Efficient Algorithms for Geometric Partial Matching

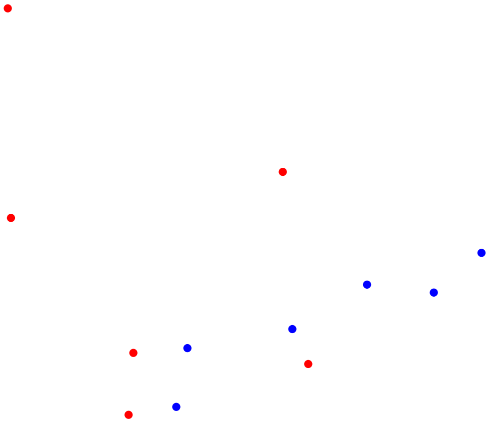
Pankaj K. Agarwal Hsien-Chih Chang Allen Xiao

Department of Computer Science, Duke University

June 2019

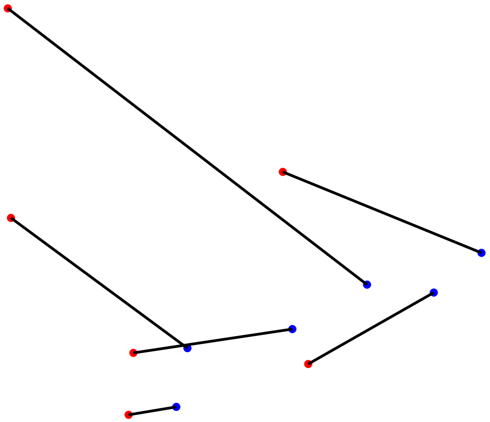
Geometric (bipartite) matching

A, B



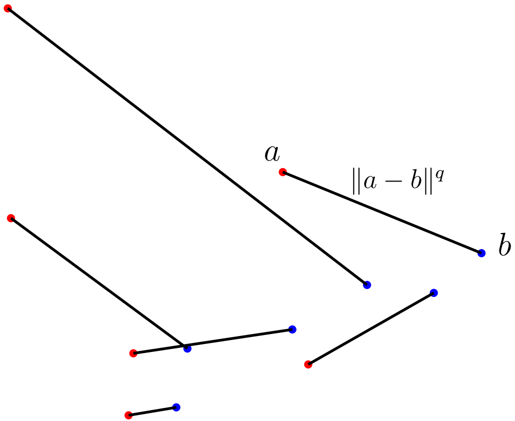
Geometric (bipartite) matching

A, B



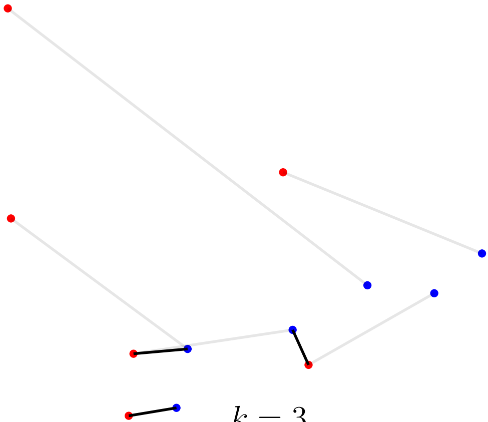
Geometric (bipartite) matching

A, B



Geometric (bipartite) partial matching

A, B



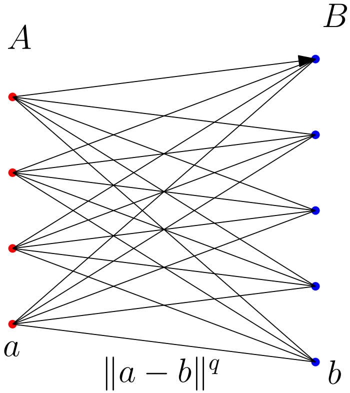
$k = 3$

Prior work

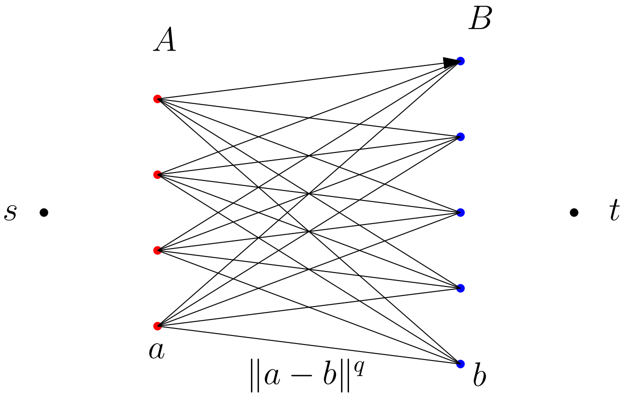
	approx.	time	valid q
Hungarian algorithm (Kuhn)	exact	$O(km + k^2 \log n)$	$q \geq 1$
		$O(kn \operatorname{polylog} n)$	
Ramshaw, Tarjan 2012	exact[1]	$O(m\sqrt{k} \log(kC))$	$q \geq 1$
	$(1+\varepsilon)$	$O(n\sqrt{k} \operatorname{polylog} n \log(1/\varepsilon))$	
Sharathkumar, Agarwal 2012	$(1+\varepsilon)$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

