Efficient Algorithms for Geometric Partial Matching

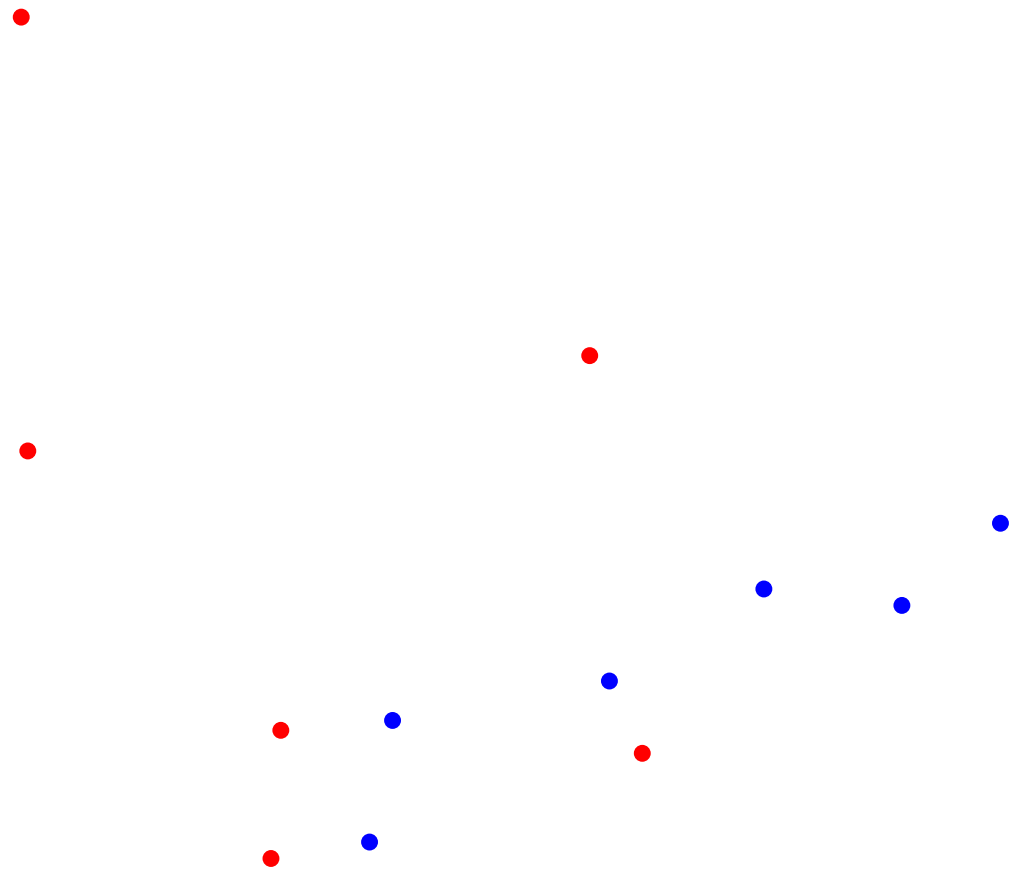
Pankaj K. Agarwal Hsien-Chih Chang Allen Xiao

Department of Computer Science, Duke University

June 2019

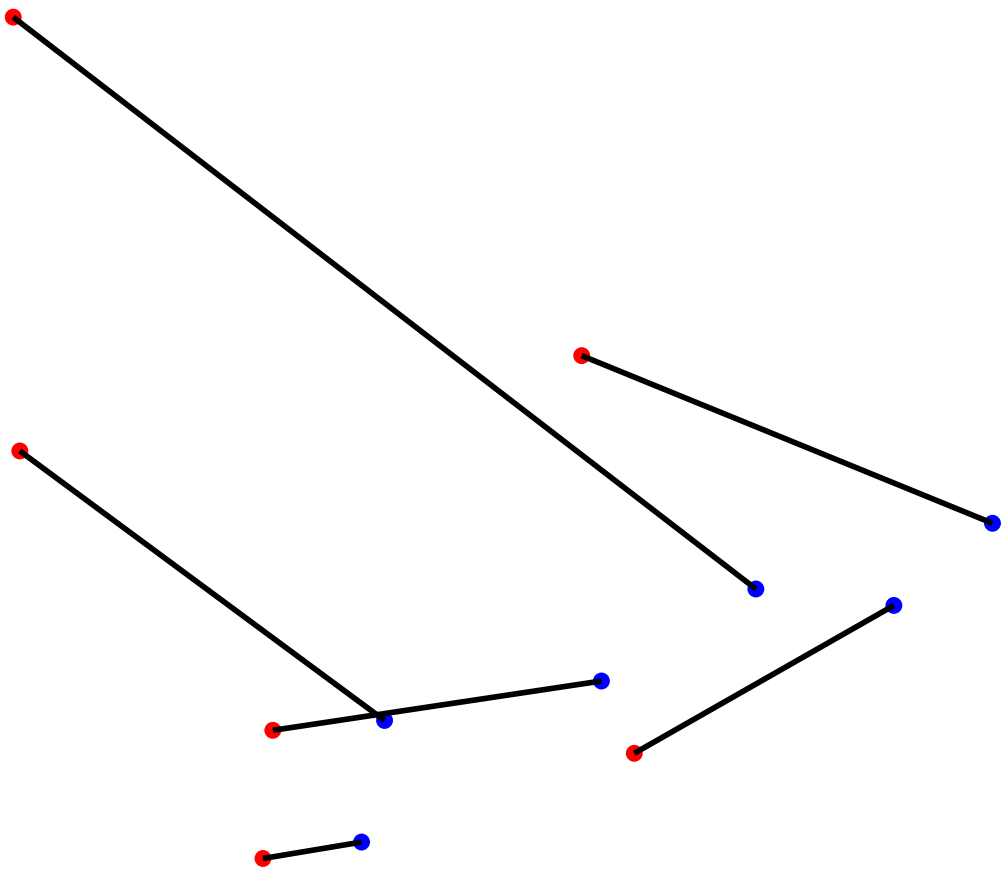
Geometric (bipartite) matching

A, B



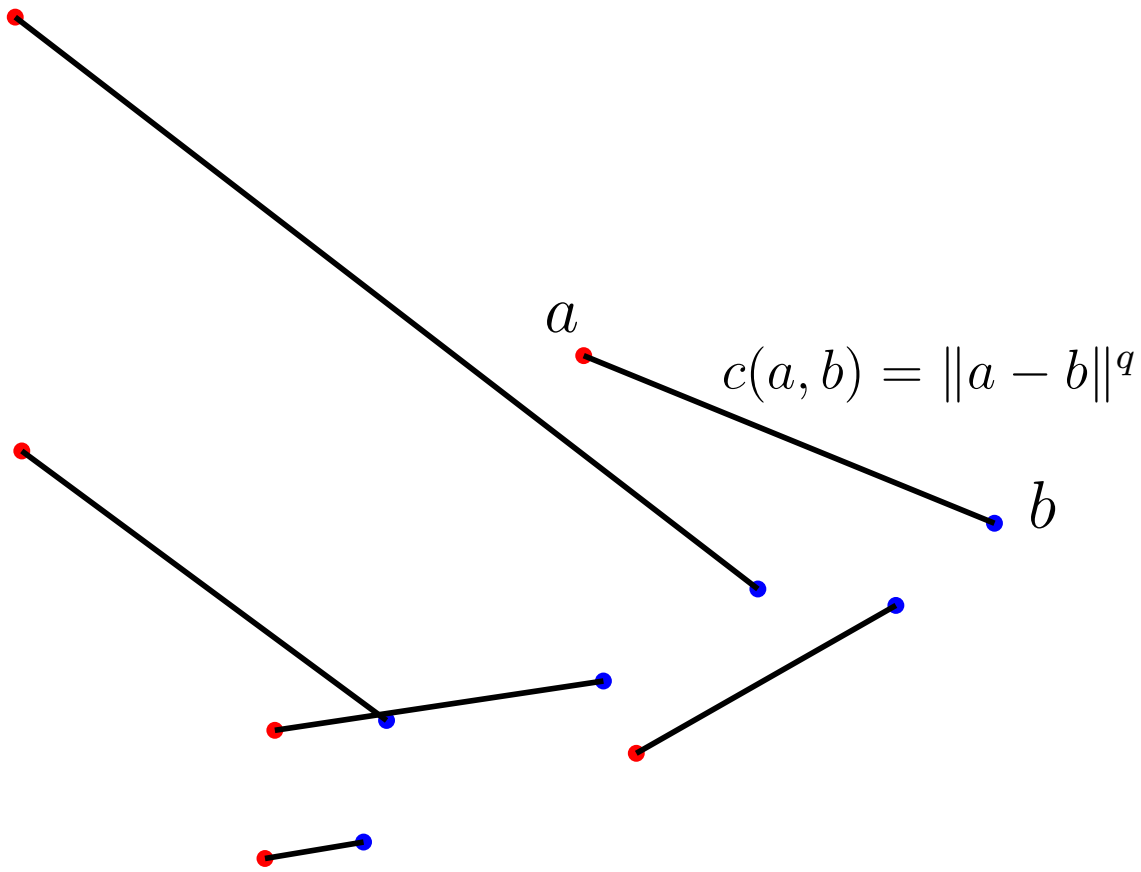
Geometric (bipartite) matching

A, B



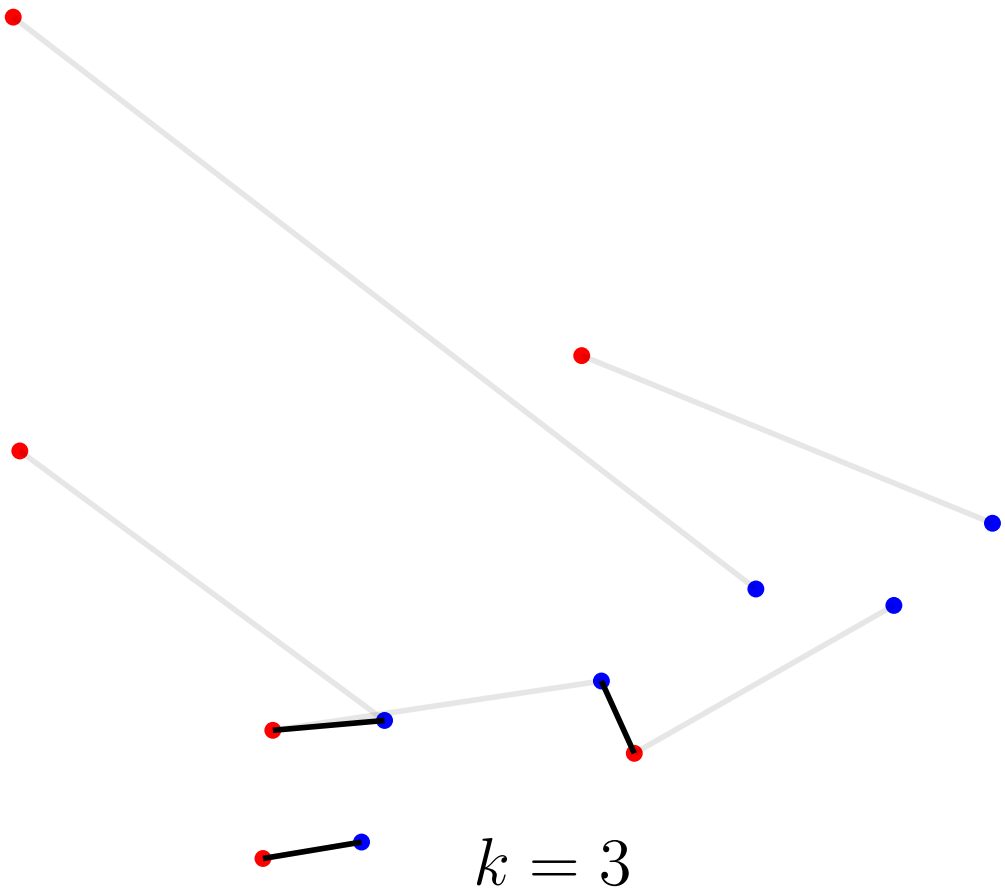
Geometric (bipartite) matching

A, B



Geometric (bipartite) partial matching

A, B

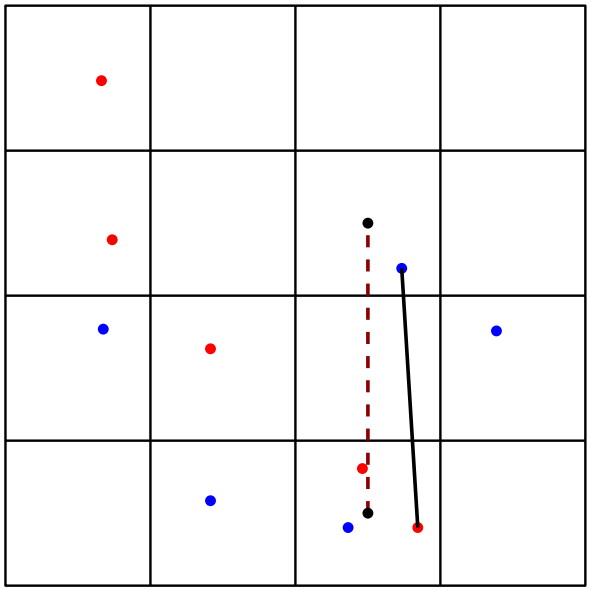


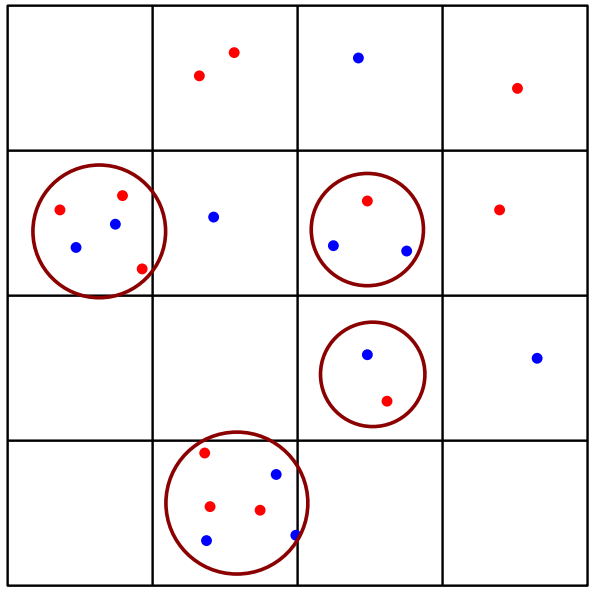
$k = 3$

Issues for geometric perfect matching algorithms

$$|A| = r, \quad |B| = n, \quad A, B \subset \mathbb{R}^2, \quad q \geq 1$$

$$\min_{k\text{-matching } M} \sum_{(a,b)\in M} \|a - b\|^q$$





For $q = 1$, [Sharathkumar-Agarwal]: $(1+\varepsilon)$-approx. in $O(n \operatorname{poly}(1/\varepsilon, \log n))$ time.

