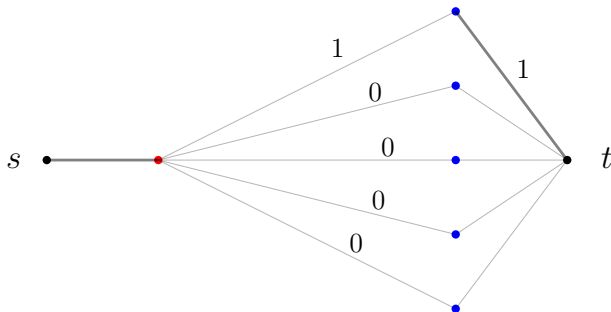


X : admissible reachable from an excess node

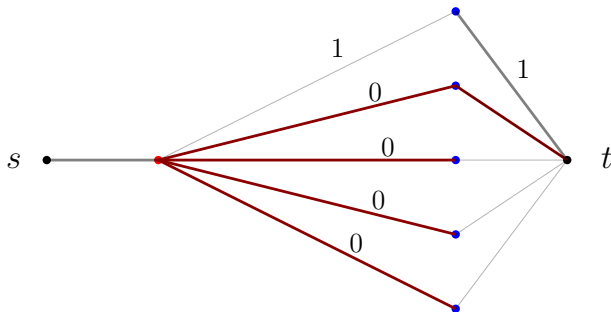


- ▶ Dynamic 2D **bichromatic closest pair** with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)
- ▶ Possible to batch potential updates (Vaidya) — only need to bound number of relaxations.

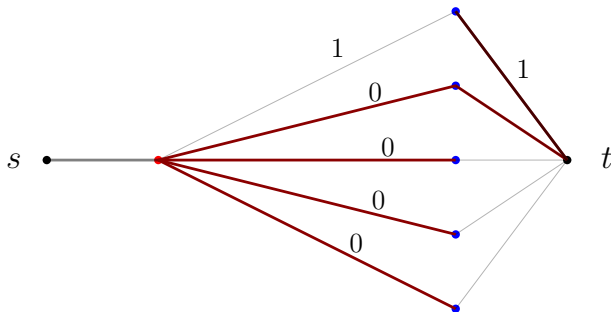
Problem: Hungarian search looks at too many nodes



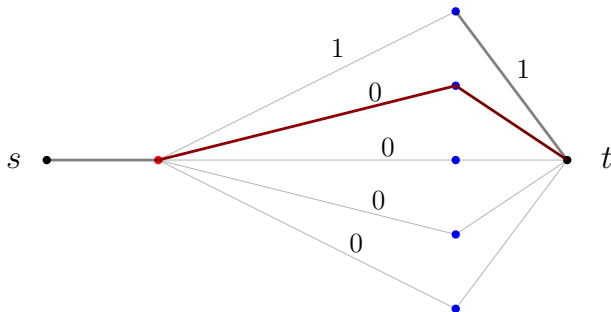
Problem: Hungarian search looks at too many nodes



Problem: Hungarian search looks at too many nodes

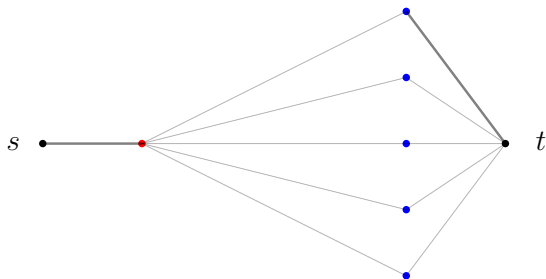


Problem: Hungarian search looks at too many nodes





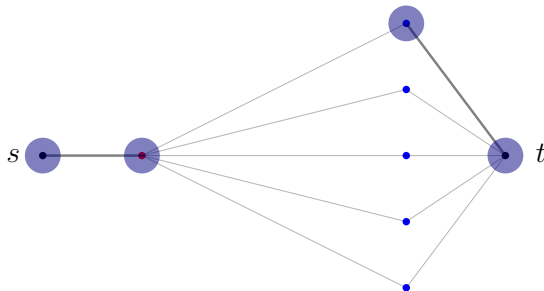
- ▶ **Alive nodes**: nonzero excess/deficit, or adjoining flow support arcs.
- ▶ **Dead nodes**: ones which aren't alive.