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

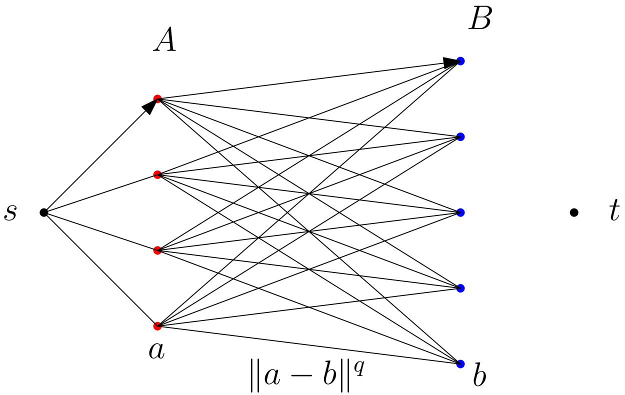
Unit-capacity min-cost flow formulation



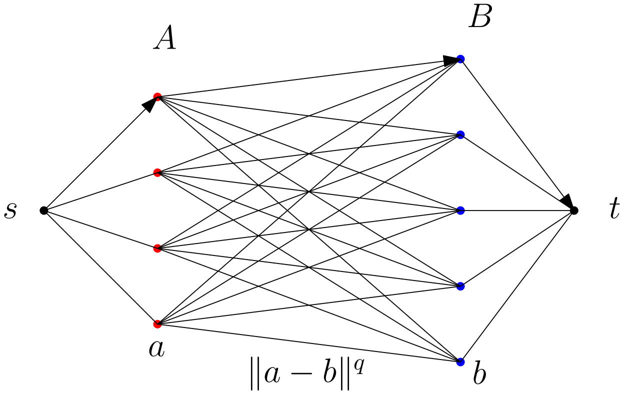
Unit-capacity min-cost flow formulation



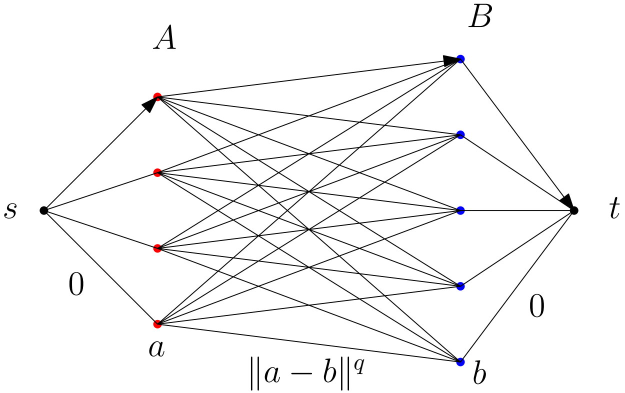
Unit-capacity min-cost flow formulation



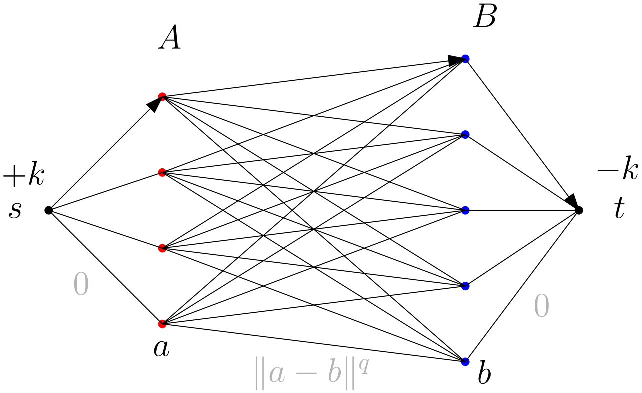
Unit-capacity min-cost flow formulation



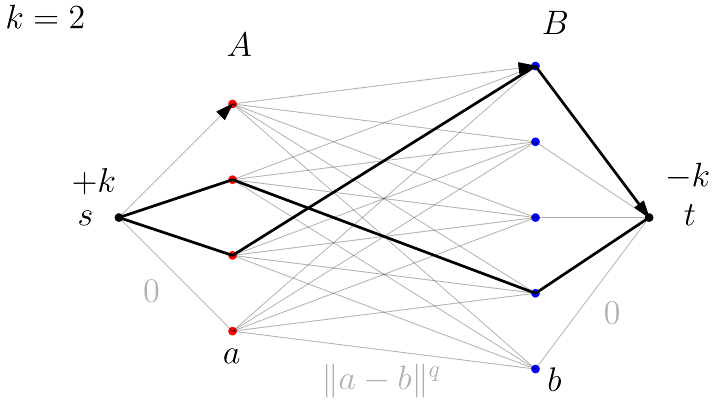
Unit-capacity min-cost flow formulation



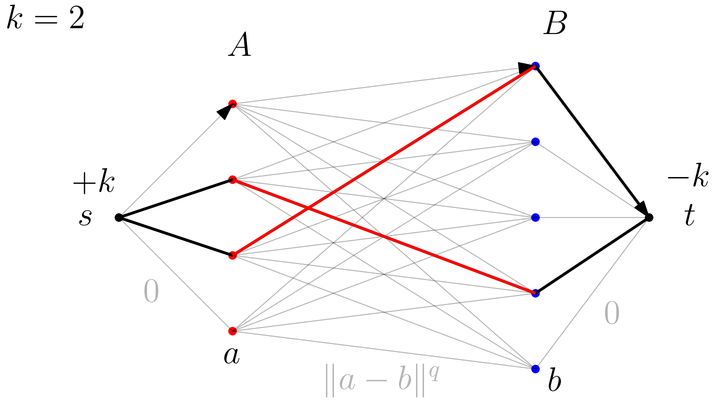
Unit-capacity min-cost flow formulation



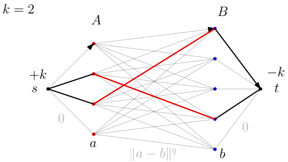
Unit-capacity min-cost flow formulation



Unit-capacity min-cost flow formulation



Flow terminology



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

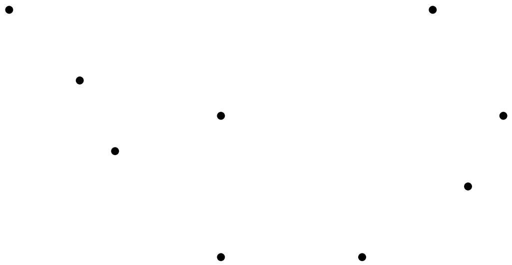
Cost-scaling (Ramshaw-Tarjan)

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

- θ-optimal circulation is $+n\theta$ approx.

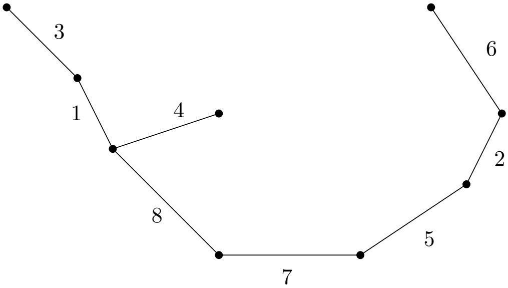
- Find θ-optimal circulations for geometrically decreasing values of θ:
  1. Reduce $\theta \leftarrow \theta/2$, while creating $O(k)$ excess.
  2. Refine this pseudoflow into a circulation, while preserving θ-optimality

Cost-scaling for geometric partial matching



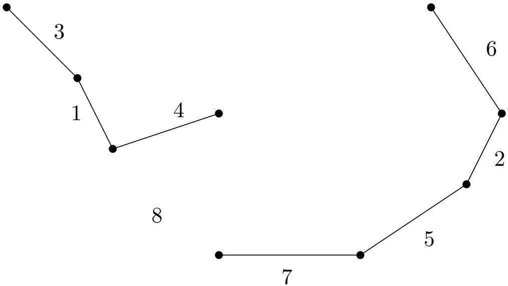