Partial matching algorithms (non-geometric)

$$|A| = r, \quad |B| = n, \quad A, B \subset \mathbb{R}^2, \quad q \geq 1$$

$$\min_{k\text{-matching } M} \sum_{(a,b)\in M} \|a - b\|^q$$

- Hungarian algorithm [Kuhn]: $O(krn + k^2 \log r)$
- Cost-scaling [Ramshaw-Tarjan, Goldberg *et al.*]: $O(rn\sqrt{k}\log(kC))$ (integer costs), or $O(rn\sqrt{k})$ time per cost scale.

Using dynamic bichromatic closest pair (BCP) data structures:

- Hungarian algorithm: $O(kn\,\mathrm{polylog}(n))$
- Cost-scaling: $O(n\sqrt{k}\,\mathrm{polylog}(n))$ time per cost scale.

Partial matching algorithms (non-geometric)

$$|A| = r, \quad |B| = n, \quad A, B \subset \mathbb{R}^2, \quad q \geq 1$$

$$\min_{k\text{-matching } M} \sum_{(a,b) \in M} \|a - b\|^q$$

- Hungarian algorithm [Kuhn]: $O(krn + k^2 \log r)$
- Cost-scaling [Ramshaw-Tarjan, Goldberg *et al.*]: $O(rn\sqrt{k}\log(kC))$ (integer costs), or $O(rn\sqrt{k})$ time per cost scale.

Using dynamic bichromatic closest pair (BCP) data structures:

- Hungarian algorithm: $O(kn\,\mathrm{polylog}(n))$
- Cost-scaling: $O(n\sqrt{k}\,\mathrm{polylog}(n))$ time per cost scale.

Our results

Previous:

- Hungarian algorithm w/ BCP: $O(kn \operatorname{polylog}(n))$
- Cost-scaling w/ BCP: $O(n\sqrt{k} \operatorname{polylog}(n))$ time per cost scale.

New:

1. Hungarian algorithm: $O((n + k^2) \operatorname{polylog}(n))$ time, exact.
2. Cost-scaling: $O((n + k\sqrt{k}) \operatorname{polylog}(n))$ time per cost scale, $O(\log(n^q/\varepsilon))$ scales, $(1 + \varepsilon)$-approx.
3. Geometric Transportation: $O(\min(n^2, nr^{3/2}) \operatorname{polylog}(n))$ time, exact.

Our results

Previous:

- Hungarian algorithm w/ BCP: $O(kn \operatorname{polylog}(n))$
- Cost-scaling w/ BCP: $O(n\sqrt{k} \operatorname{polylog}(n))$ time per cost scale.

New:

1. Hungarian algorithm: $O((n + k^2) \operatorname{polylog}(n))$ time, exact.
2. Cost-scaling: $O((n + k\sqrt{k}) \operatorname{polylog}(n))$ time per cost scale, $O(\log(n^q/\varepsilon))$ scales, $(1 + \varepsilon)$-approx.
3. Geometric Transportation: $O(\min(n^2, nr^{3/2}) \operatorname{polylog}(n))$ time, exact.

Our results

Previous:

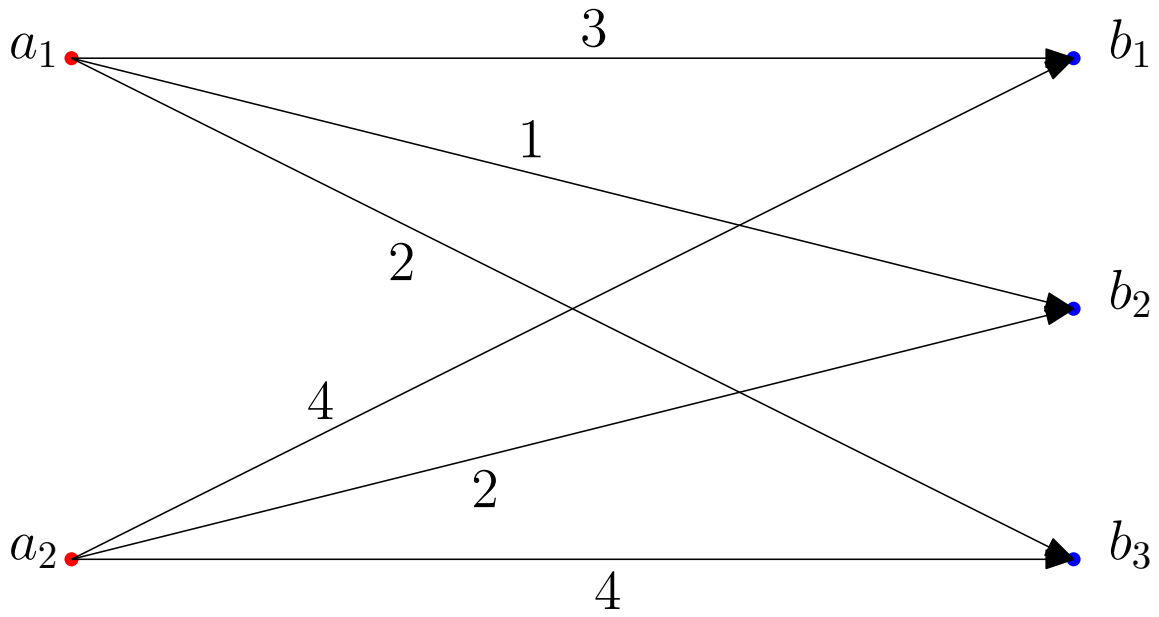
- Hungarian algorithm w/ BCP: $O(kn \operatorname{polylog}(n))$
- Cost-scaling w/ BCP: $O(n\sqrt{k} \operatorname{polylog}(n))$ time per cost scale.

New:

1. Hungarian algorithm: $O((n + k^2) \operatorname{polylog}(n))$ time, exact.
2. Cost-scaling: $O((n + k\sqrt{k}) \operatorname{polylog}(n))$ time per cost scale, $O(\log(n^q/\varepsilon))$ scales, $(1 + \varepsilon)$-approx.