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



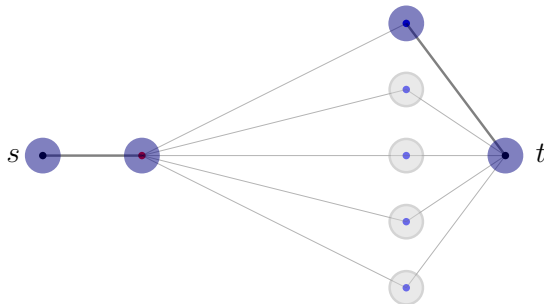
- ▶ **Alive nodes**: nonzero excess/deficit, or adjoining flow support arcs.
- ▶ **Dead nodes**: ones which aren't alive.



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



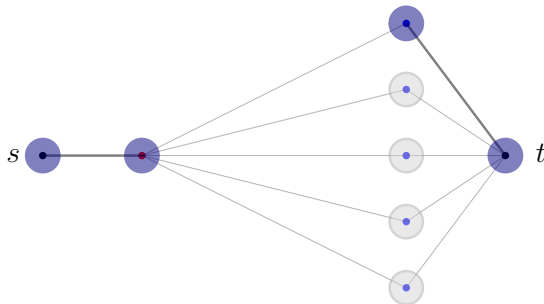
- ▶ **Alive nodes**: nonzero excess/deficit, or adjoining flow support arcs.
- ▶ **Dead nodes**: ones which aren't alive.



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



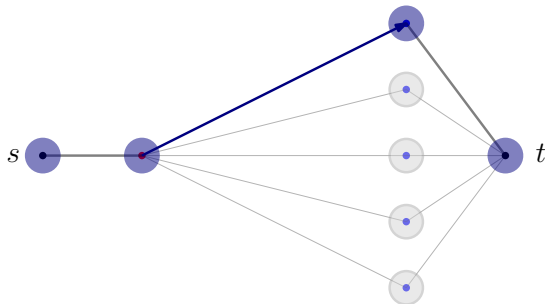
- ▶ **Alive nodes:** nonzero excess/deficit, or adjoining flow support arcs.
- ▶ **Dead nodes:** ones which aren't alive.



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



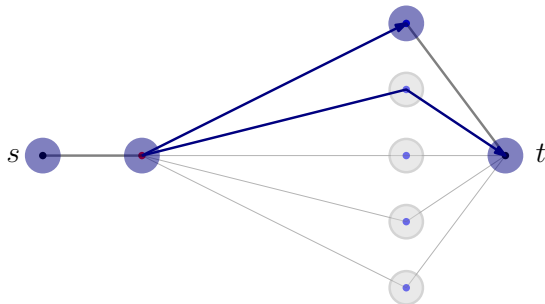
- ▶ **Alive nodes**: nonzero excess/deficit, or adjoining flow support arcs.
- ▶ **Dead nodes**: ones which aren't alive.



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

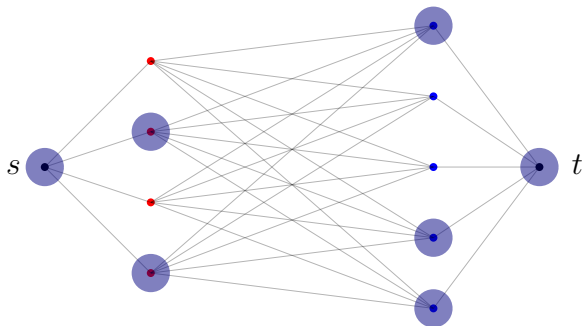


- ▶ **Alive nodes**: nonzero excess/deficit, or adjoining flow support arcs.
- ▶ **Dead nodes**: ones which aren't alive.



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

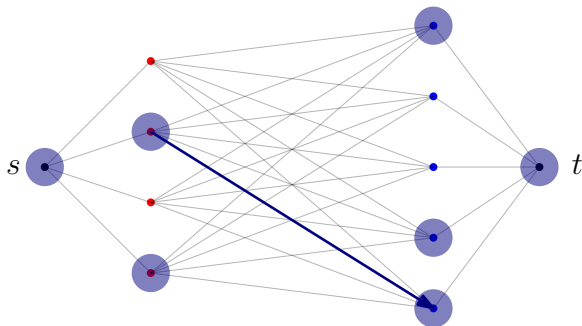
- ▶ Alive paths have length 1, 2, or 3.



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



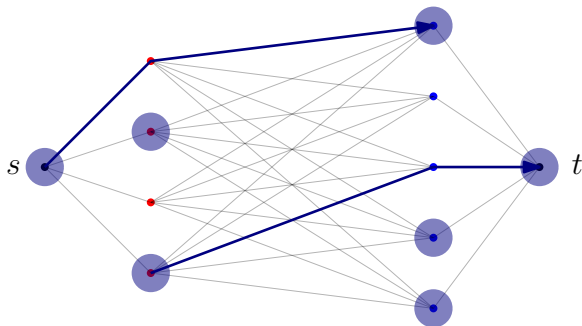
- ▶ Alive paths have length 1, 2, or 3.