- There exists a k-matching whose longest edge is $(n^q \cdot \alpha)$, and a $(\varepsilon\alpha/6k)$-optimal circulation is $(1+\varepsilon)$-approx.
- $(1+\varepsilon)$-approx. geometric partial matching reduces into executing $O(\log(n^q/\varepsilon))$ cost scales.

Cost-scaling for geometric partial matching

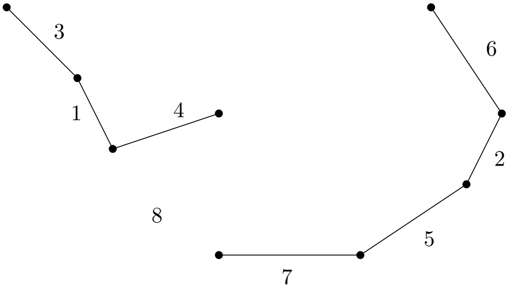


- There exists a k-matching whose longest edge is $(n^q \cdot \alpha)$, and a $(\varepsilon\alpha/6k)$-optimal circulation is $(1+\varepsilon)$-approx.
- $(1+\varepsilon)$-approx. geometric partial matching reduces into executing $O(\log(n^q/\varepsilon))$ cost scales.

Cost-scaling for geometric partial matching

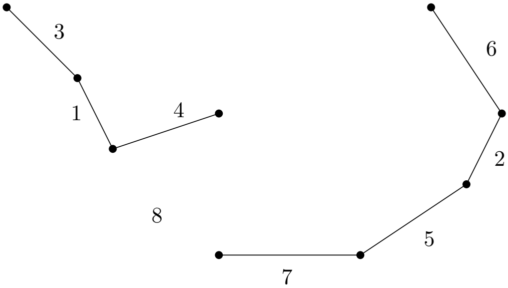


- There exists a k-matching whose longest edge is $(n^q \cdot \alpha)$, and a $(\varepsilon\alpha/6k)$-optimal circulation is $(1+\varepsilon)$-approx.
- $(1+\varepsilon)$-approx. geometric partial matching reduces into executing $O(\log(n^q/\varepsilon))$ cost scales.



- There exists a k-matching whose longest edge is $(n^q \cdot \alpha)$, and a $(\varepsilon\alpha/6k)$-optimal circulation is $(1+\varepsilon)$-approx.
- $(1+\varepsilon)$-approx. geometric partial matching reduces into executing $O(\log(n^q/\varepsilon))$ cost scales.