3. Geometric Transportation: $O(\min(n^2, nr^{3/2}) \operatorname{polylog}(n))$ time, exact.

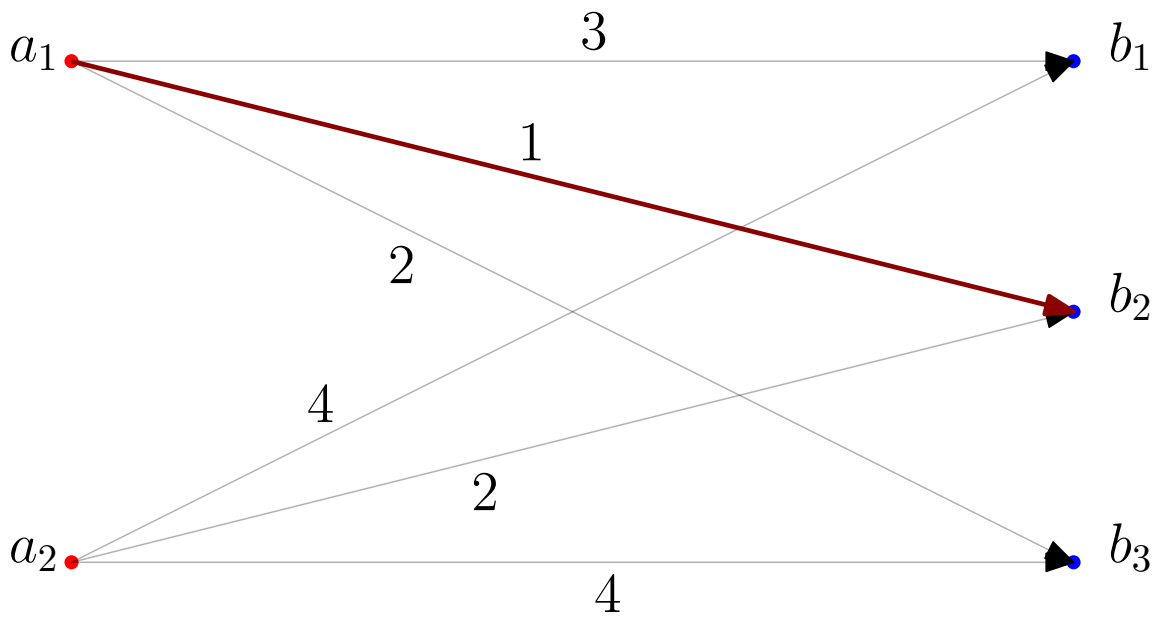
Hungarian algorithm



Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

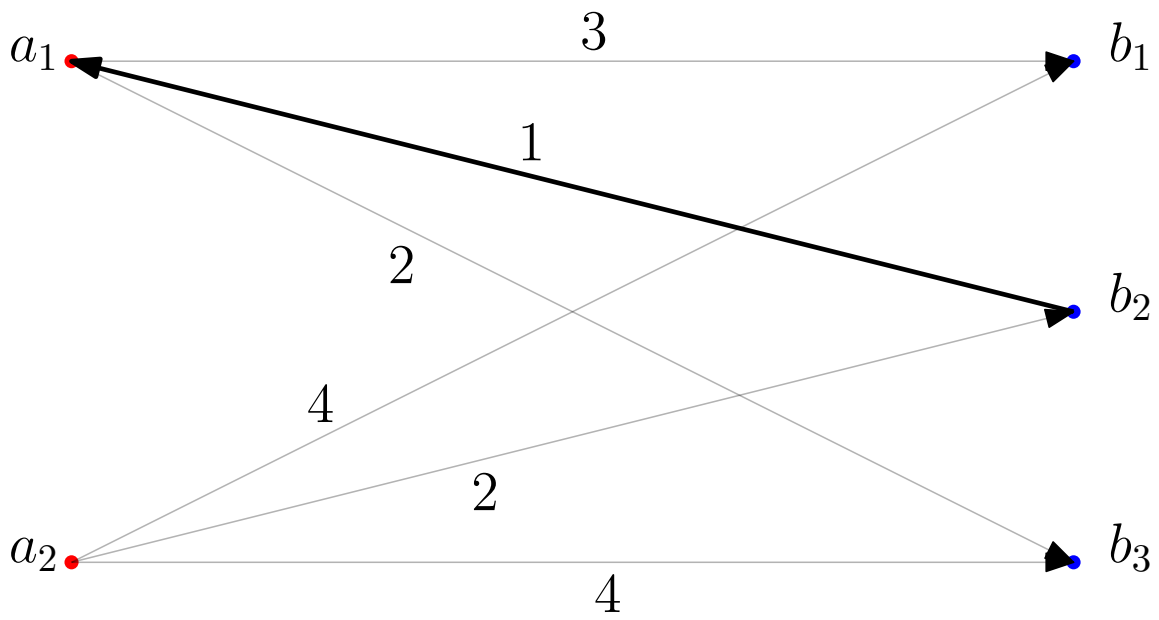
Hungarian algorithm



Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

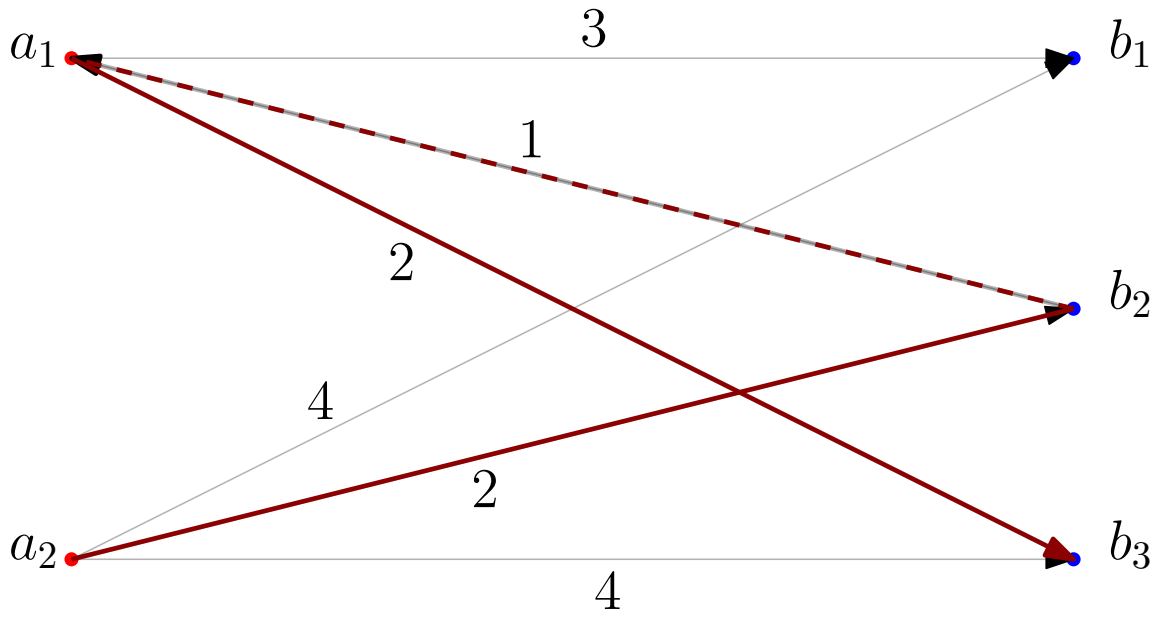
Hungarian algorithm



Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

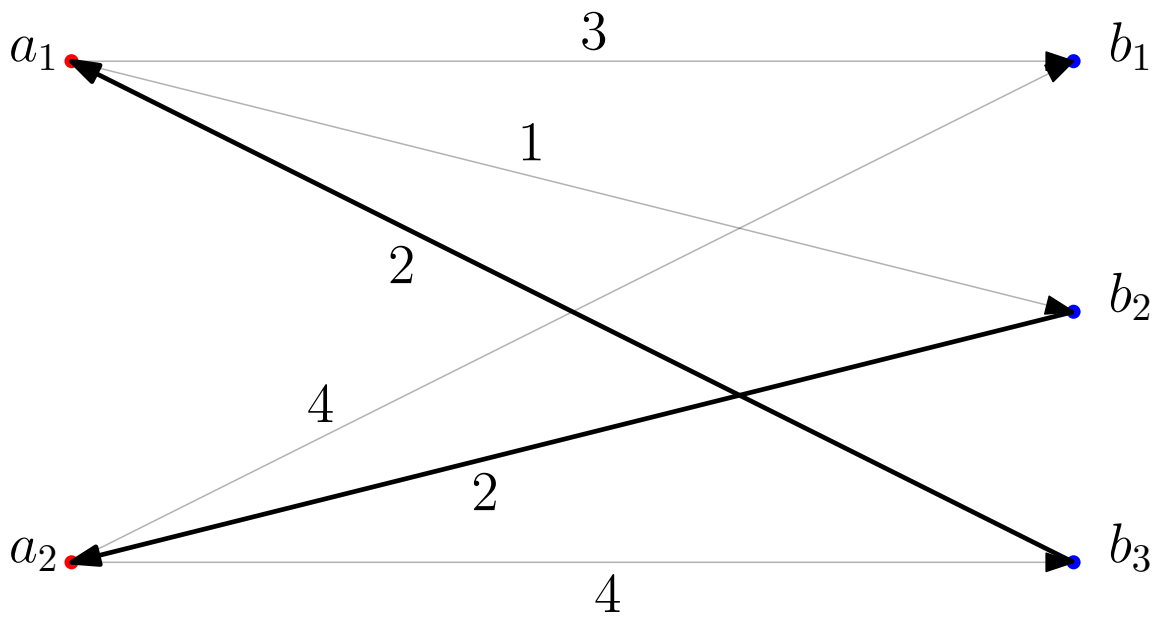
Hungarian algorithm



Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

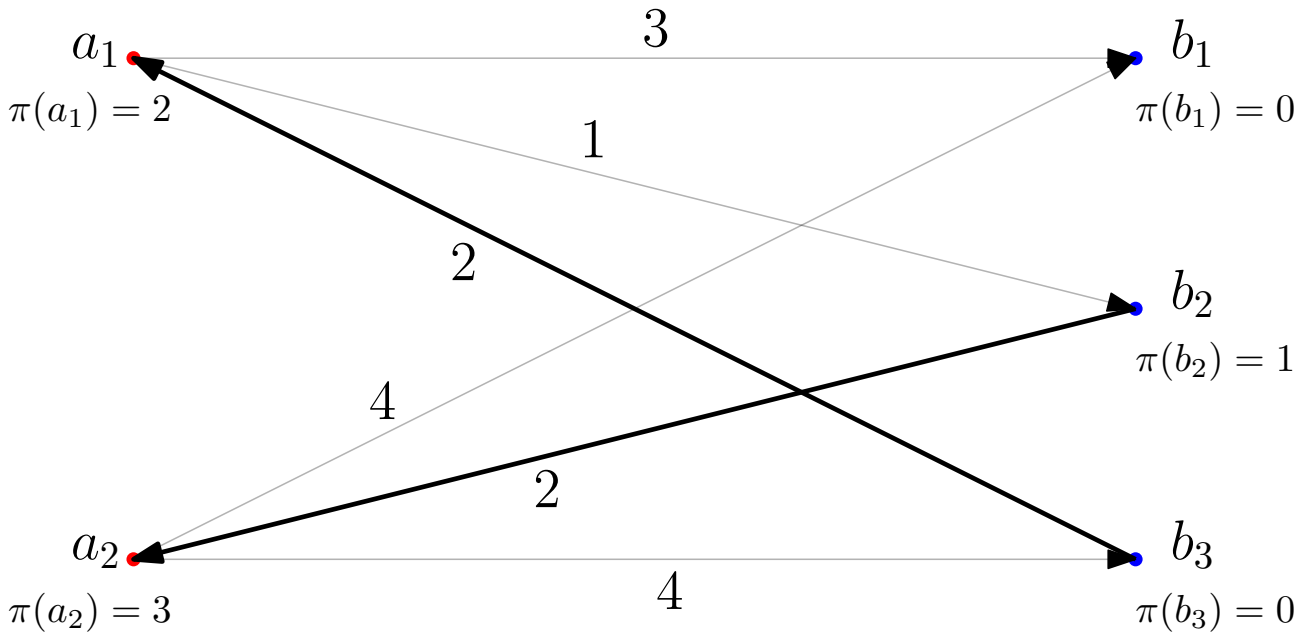
Hungarian algorithm



Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

Hungarian algorithm

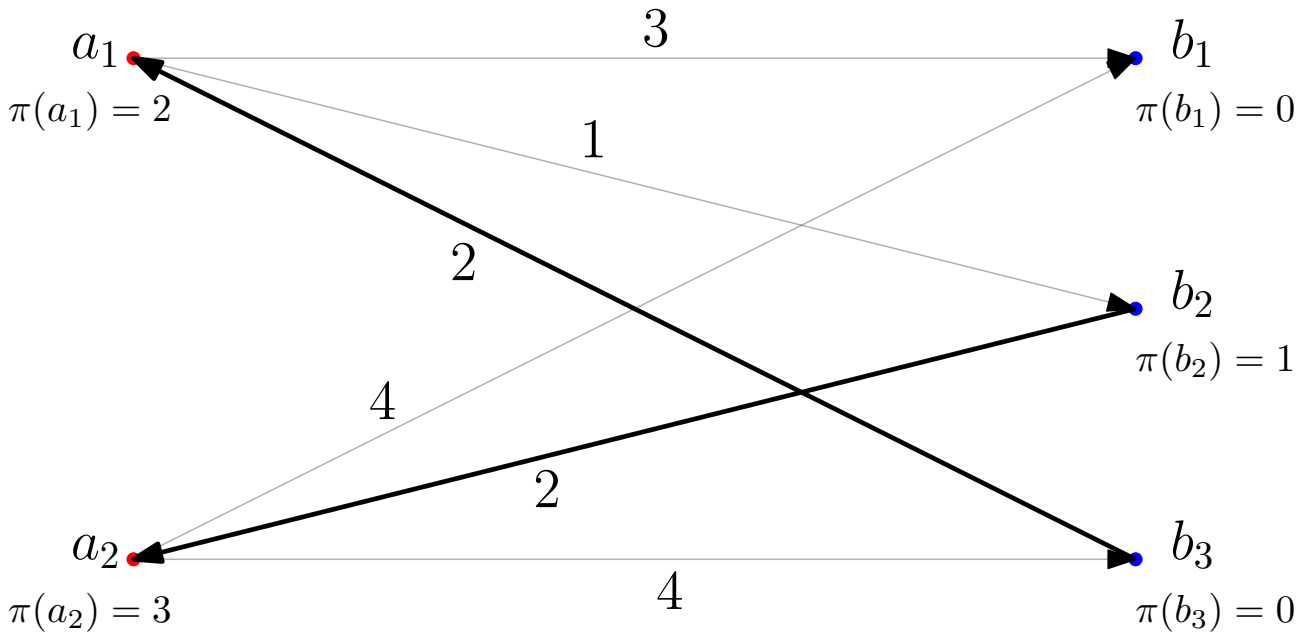


Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

Hungarian algorithm

$$c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$$

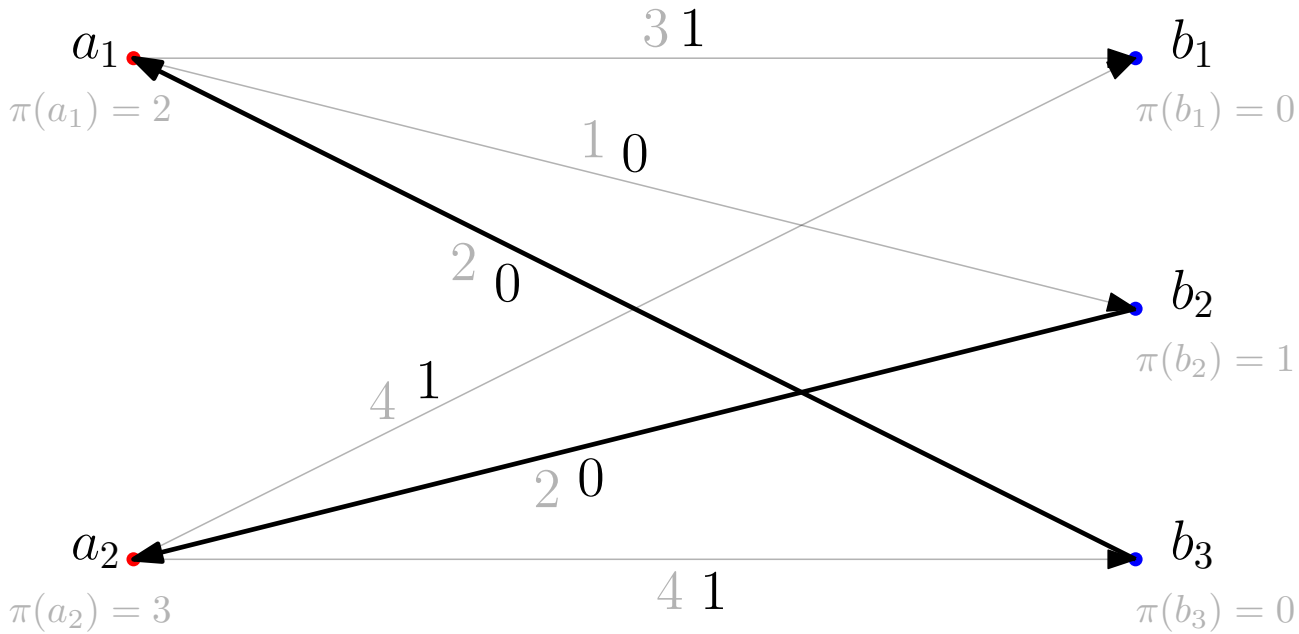


Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

Hungarian algorithm

$$c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$$

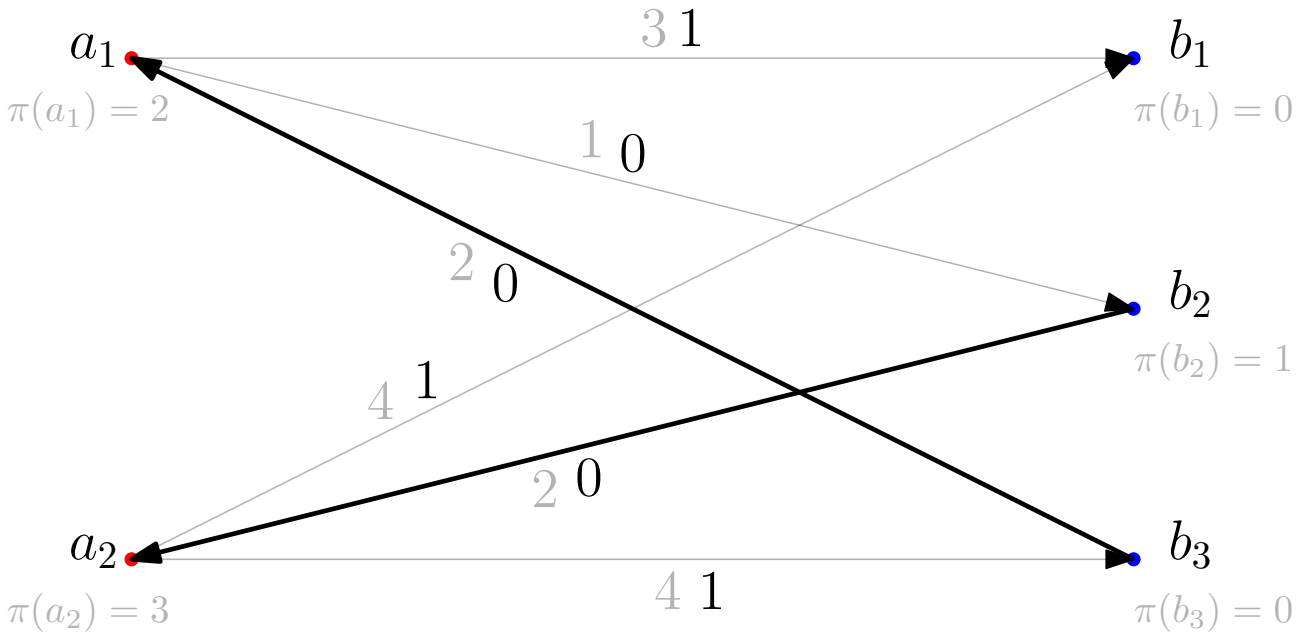


Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

Hungarian algorithm

$$c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$$

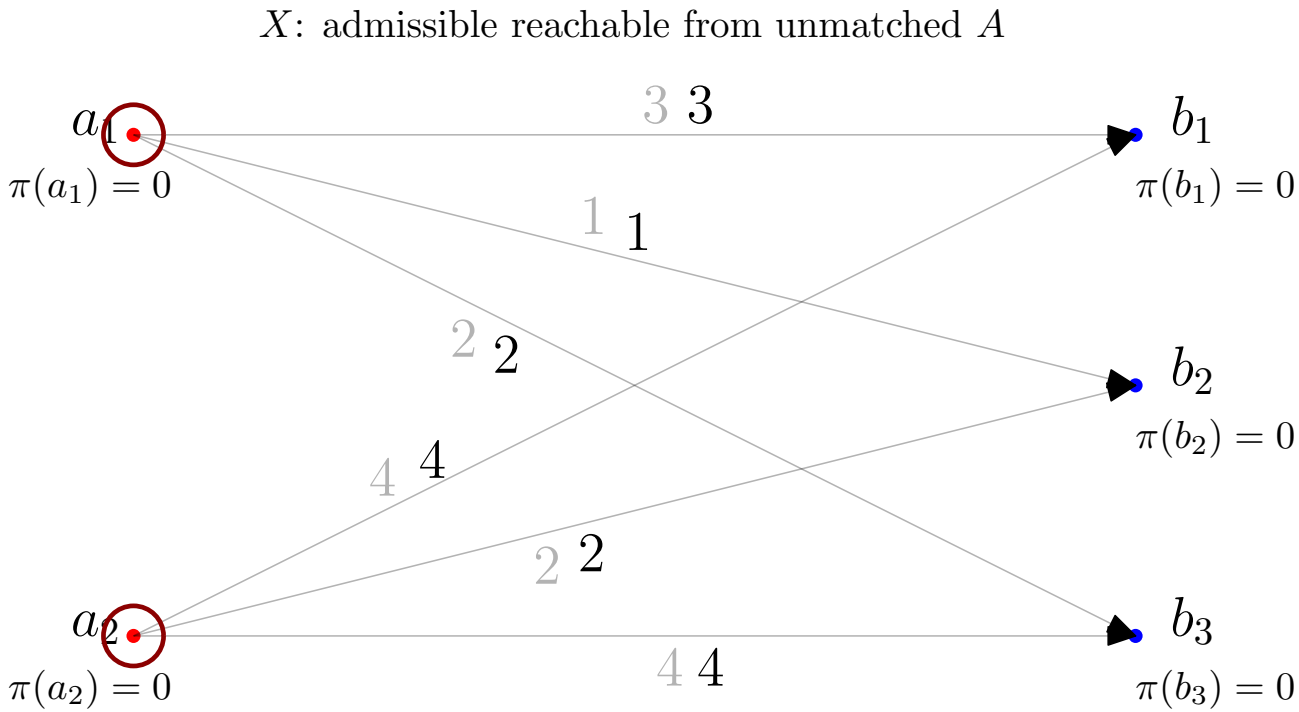


Hungarian algorithm:

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

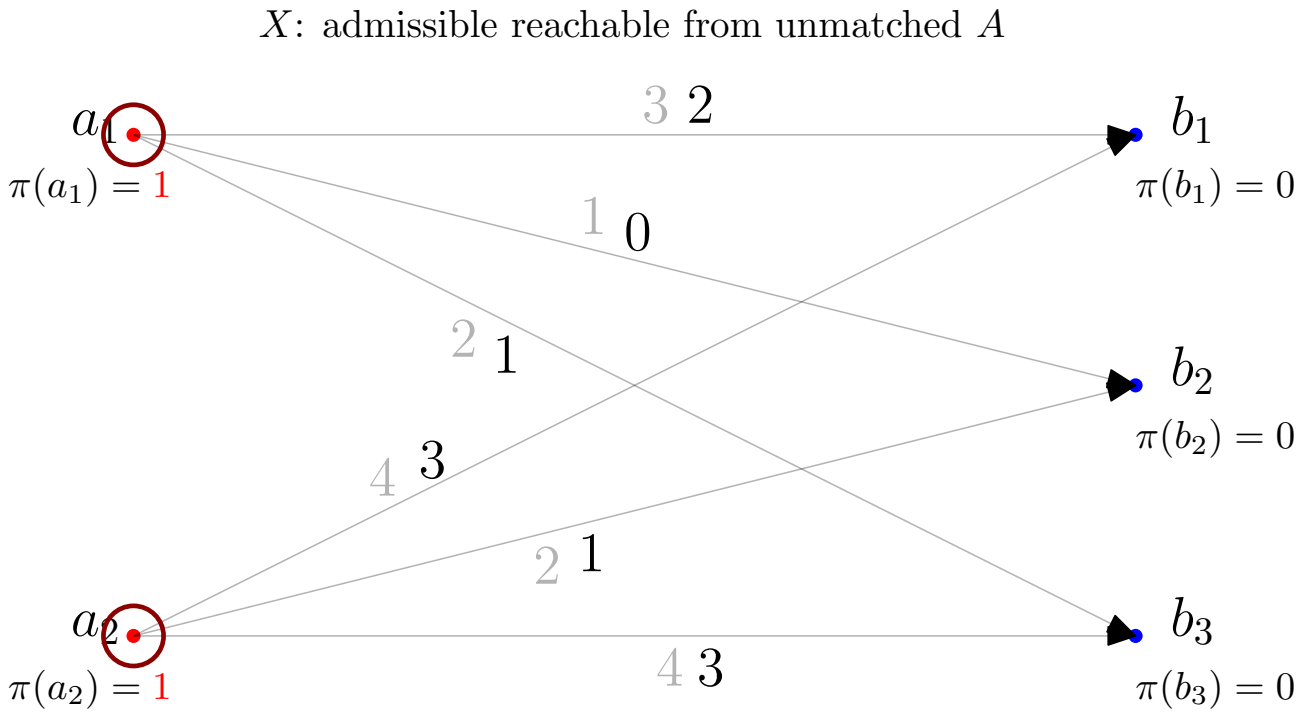
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



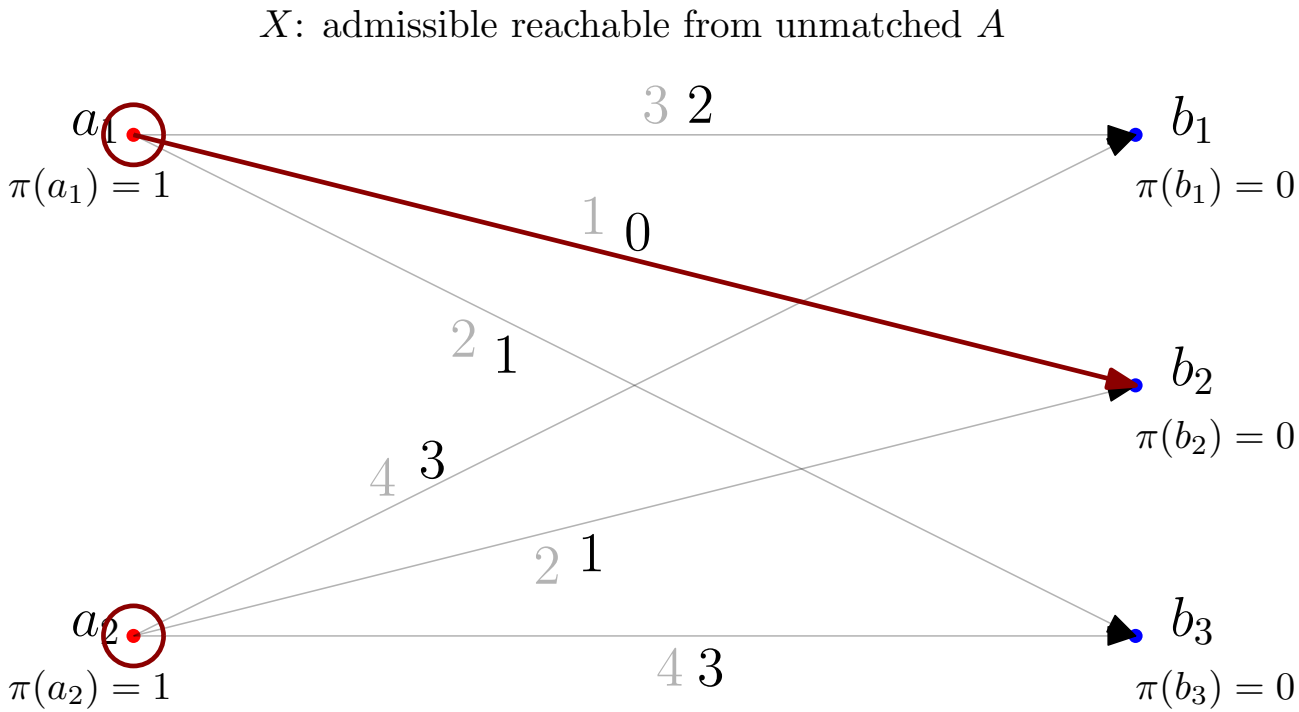
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



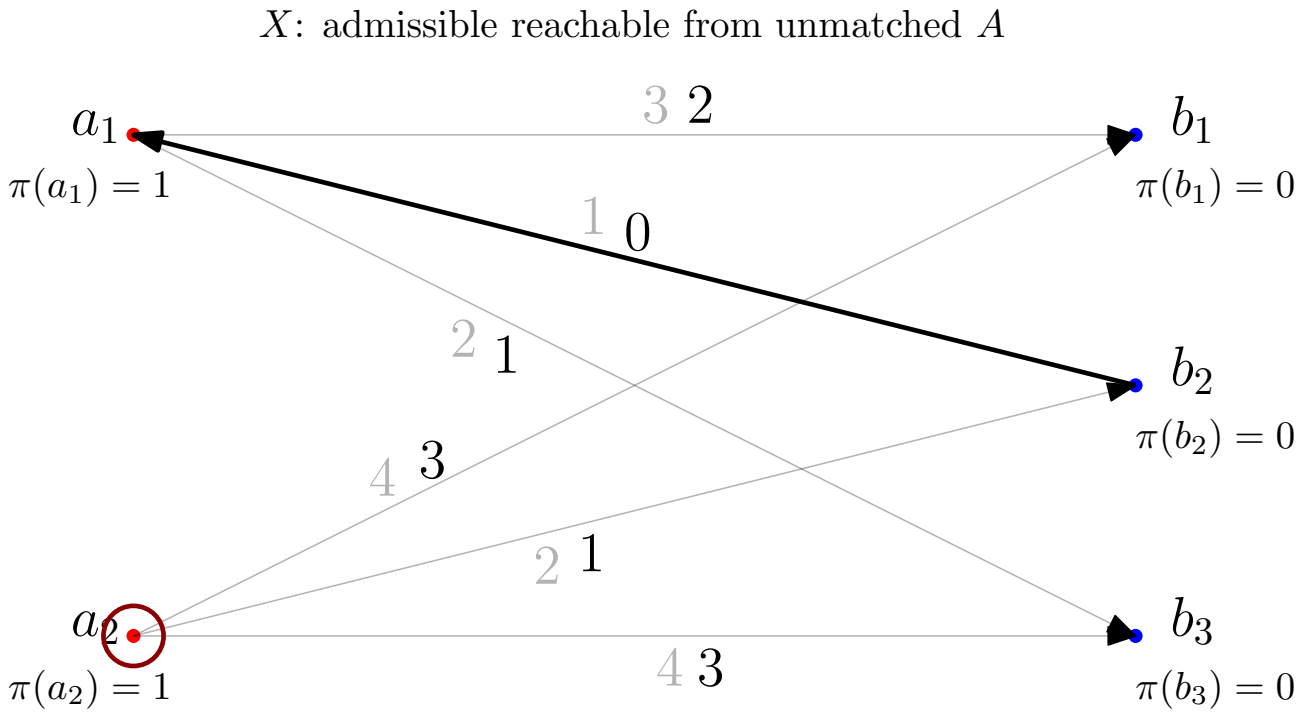
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



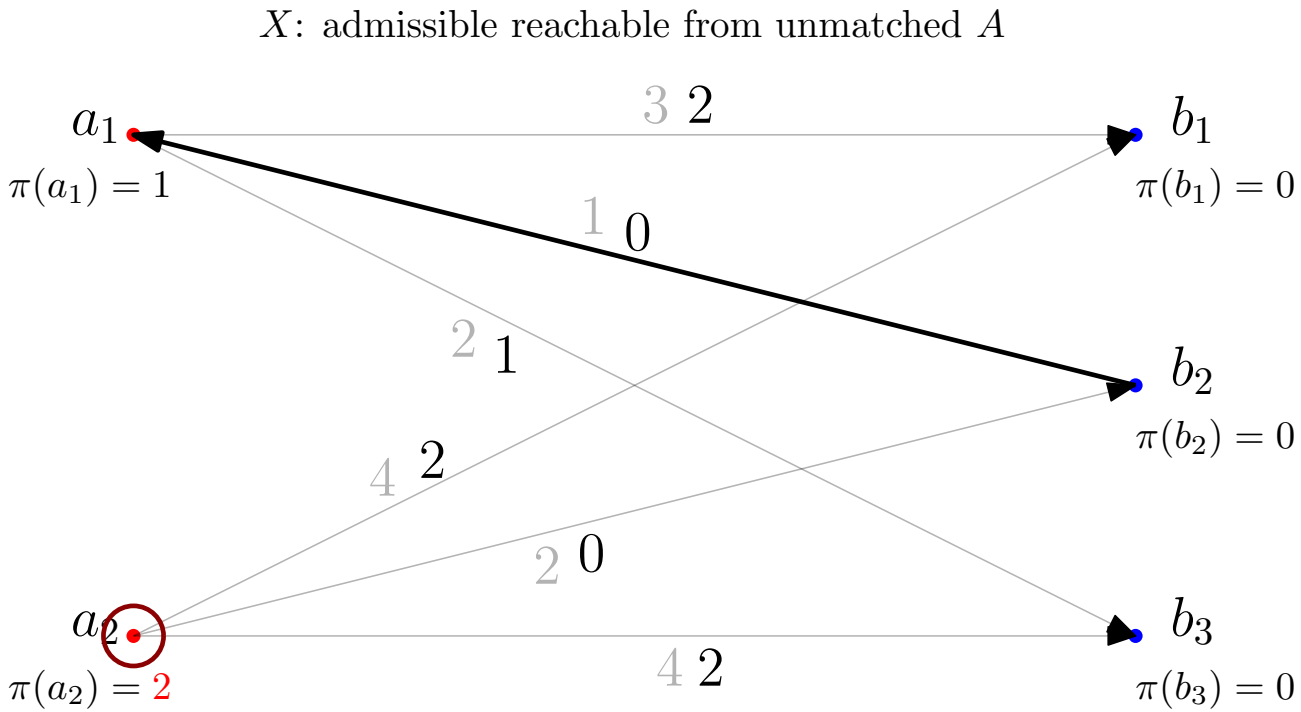
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



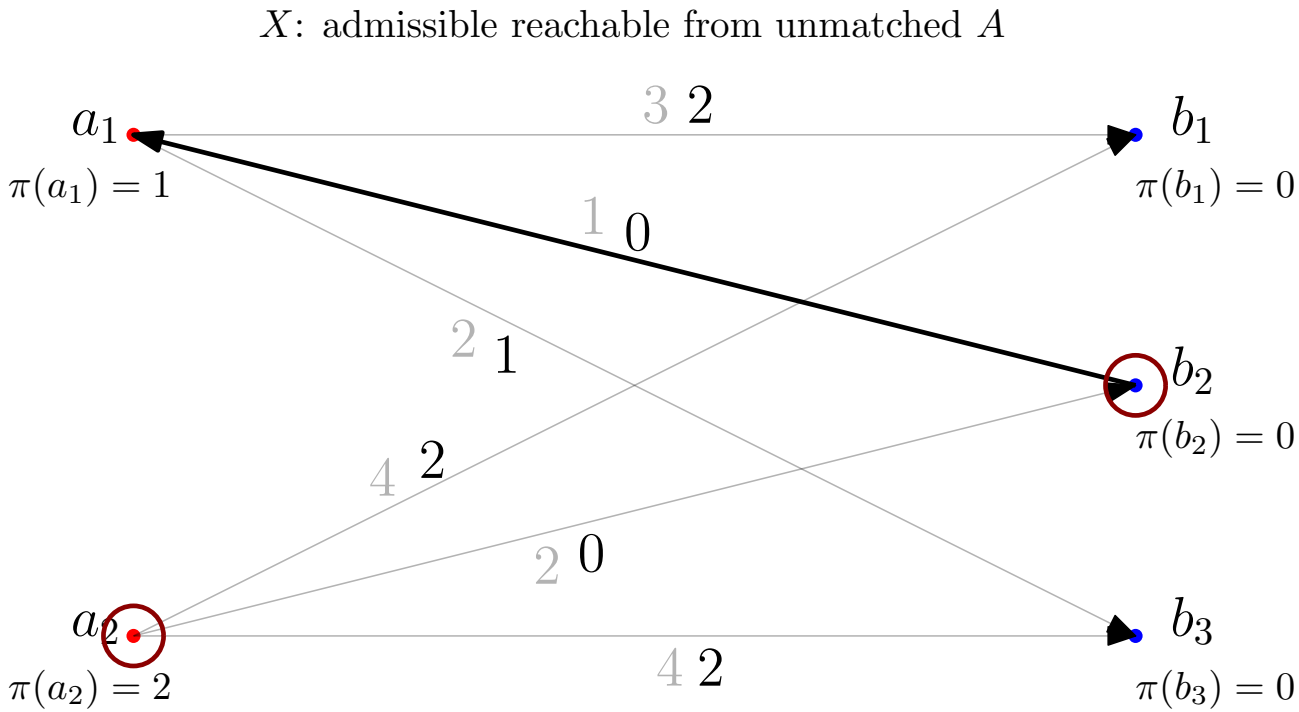
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