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



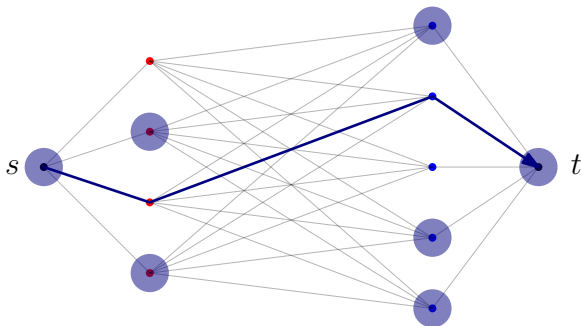
- ▶ Alive paths have length 1, 2, or 3.



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



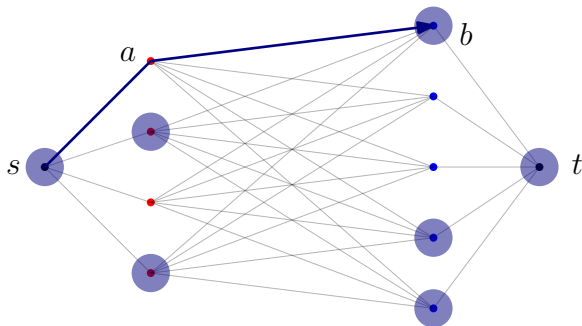
- ▶ Alive paths have length 1, 2, or 3.



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



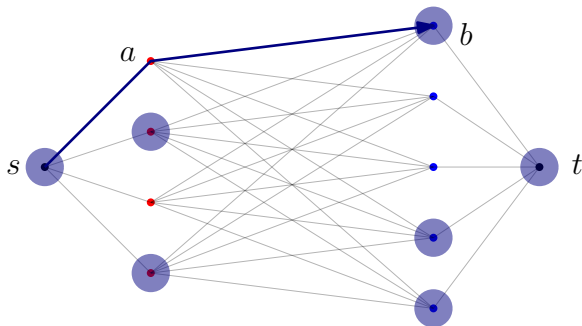
- ▶ Alive paths have length 1, 2, or 3.



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

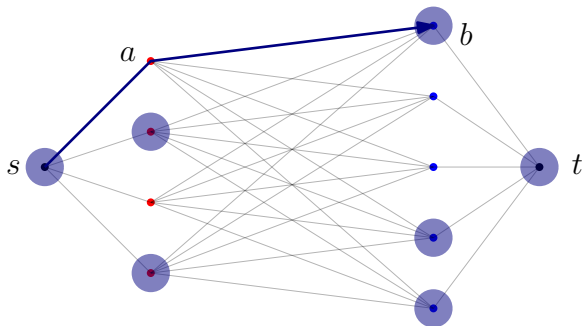


- ▶ Alive paths have length 1, 2, or 3.



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

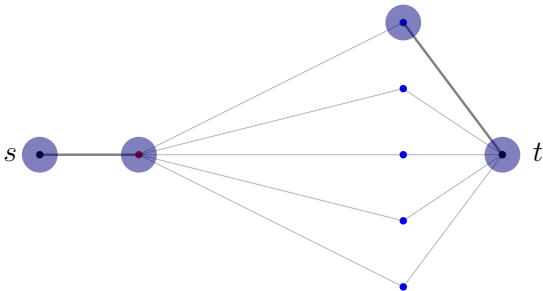
- ▶ Alive paths have length 1, 2, or 3.



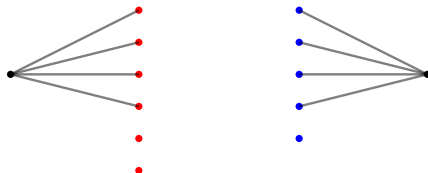
- ▶ Telescoping: $c_\pi(s \rightarrow a \rightarrow b) = c(a \rightarrow b) - \pi(s) + \pi(b)$ (use BCP)
- ▶ Only $O(k)$ relaxations per Hungarian search.
- ▶ Also: find a blocking flow in $O(k)$ relaxations (DFS).