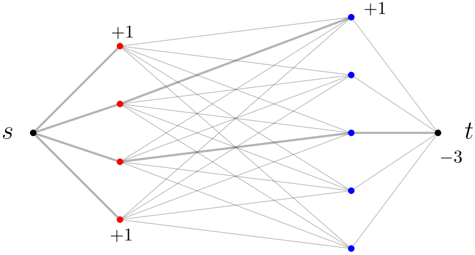
Cost-scaling for geometric partial matching



- There exists a k-matching whose longest edge is $(n^q \cdot \alpha)$, and a $(\varepsilon\alpha/6k)$-optimal circulation is $(1+\varepsilon)$-approx.
- $(1+\varepsilon)$-approx. geometric partial matching reduces into executing $O(\log(n^q/\varepsilon))$ cost scales.

Refinement by blocking flows (Ramshaw-Tarjan)

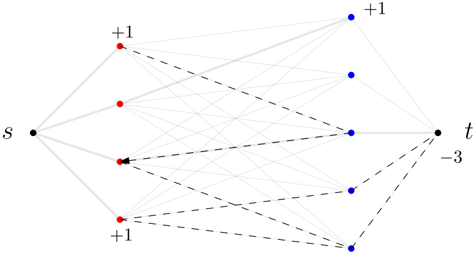
- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.



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

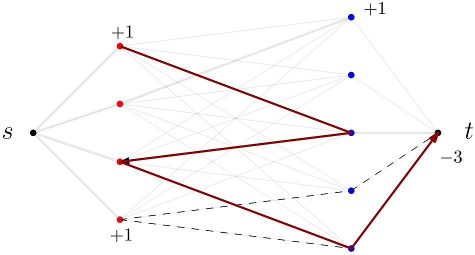
Refinement by blocking flows (Ramshaw-Tarjan)

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



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

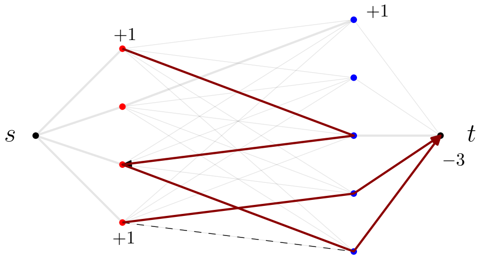
- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.



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

Refinement by blocking flows (Ramshaw-Tarjan)

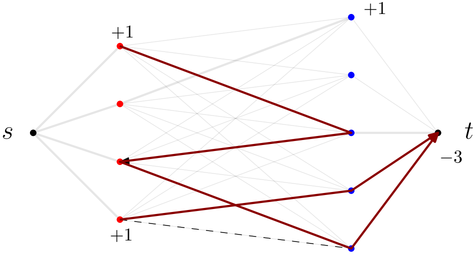
- Inside Refine:
  1. Hungarian search: raise potentials until an excess-deficit admissible path exists.
  2. Augment by an admissible blocking flow.



- $O(\sqrt{k})$ 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.



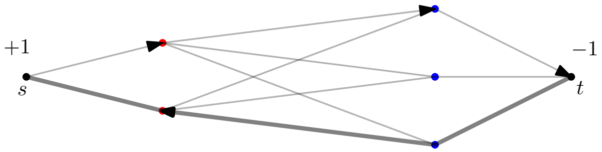
- $O(\sqrt{k})$ blocking flows before f becomes a circulation.

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.

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

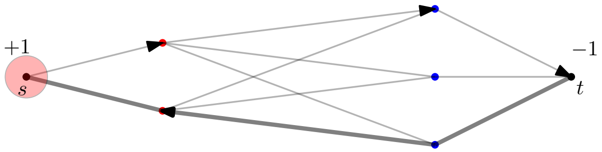
X: admissible reachable from an excess node



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

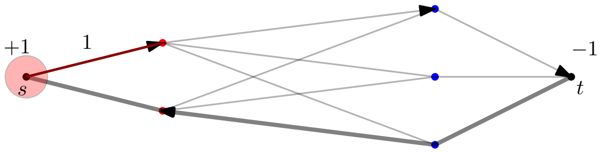
Hungarian search with BCP (Agarwal-Efrat-Sharir)

X: admissible reachable from an excess node



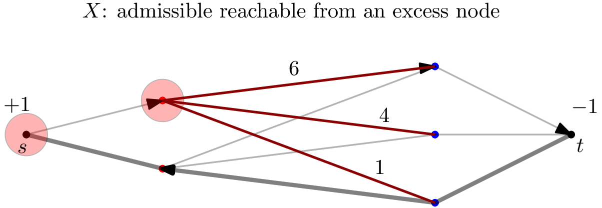
- Dynamic 2D BCP with $O(\text{polylog}\, n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)

X: admissible reachable from an excess node



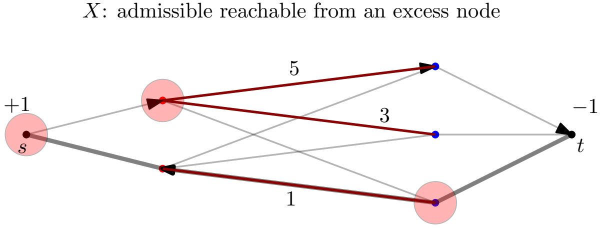
- Dynamic 2D BCP with $O(\text{polylog } n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)

Hungarian search with BCP (Agarwal-Efrat-Sharir)