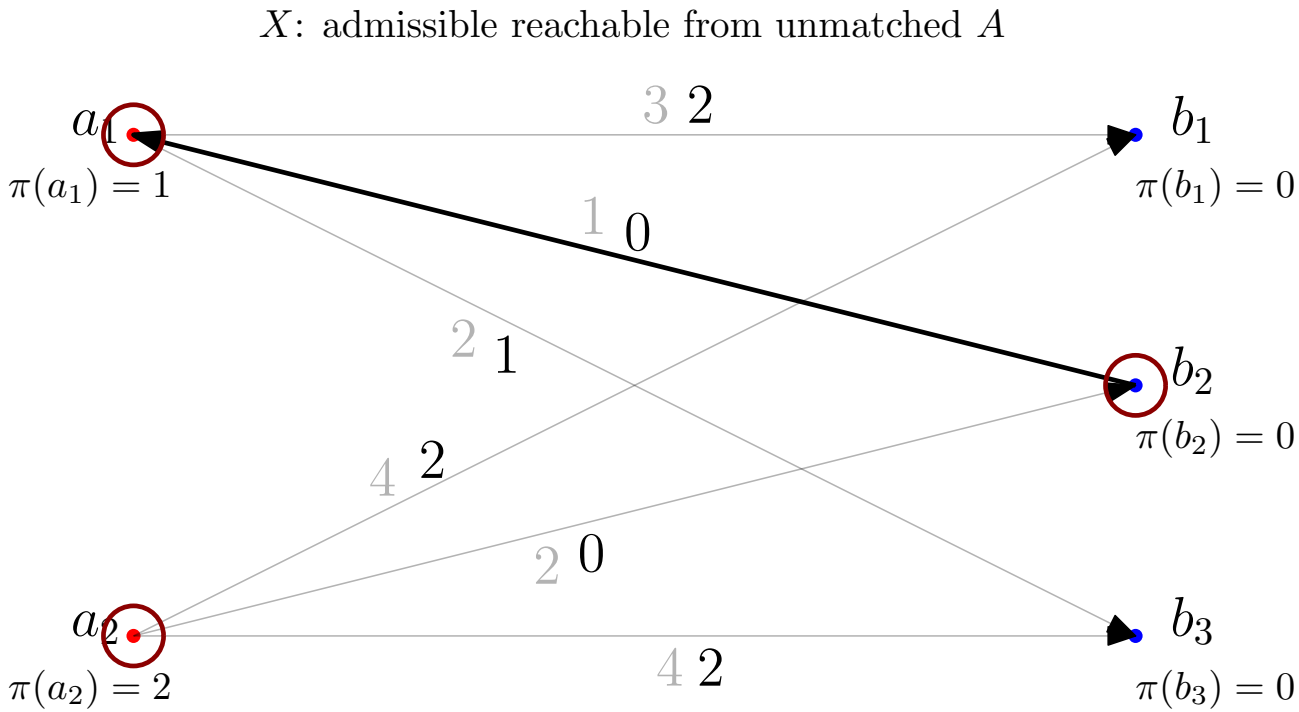
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



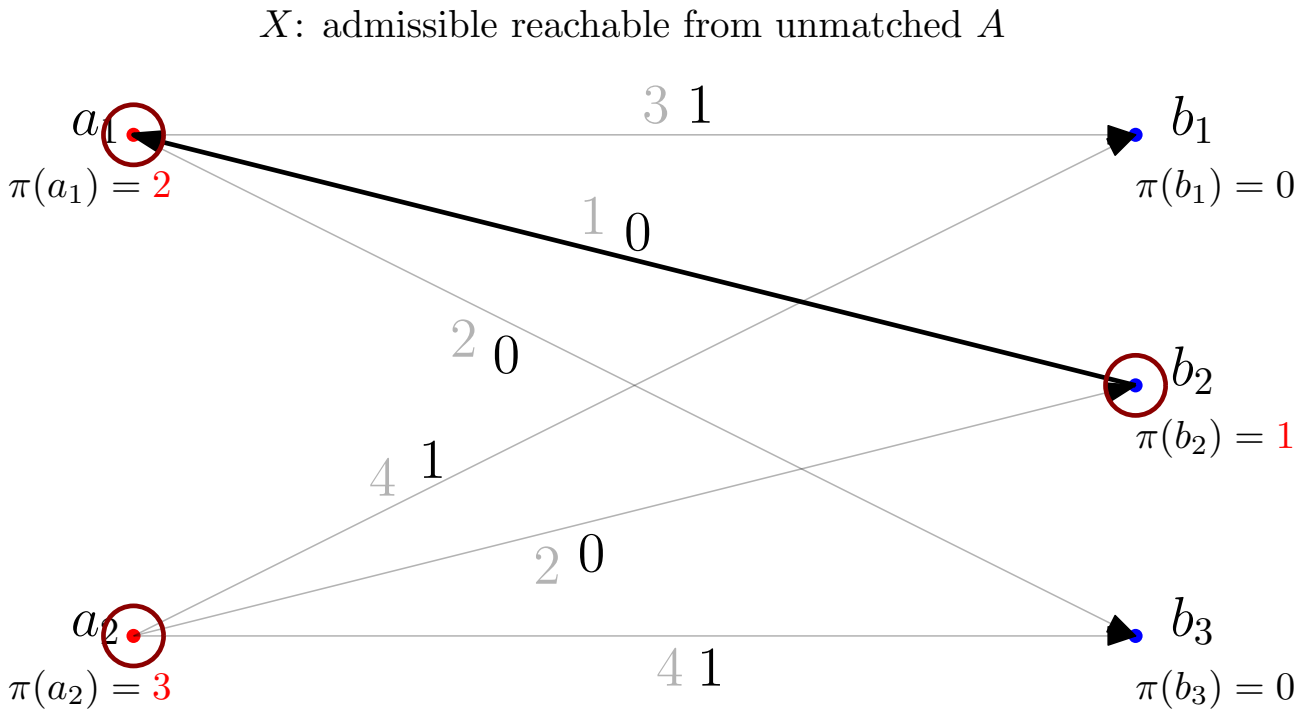
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



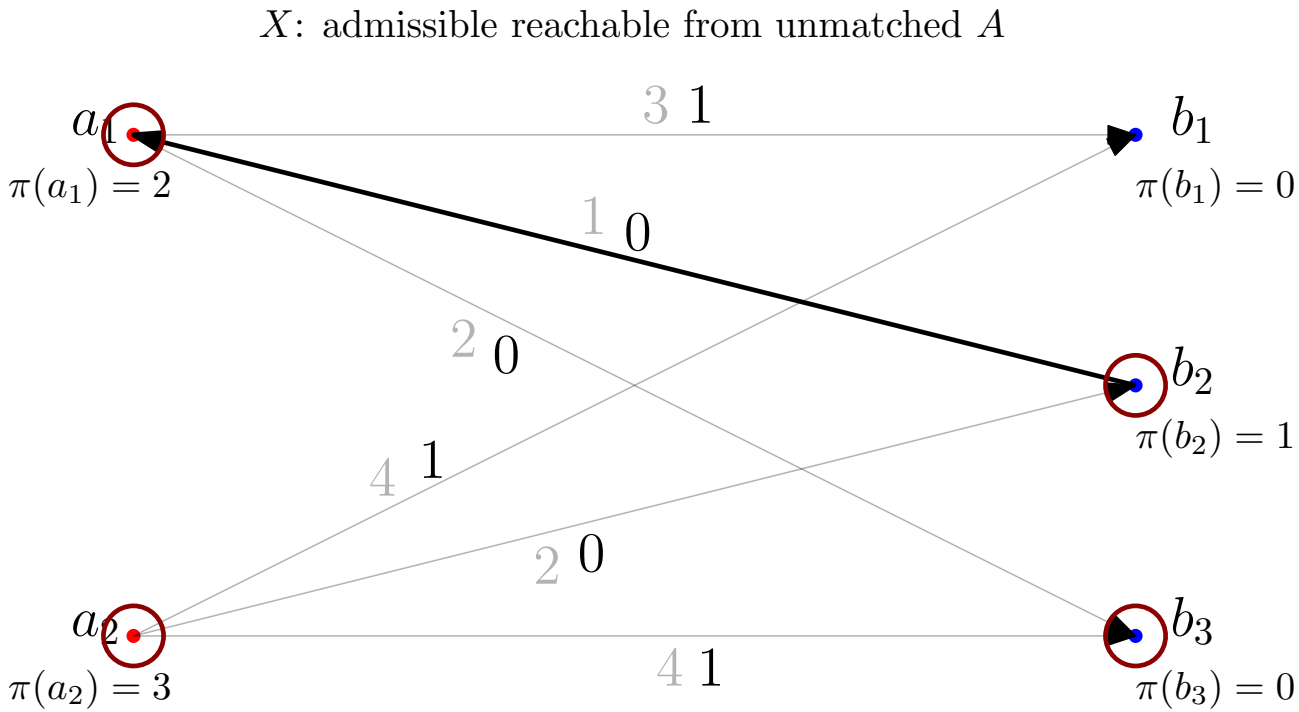
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



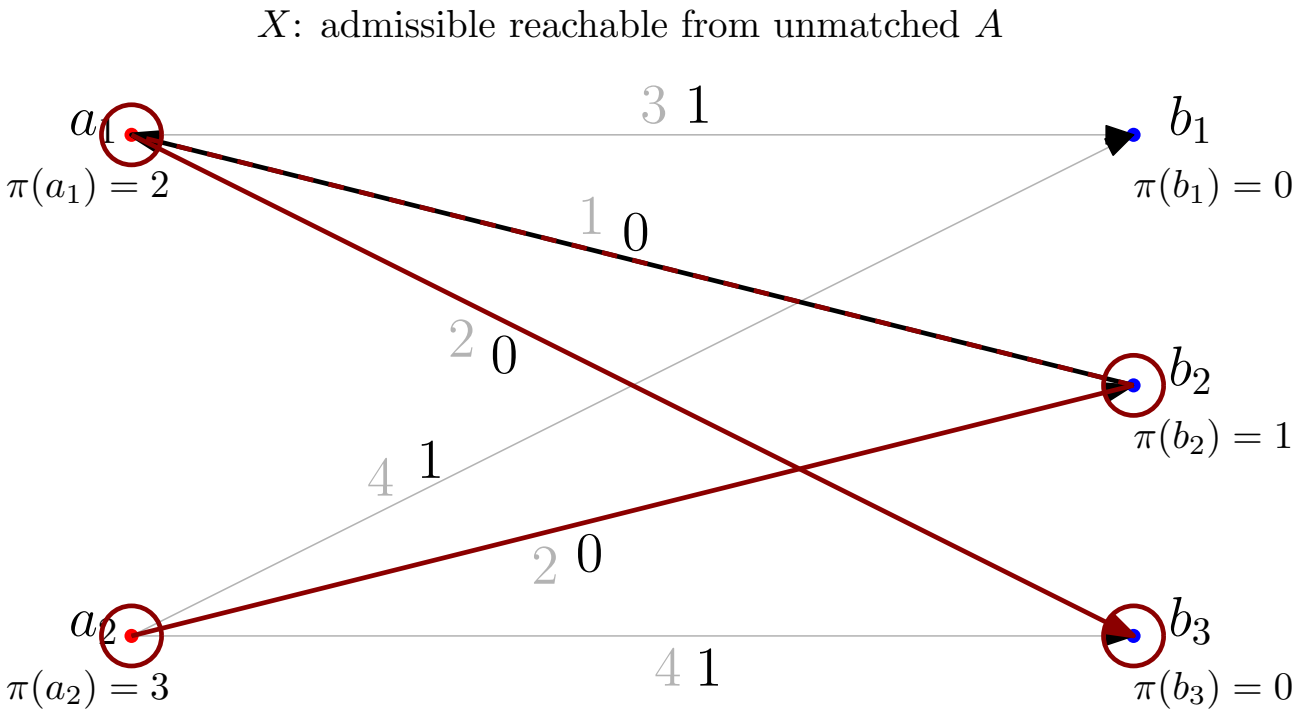
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



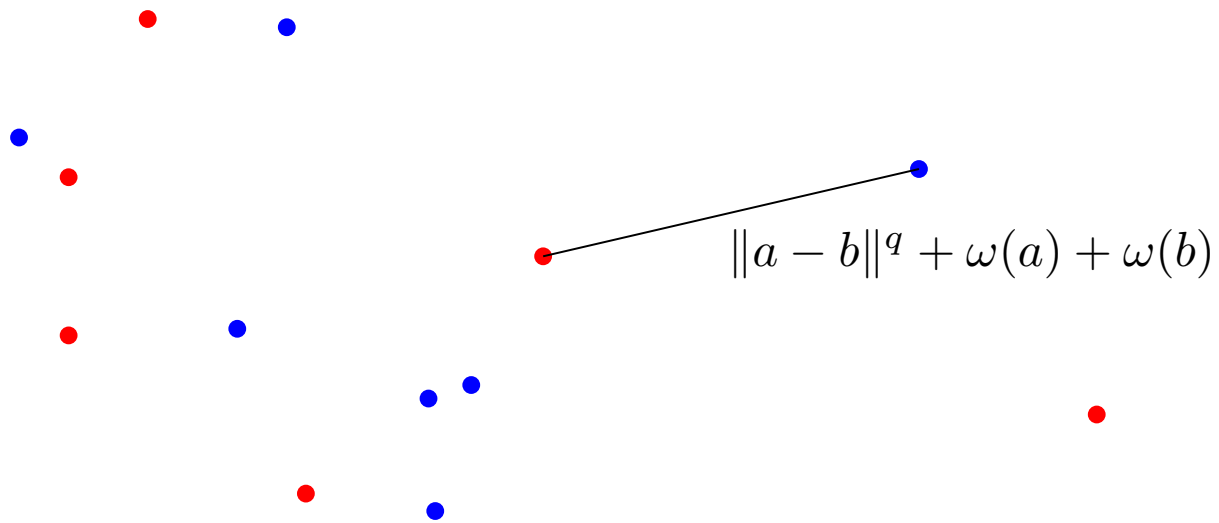
Hungarian search

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).



Hungarian search, with BCP

- "What is the minimum reduced cost residual arc leaving X?"
- For $B \to A$ residual arcs, min-heap over matching edges (at most k).
- For $A \to B$ residual arcs... bichromatic closest pair between $A \cap X$ and $B \setminus X$.



- [Kaplan *et al.* 17]: For points in $\mathbb{R}^2$, insert/delete in $O(\operatorname{polylog} n)$ time, query BCP in $O(\log^2 n)$ time.