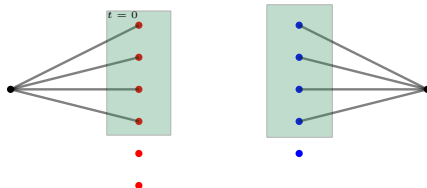
- Dynamic 2D BCP with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)



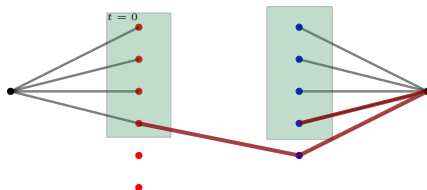
- Some BCP may begin a Hungarian search with $\Theta(n)$ vertices.
- Can't afford to construct from scratch for every Hungarian search.



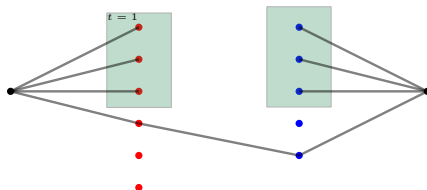
- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



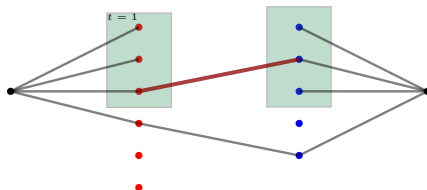
- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



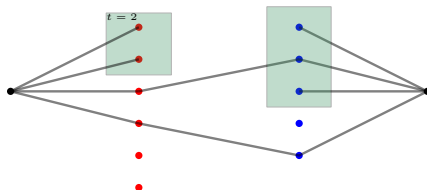
- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



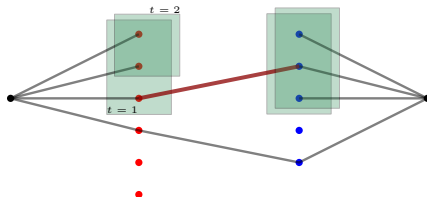
- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



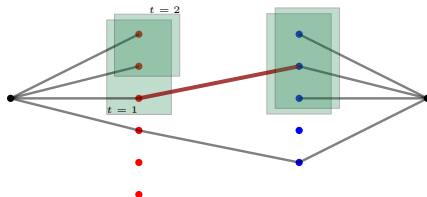
- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?

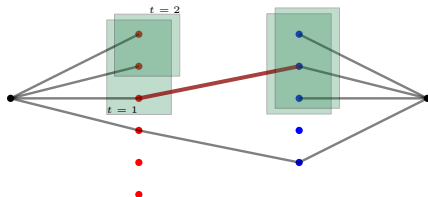


- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).

▶ Persistence?



- ▶ Let X_t be the set X at the start of the t -th Hungarian search.
- ▶ X_t and X_{t+1} differ by only a few nodes (newly matched, dead/alive changes).
- ▶ To generate X_{t+1} :
 1. **Rewind** the BCP updates of the last Hungarian search to obtain X_t .
 2. Apply the few changes (newly matched, dead/alive).
- ▶ Persistence?



- ▶ $O(\log(n^q/\varepsilon))$ 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 $O(k \text{ polylog } n)$ time (**rewinding**). Need to spend $O(n \text{ polylog } n)$ once per scale to build data structures for $t = 0$.
- ▶ $O((n + k\sqrt{k}) \text{ polylog } n \log(1/\varepsilon))$
- ▶ Thank you.



- ▶ $O(\log(n^q/\varepsilon))$ 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 $O(k \text{ polylog } n)$ time (**rewinding**). Need to spend $O(n \text{ polylog } n)$ once per scale to build data structures for $t = 0$.
- ▶ $O((n + k\sqrt{k}) \text{ polylog } n \log(1/\varepsilon))$
- ▶ Thank you.



- ▶ $O(\log(n^q/\varepsilon))$ 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 $O(k \text{ polylog } n)$ time (**rewinding**). Need to spend $O(n \text{ polylog } n)$ once per scale to build data structures for $t = 0$.
- ▶ $O((n + k\sqrt{k}) \text{ polylog } n \log(1/\varepsilon))$
- ▶ Thank you.



- ▶ $O(\log(n^q/\varepsilon))$ 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 $O(k \text{ polylog } n)$ time (**rewinding**). Need to spend $O(n \text{ polylog } n)$ once per scale to build data structures for $t = 0$.
- ▶ $O((n + k\sqrt{k}) \text{ polylog } n \log(1/\varepsilon))$
- ▶ Thank you.



- ▶ $O(\log(n^q/\varepsilon))$ 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 $O(k \text{ polylog } n)$ time (**rewinding**). Need to spend $O(n \text{ polylog } n)$ once per scale to build data structures for $t = 0$.
- ▶ $O((n + k\sqrt{k}) \text{ polylog } n \log(1/\varepsilon))$
- ▶ Thank you.