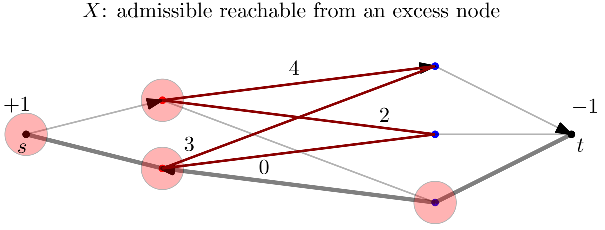
- Dynamic 2D BCP with $O(\text{polylog}\, n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)

Hungarian search with BCP (Agarwal-Efrat-Sharir)



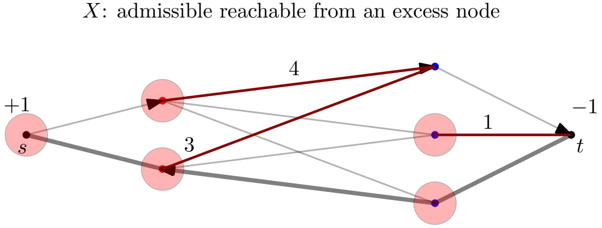
- Dynamic 2D BCP with $O(\operatorname{polylog} n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)

Hungarian search with BCP (Agarwal-Efrat-Sharir)



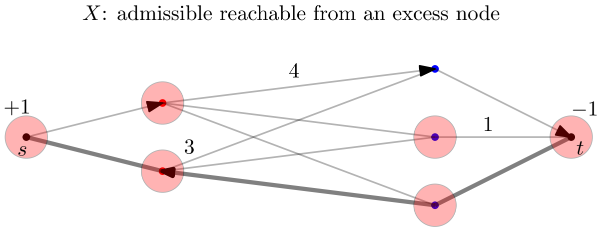
- Dynamic 2D BCP with $O(\operatorname{polylog} n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)

Hungarian search with BCP (Agarwal-Efrat-Sharir)



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

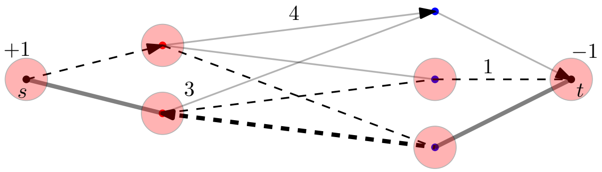
Hungarian search with BCP (Agarwal-Efrat-Sharir)



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

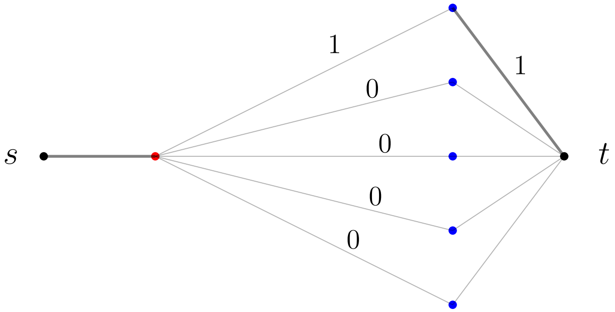
Hungarian search with BCP (Agarwal-Efrat-Sharir)

X: admissible reachable from an excess node

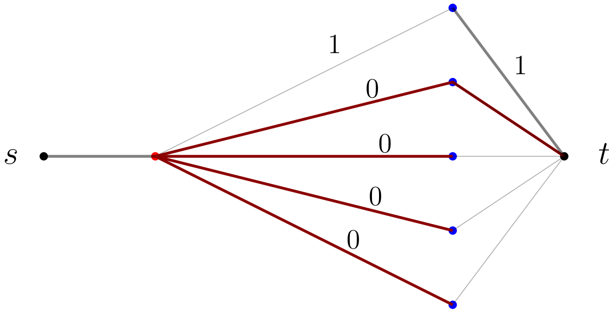


- Dynamic 2D BCP with $O(\mathrm{polylog}\, n)$ update time, $O(\log^2 n)$ query time (Kaplan *et al.* SODA'17)

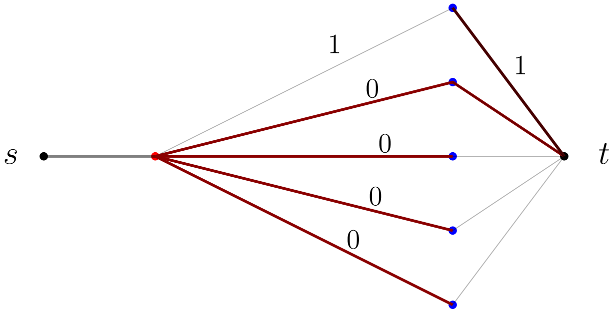
Problem: Hungarian search looks at too many nodes



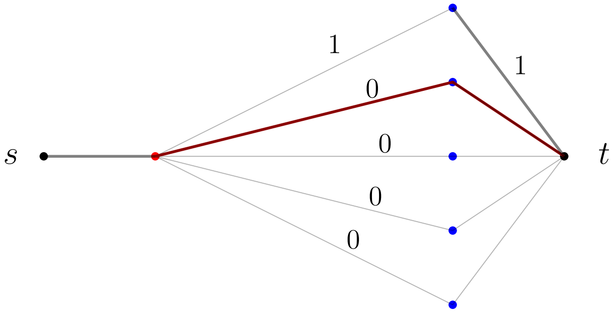
Problem: Hungarian search looks at too many nodes