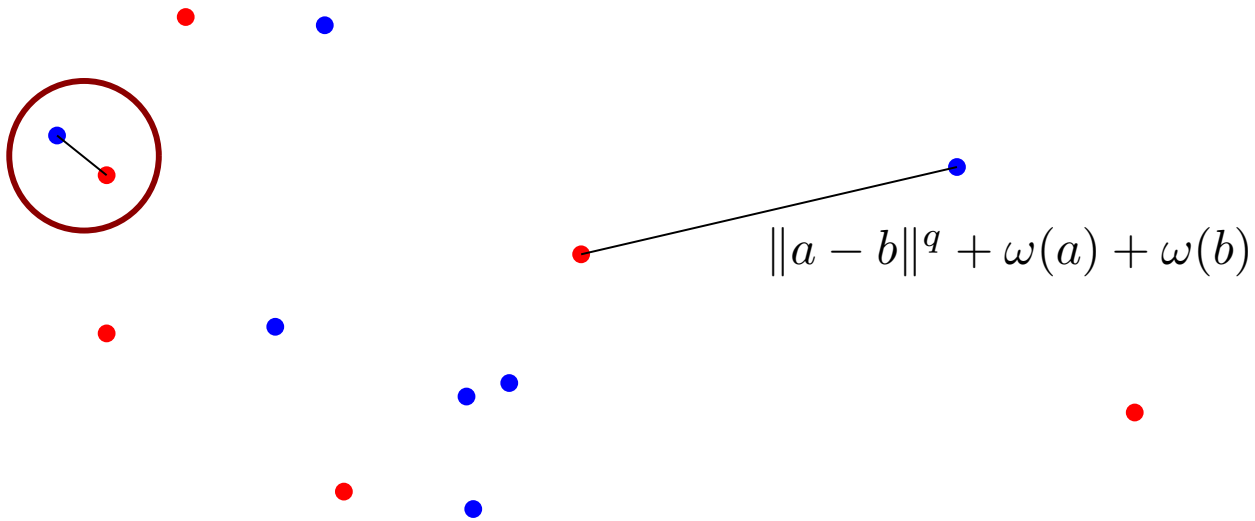
Hungarian search, with BCP

- "What is the minimum reduced cost residual arc leaving X?"
- For $B \to A$ residual arcs, min-heap over matching edges (at most k).
- For $A \to B$ residual arcs... bichromatic closest pair between $A \cap X$ and $B \setminus X$.

- [Kaplan *et al.* 17]: For points in $\mathbb{R}^2$, insert/delete in $O(\operatorname{polylog} n)$ time, query BCP in $O(\log^2 n)$ time.

Hungarian search, with BCP

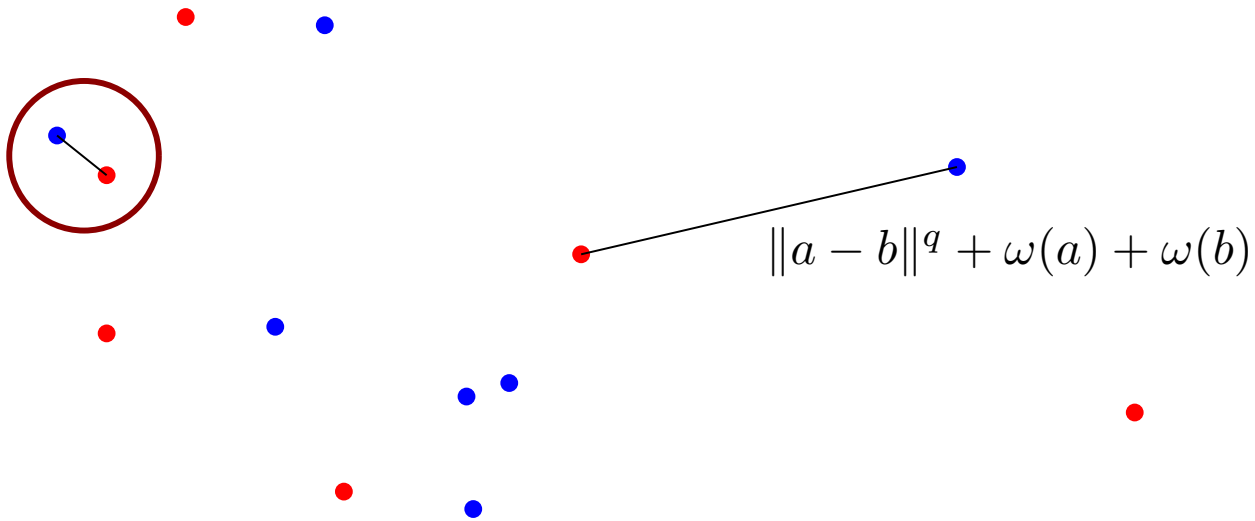
- "What is the minimum reduced cost residual arc leaving X?"
- For $B \to A$ residual arcs, min-heap over matching edges (at most k).
- For $A \to B$ residual arcs... bichromatic closest pair between $A \cap X$ and $B \setminus X$.



- [Kaplan *et al.* 17]: For points in $\mathbb{R}^2$, insert/delete in $O(\operatorname{polylog} n)$ time, query BCP in $O(\log^2 n)$ time.

Hungarian search, with BCP

- "What is the minimum reduced cost residual arc leaving X?"
- For $B \to A$ residual arcs, min-heap over matching edges (at most k).
- For $A \to B$ residual arcs... bichromatic closest pair between $A \cap X$ and $B \setminus X$.



- [Kaplan *et al.* 17]: For points in $\mathbb{R}^2$, insert/delete in $O(\operatorname{polylog} n)$ time, query BCP in $O(\log^2 n)$ time.

Problem: BCP initialization

- k augmentations
- Goal: $O(k \operatorname{polylog} n)$ time per augmentation
- $O(k)$ relaxations per Hungarian search (at most k matching edges)
- But, the initial BCP data structure of each Hungarian search has size $O(n)$.
- $\ldots O(kn \operatorname{polylog} n)$ time so far.

Problem: BCP initialization

- k augmentations
- Goal: $O(k \operatorname{polylog} n)$ time per augmentation
- $O(k)$ relaxations per Hungarian search (at most k matching edges)
- But, the initial BCP data structure of each Hungarian search has size $O(n)$.
- . . . $O(kn \operatorname{polylog} n)$ time so far.

Problem: BCP initialization

- k augmentations
- Goal: $O(k \operatorname{polylog} n)$ time per augmentation
- $O(k)$ relaxations per Hungarian search (at most k matching edges)
- But, the initial BCP data structure of each Hungarian search has size $O(n)$.
- . . . $O(kn \operatorname{polylog} n)$ time so far.

Problem: BCP initialization

- k augmentations
- Goal: $O(k \operatorname{polylog} n)$ time per augmentation
- $O(k)$ relaxations per Hungarian search (at most k matching edges)
- But, the initial BCP data structure of each Hungarian search has size $O(n)$.
- $\ldots O(kn \operatorname{polylog} n)$ time so far.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?

  Unfortunately, the BCP insert/delete bounds are amortized.

Rewinding

- Initial X is the set of unmatched A vertices.
- Initial X loses exactly one vertex after an augmentation.

1. Record the BCP operations we did during the Hungarian search, then rewind them to recover the initial BCP state.
2. Delete one more point from the BCP to get the next initial state.

- $O(k \operatorname{polylog} n)$ time to rewind.
- Build the whole BCP once ($O(n \operatorname{polylog} n)$), reuse every augmentation.
- Persistence?
  Unfortunately, the BCP insert/delete bounds are amortized.

Efficient potential updates

- During the Hungarian search, how do we have time to raise π for every $v \in X$?
- [Vaidya 89]: batch changes into a single variable δ.
  "Raise π for every $v \in X$ by α" replaced by $\delta \leftarrow \delta + \alpha$.
- When a point enters X we save it with $\gamma(v) \leftarrow \pi(v) - \delta$.
- At any point, $\pi(v) = \gamma(v) + \delta$ for $v \in X$.
- To use with rewinding, don't reset δ between augmentations.
- Summary: Build the BCP once ($O(n \operatorname{polylog} n)$). Each Hungarian search and augmentation takes $O(k \operatorname{polylog} n)$ time.
  $O((n + k^2) \operatorname{polylog} n)$ time total.

Efficient potential updates

- During the Hungarian search, how do we have time to raise π for every $v \in X$?
- [Vaidya 89]: batch changes into a single variable δ.
  "Raise π for every $v \in X$ by α" replaced by $\delta \leftarrow \delta + \alpha$.

- When a point enters X we save it with $\gamma(v) \leftarrow \pi(v) - \delta$.
- At any point, $\pi(v) = \gamma(v) + \delta$ for $v \in X$.

- To use with rewinding, don't reset δ between augmentations.

- Summary: Build the BCP once ($O(n \operatorname{polylog} n)$). Each Hungarian search and augmentation takes $O(k \operatorname{polylog} n)$ time.
  $O((n + k^2) \operatorname{polylog} n)$ time total.

Efficient potential updates

- During the Hungarian search, how do we have time to raise π for every $v \in X$?
- [Vaidya 89]: batch changes into a single variable δ.
  "Raise π for every $v \in X$ by α" replaced by $\delta \leftarrow \delta + \alpha$.
- When a point enters X we save it with $\gamma(v) \leftarrow \pi(v) - \delta$.
- At any point, $\pi(v) = \gamma(v) + \delta$ for $v \in X$.
- To use with rewinding, don't reset δ between augmentations.
- Summary: Build the BCP once ($O(n \operatorname{polylog} n)$). Each Hungarian search and augmentation takes $O(k \operatorname{polylog} n)$ time.
  $O((n + k^2) \operatorname{polylog} n)$ time total.

Efficient potential updates

- During the Hungarian search, how do we have time to raise π for every $v \in X$?
- [Vaidya 89]: batch changes into a single variable δ.
  "Raise π for every $v \in X$ by α" replaced by $\delta \leftarrow \delta + \alpha$.

- When a point enters X we save it with $\gamma(v) \leftarrow \pi(v) - \delta$.
- At any point, $\pi(v) = \gamma(v) + \delta$ for $v \in X$.

- To use with rewinding, don't reset δ between augmentations.

- Summary: Build the BCP once ($O(n \operatorname{polylog} n)$). Each Hungarian search and augmentation takes $O(k \operatorname{polylog} n)$ time.
  $O((n + k^2) \operatorname{polylog} n)$ time total.

Efficient potential updates

- During the Hungarian search, how do we have time to raise π for every $v \in X$?
- [Vaidya 89]: batch changes into a single variable δ.
  "Raise π for every $v \in X$ by α" replaced by $\delta \leftarrow \delta + \alpha$.

- When a point enters X we save it with $\gamma(v) \leftarrow \pi(v) - \delta$.
- At any point, $\pi(v) = \gamma(v) + \delta$ for $v \in X$.

- To use with rewinding, don't reset δ between augmentations.

- Summary: Build the BCP once ($O(n \operatorname{polylog} n)$). Each Hungarian search and augmentation takes $O(k \operatorname{polylog} n)$ time.
  $O((n + k^2) \operatorname{polylog} n)$ time total.

Efficient potential updates

- During the Hungarian search, how do we have time to raise π for every $v \in X$?
- [Vaidya 89]: batch changes into a single variable δ.
  "Raise π for every $v \in X$ by α" replaced by $\delta \leftarrow \delta + \alpha$.

- When a point enters X we save it with $\gamma(v) \leftarrow \pi(v) - \delta$.
- At any point, $\pi(v) = \gamma(v) + \delta$ for $v \in X$.

- To use with rewinding, don't reset δ between augmentations.

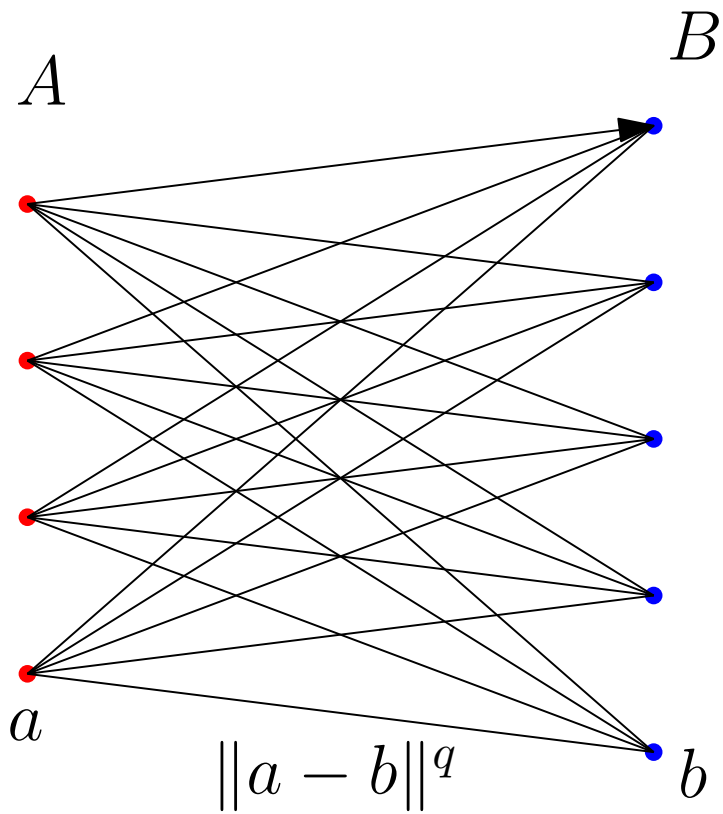
- Summary: Build the BCP once ($O(n \operatorname{polylog} n)$). Each Hungarian search and augmentation takes $O(k \operatorname{polylog} n)$ time.
  $O((n + k^2) \operatorname{polylog} n)$ time total.