Problem: Hungarian search looks at too many nodes

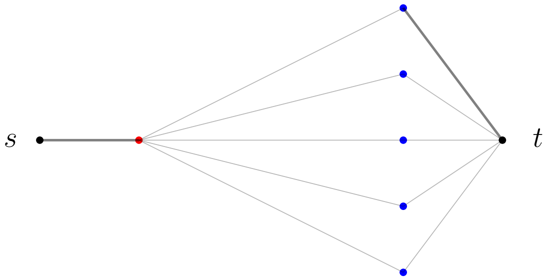


Problem: Hungarian search looks at too many nodes



Dead/alive nodes

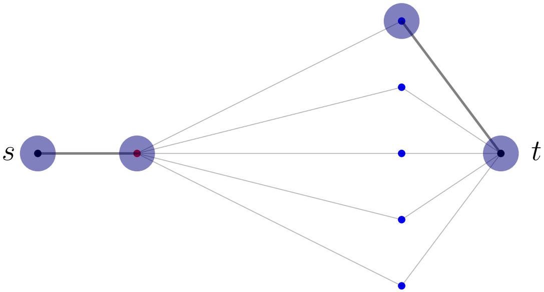
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Dead/alive nodes

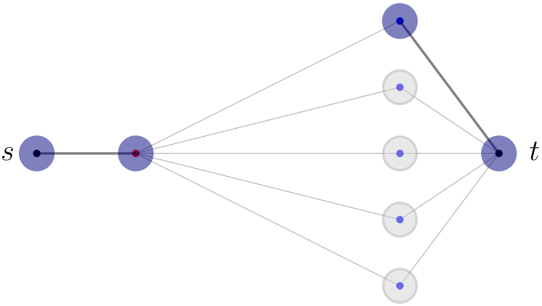
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Dead/alive nodes

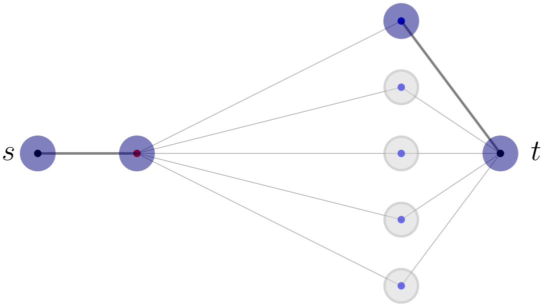
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Dead/alive nodes

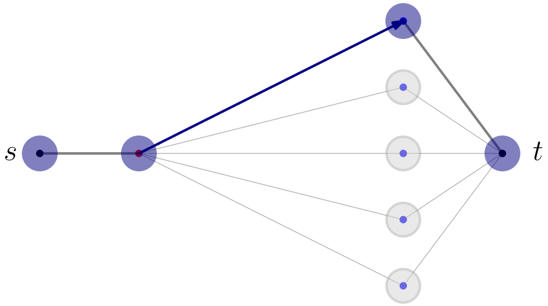
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Dead/alive nodes

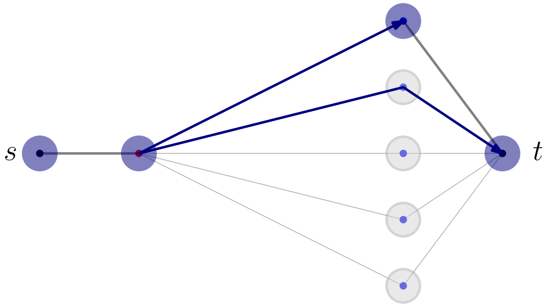
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Dead/alive nodes

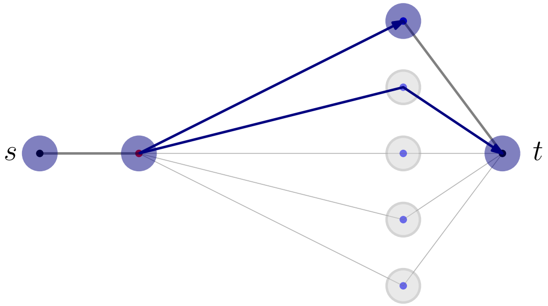
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Dead/alive nodes

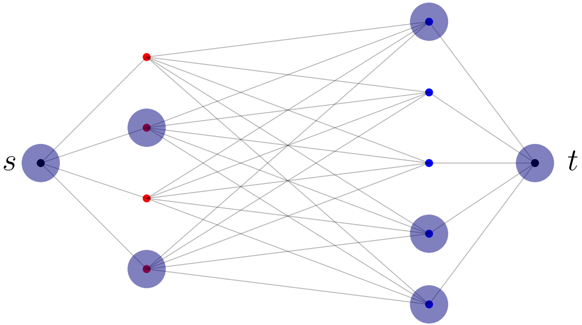
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- Alive path: residual path between two alive nodes with no other alive nodes in between.
- Don't need to track potential of dead nodes.