Our results

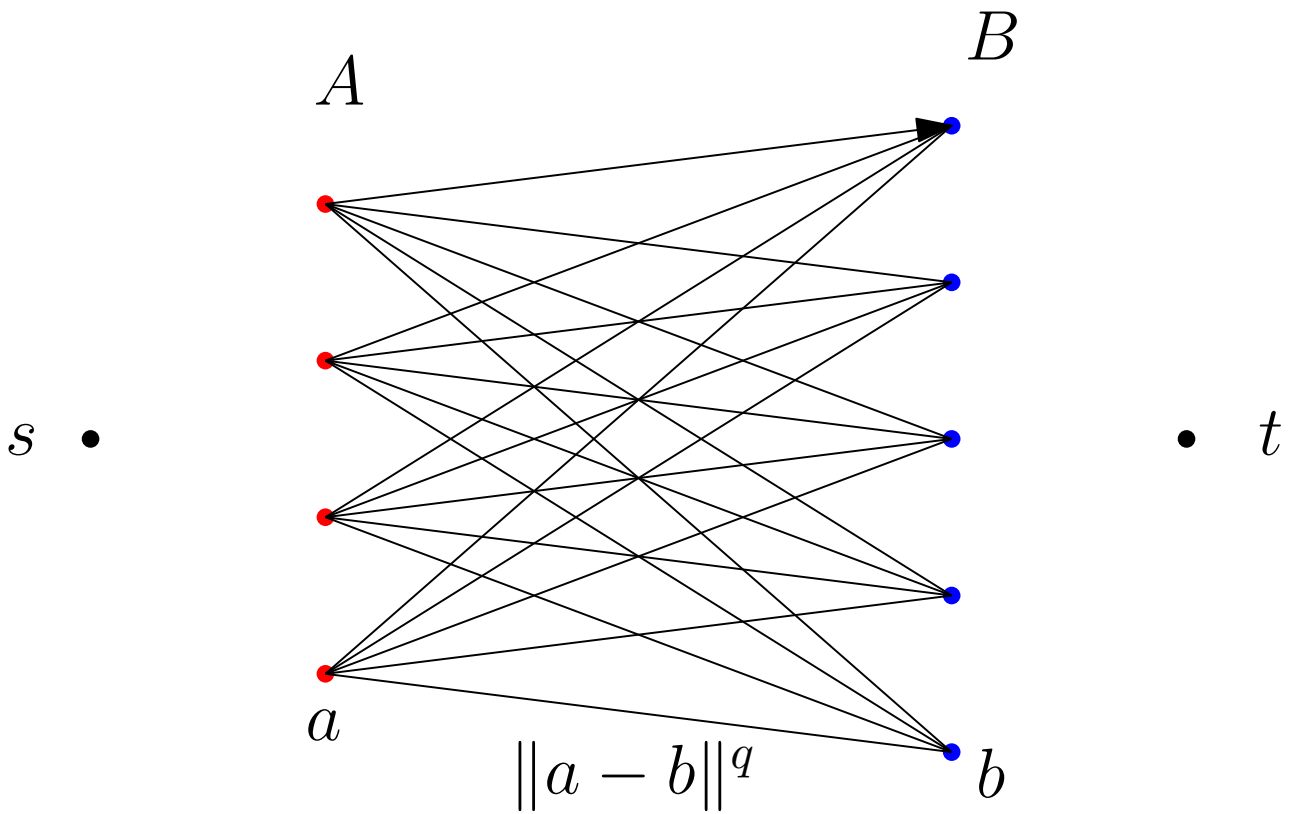
New:

1. Hungarian algorithm: $O((n + k^2)\,\mathrm{polylog}(n))$ time, exact.
2. Cost-scaling: $O((n + k\sqrt{k})\,\mathrm{polylog}(n))$ time per cost scale, $O(\log(n^q/\varepsilon))$ scales, $(1+\varepsilon)$-approx.
3. Geometric Transportation: $O(\min(n^2, nr^{3/2})\,\mathrm{polylog}(n))$ time, exact.

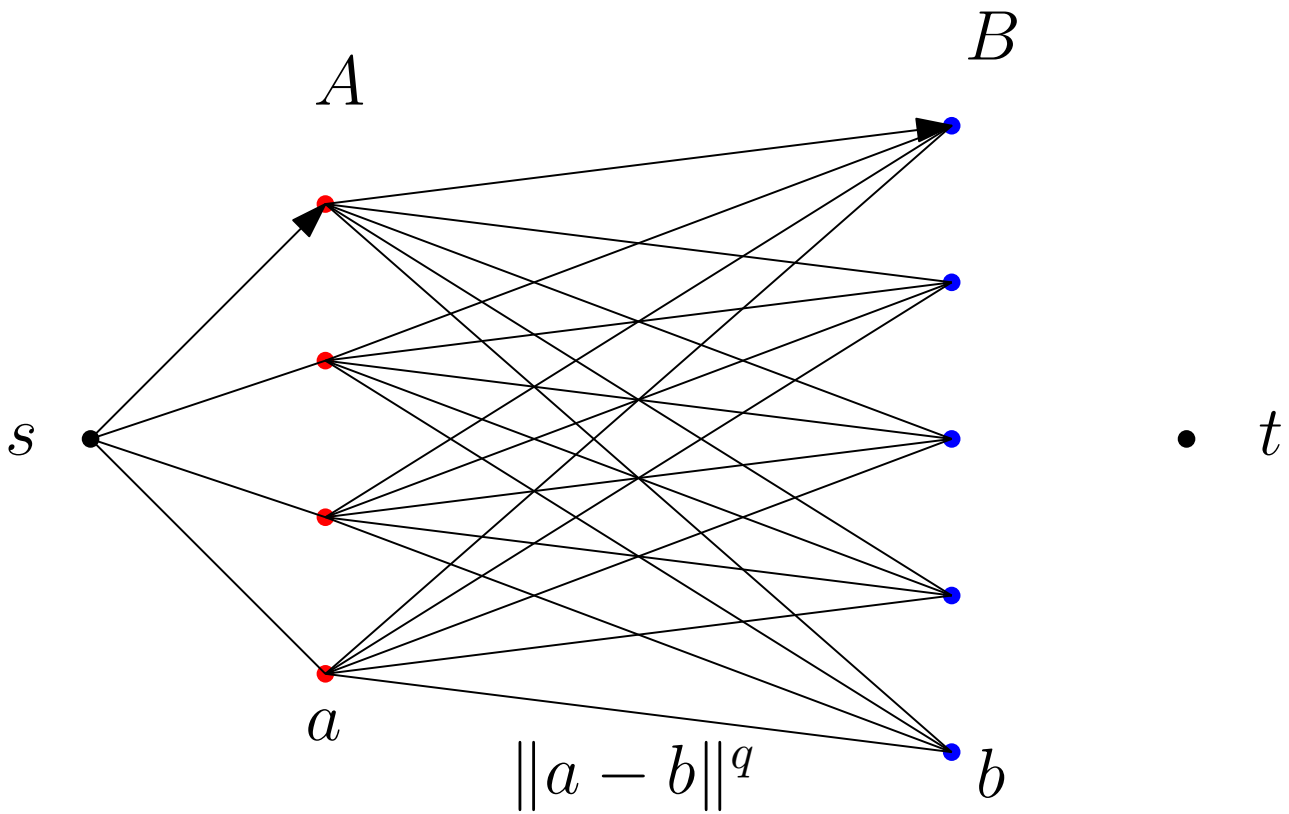
Geometric partial matching as unit-capacity min-cost flow



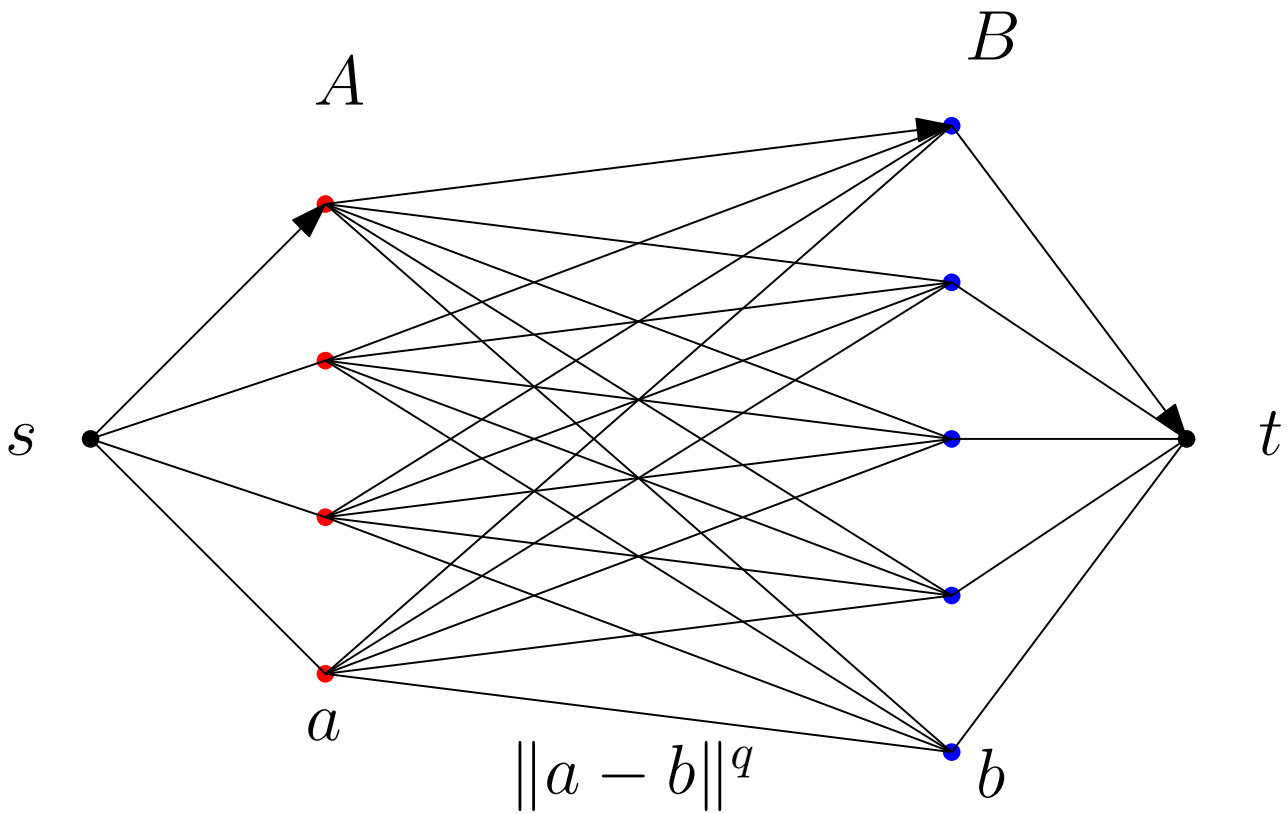
Geometric partial matching as unit-capacity min-cost flow



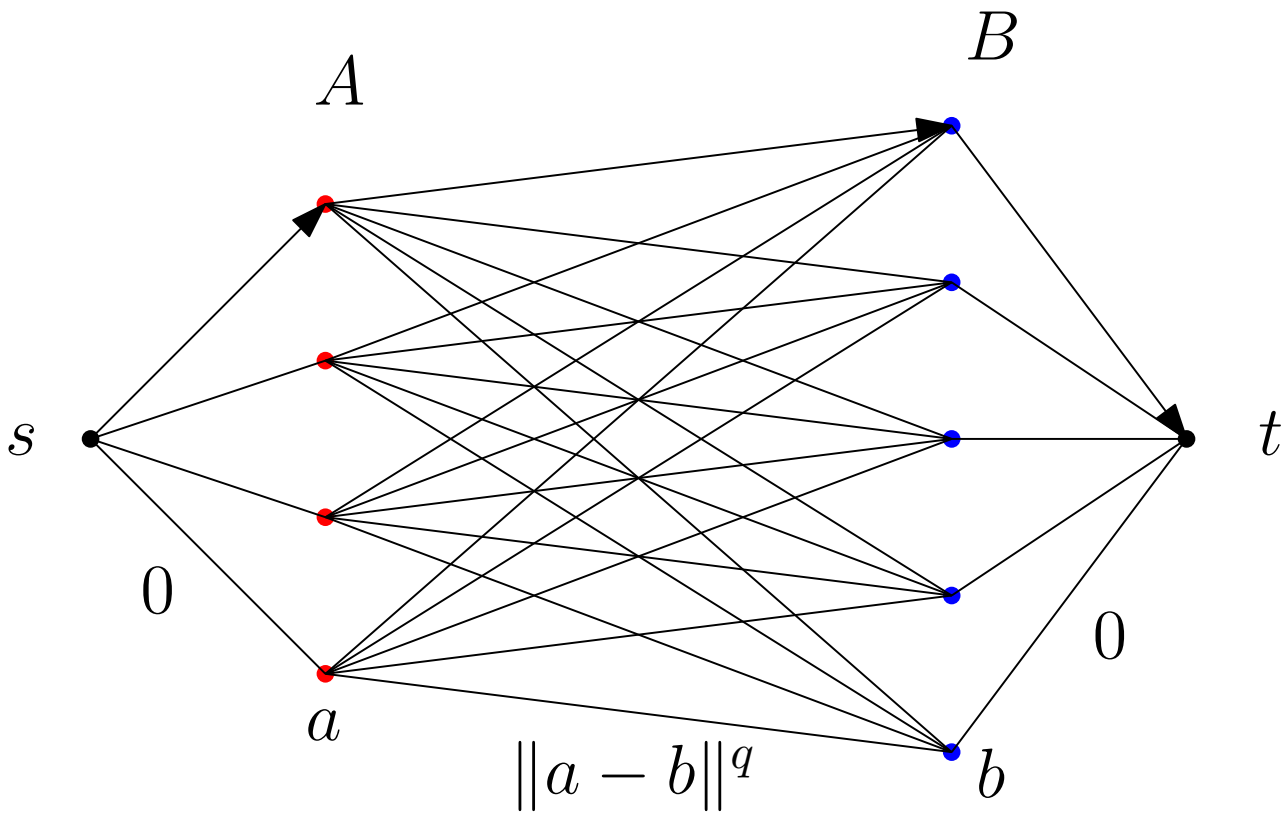
Geometric partial matching as unit-capacity min-cost flow



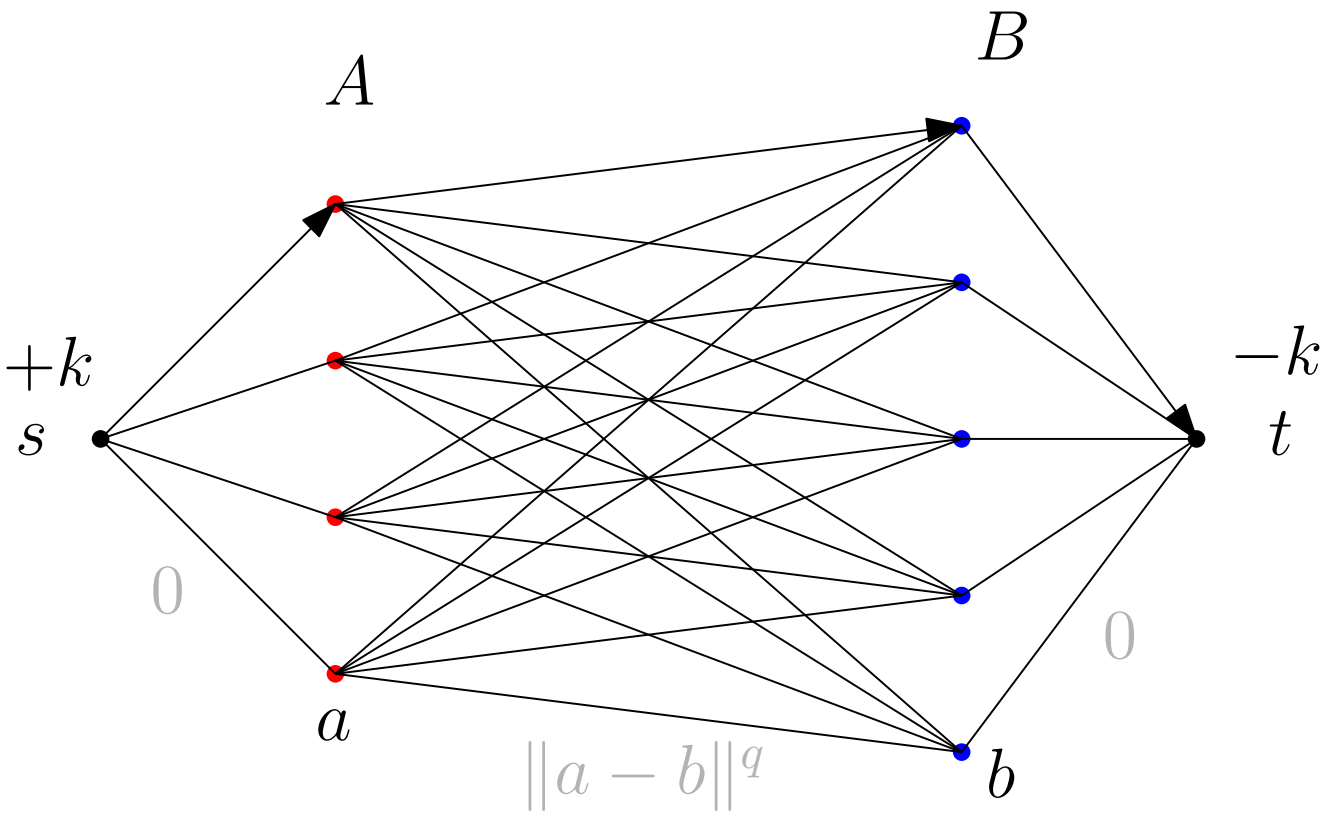
Geometric partial matching as unit-capacity min-cost flow