Relaxing alive paths

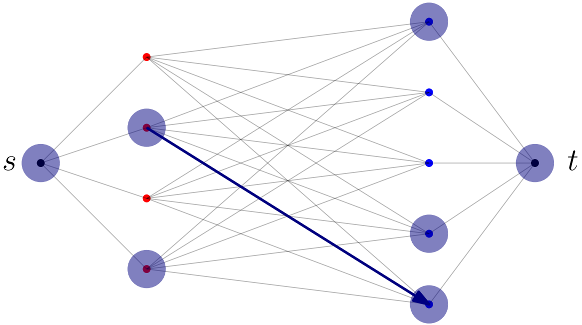
- Alive paths have length 1, 2, or 3.



- 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

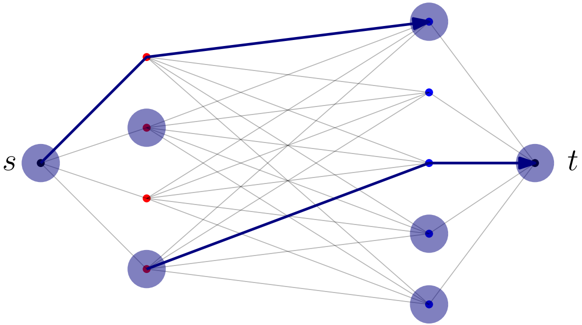
- Alive paths have length 1, 2, or 3.



- 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

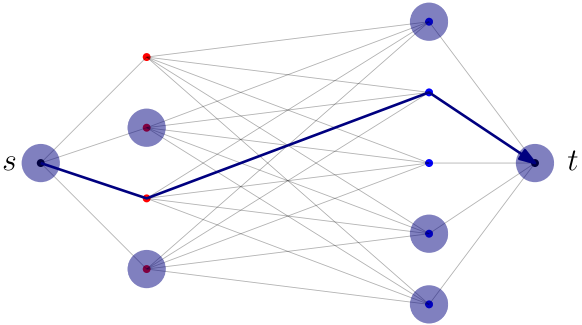
- Alive paths have length 1, 2, or 3.



- 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

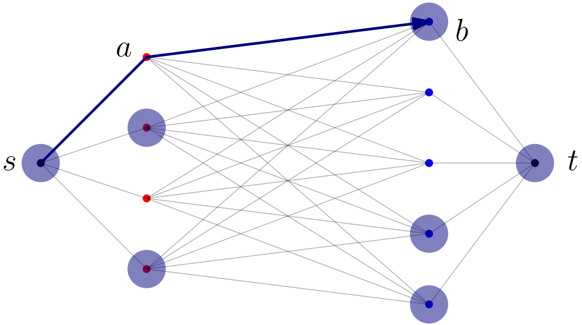
- Alive paths have length 1, 2, or 3.



- 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

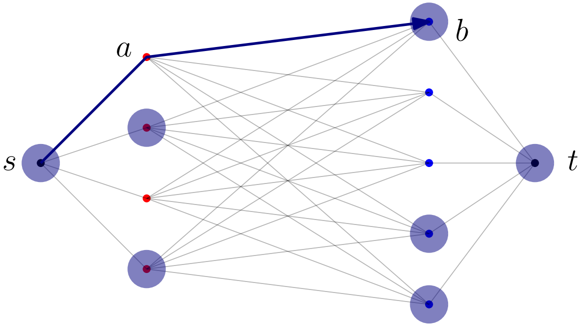
- Alive paths have length 1, 2, or 3.



- 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

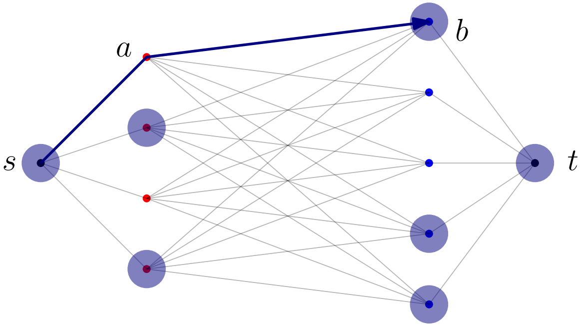
- Alive paths have length 1, 2, or 3.



- 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

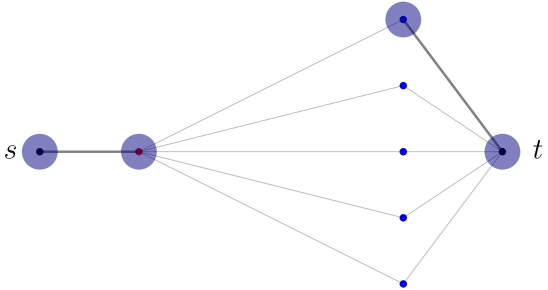
- Alive paths have length 1, 2, or 3.



- 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).

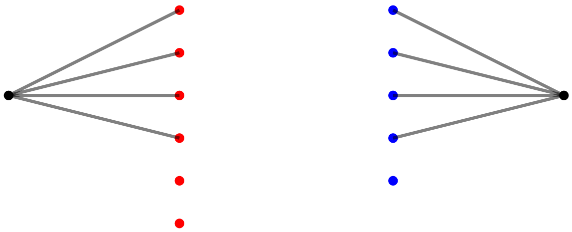
Problem: BCP initialization

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



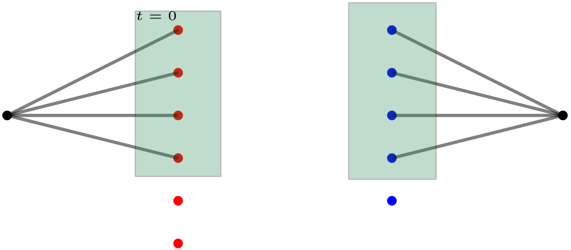
- Some BCP may begin a Hungarian search with $\Theta(n)$ vertices.
- Can't afford to construct from scratch for every Hungarian search.

Initial BCP by rewinding



- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

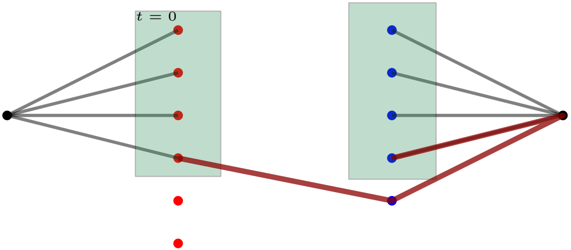
Initial BCP by rewinding



- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?

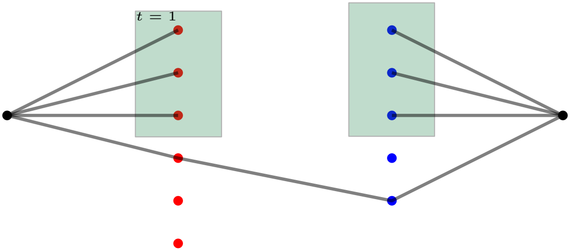
- Construct once $(O(n \operatorname{polylog} n))$, then $O(k \operatorname{polylog} n)$ time to rewind.

Initial BCP by rewinding



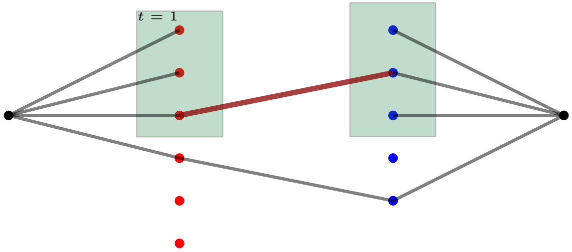
- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

Initial BCP by rewinding



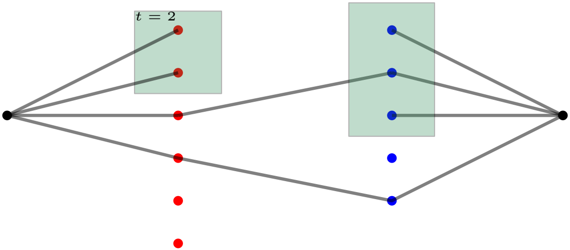
- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

Initial BCP by rewinding

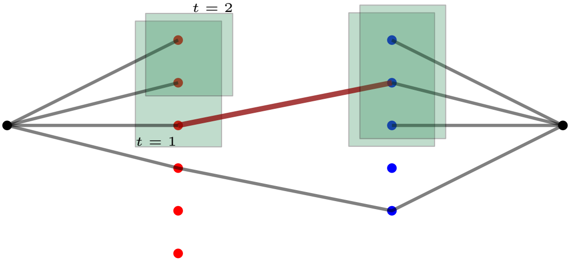


- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

Initial BCP by rewinding

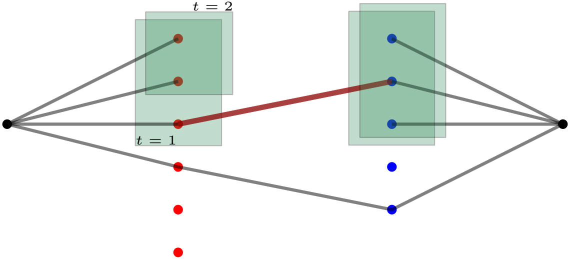


- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.



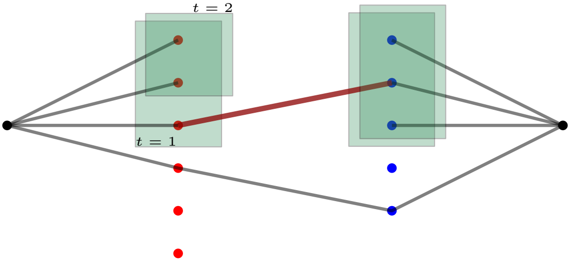
- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

Initial BCP by rewinding



- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

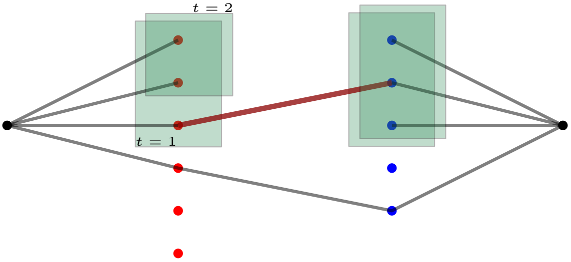
Initial BCP by rewinding



- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?

- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind.

Initial BCP by rewinding



- X_t and X_{t+1} differ by the newly-matched A nodes.
- Generate X_{t+1} by rewinding the BCP updates done on X_t, then deleting the newly-matched nodes.
  Same number of BCP updates as the Hungarian search.
- Persistence?
- Construct once ($O(n \operatorname{polylog} n)$), then $O(k \operatorname{polylog} n)$ time to rewind

The End

Thank you.