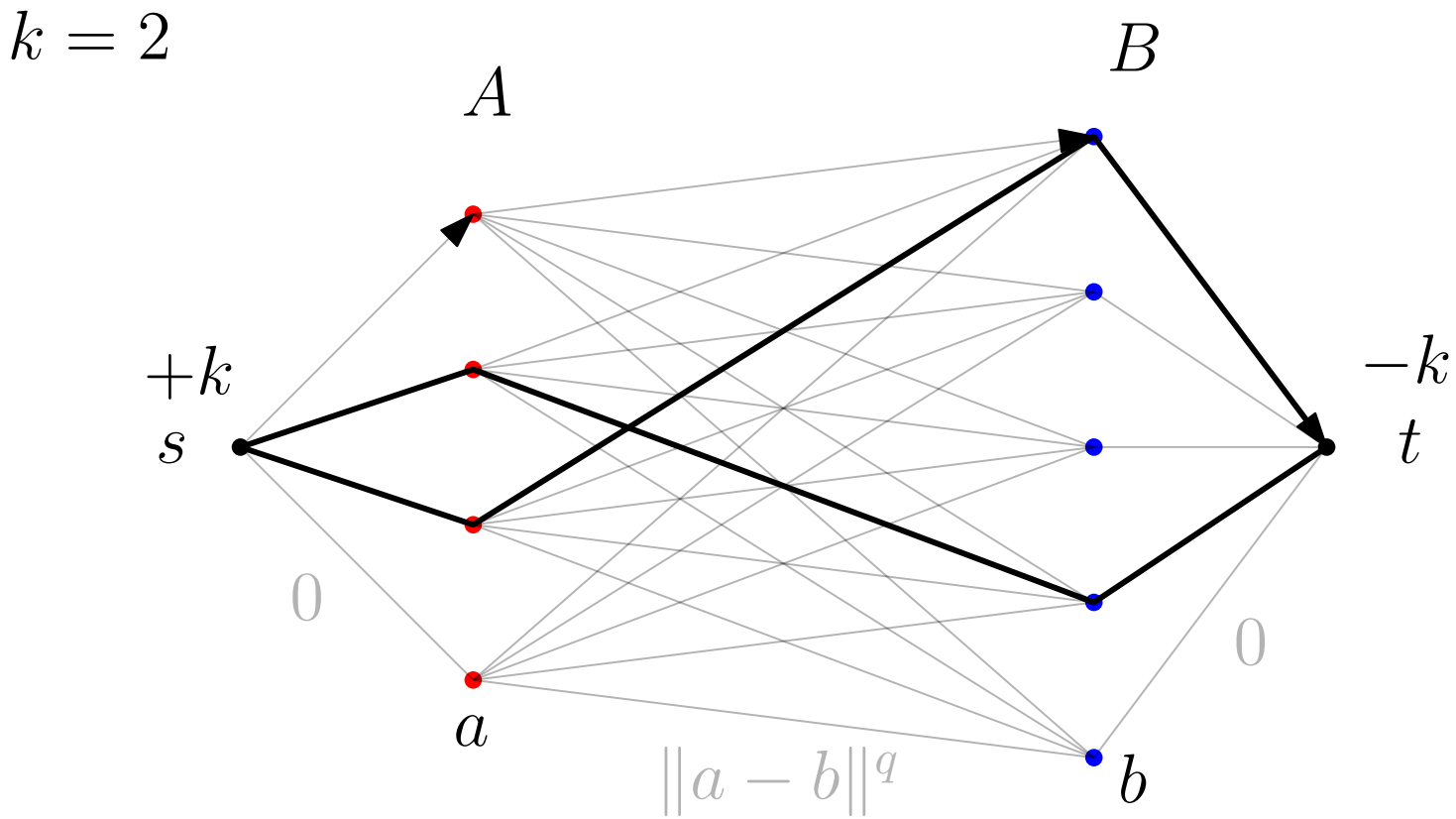
Geometric partial matching as unit-capacity min-cost flow



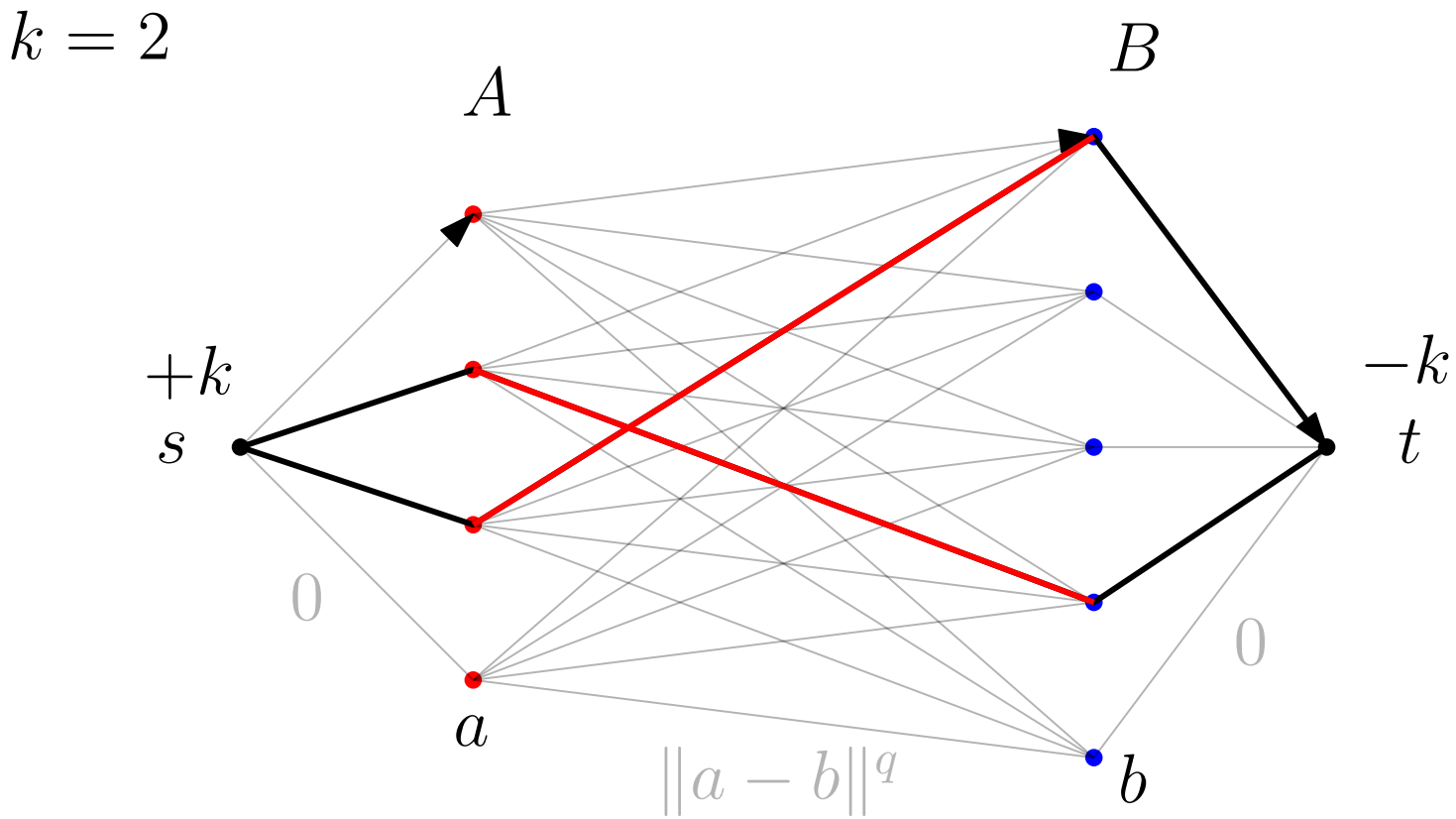
Geometric partial matching as unit-capacity min-cost flow



Geometric partial matching as unit-capacity min-cost flow



Geometric partial matching as unit-capacity min-cost flow



Partial matching by cost-scaling

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- θ-optimal: $c_\pi(v, w) \geq -\theta$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).

Lemma

A θ-optimal circulation for the partial matching network has cost at most $c(f^*) + 6k\theta$.

- Find θ-optimal circulations for shrinking values of scale θ.

Partial matching by cost-scaling

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- θ-optimal: $c_\pi(v, w) \geq -\theta$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).

Lemma

A θ-optimal circulation for the partial matching network has cost at most $c(f^*) + 6k\theta$.

- Find θ-optimal circulations for shrinking values of scale θ.

Partial matching by cost-scaling

- Reduced cost: $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w)$
- Keep π feasible — $c_\pi(v, w) \geq 0$ for all residual arcs (v, w).
- θ-optimal: $c_\pi(v, w) \geq -\theta$ for all residual arcs (v, w).
- Admissible: $c_\pi(v, w) \leq 0$ for residual arc (v, w).

Lemma

A θ-optimal circulation for the partial matching network has cost at most $c(f^*) + 6k\theta$.

- Find θ-optimal circulations for shrinking values of scale θ.

Partial matching by cost-scaling

Given a 2θ-optimal circulation from the previous scale,

1. Find a θ-optimal pseudoflow f with $O(k)$ excess.
2. Refine f into a θ-optimal circulation.

Lemma

Can reduce $(1+\varepsilon)$-approx. geometric partial matching to executing $O(\log(n^q/\varepsilon))$ scales of the cost-scaling algorithm.

- Focus on solving each scale efficiently.

Partial matching by cost-scaling

Given a 2θ-optimal circulation from the previous scale,

1. Find a θ-optimal pseudoflow f with $O(k)$ excess.
2. Refine f into a θ-optimal circulation.

Lemma

Can reduce $(1+\varepsilon)$-approx. geometric partial matching to executing $O(\log(n^q/\varepsilon))$ scales of the cost-scaling algorithm.

- Focus on solving each scale efficiently.

Partial matching by cost-scaling

Given a 2θ-optimal circulation from the previous scale,

1. Find a θ-optimal pseudoflow f with $O(k)$ excess.
2. Refine f into a θ-optimal circulation.

Lemma

Can reduce $(1+\varepsilon)$-approx. geometric partial matching to executing $O(\log(n^q/\varepsilon))$ scales of the cost-scaling algorithm.

- Focus on solving each scale efficiently.

Partial matching by cost-scaling

Given a 2θ-optimal circulation from the previous scale,

1. Find a θ-optimal pseudoflow f with $O(k)$ excess.
2. Refine f into a θ-optimal circulation.

Lemma

Can reduce $(1+\varepsilon)$-approx. geometric partial matching to executing $O(\log(n^q/\varepsilon))$ scales of the cost-scaling algorithm.

- Focus on solving each scale efficiently.

Partial matching by cost-scaling

Given a 2θ-optimal circulation from the previous scale,

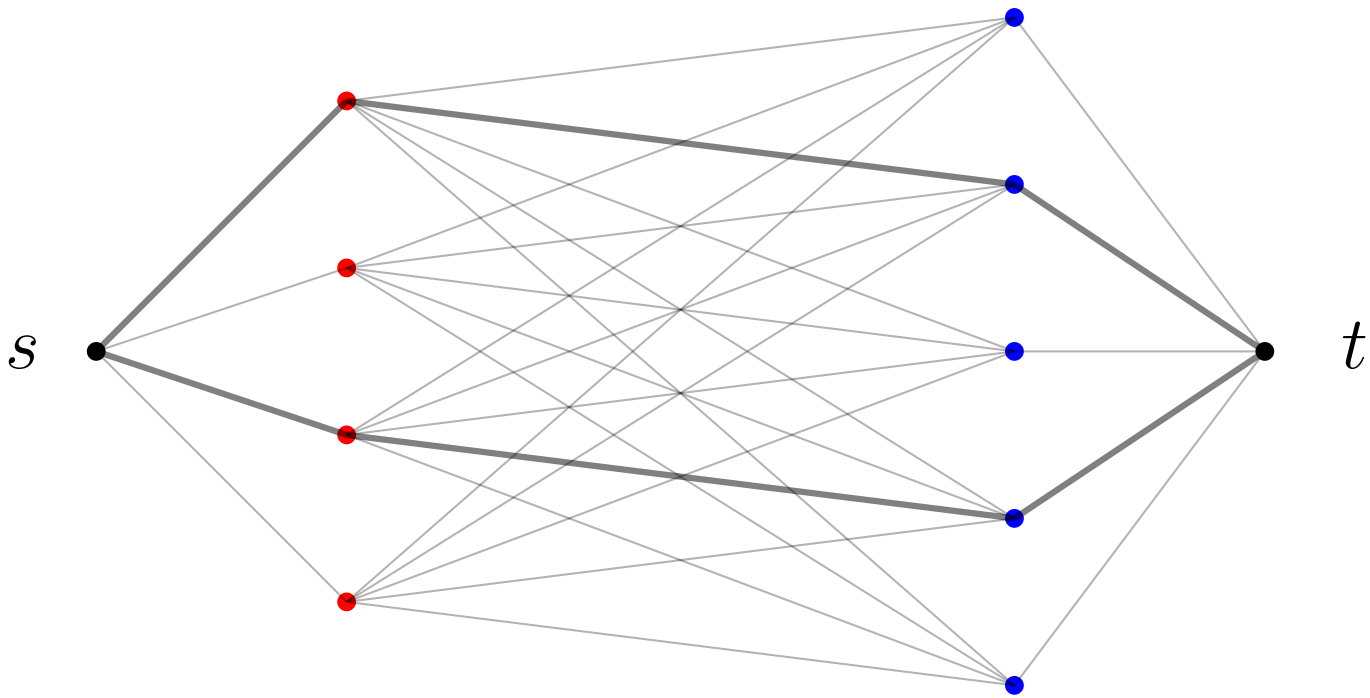
1. Find a θ-optimal pseudoflow f with $O(k)$ excess.
2. Refine f into a θ-optimal circulation.

Lemma

Can reduce $(1+\varepsilon)$-approx. geometric partial matching to executing $O(\log(n^q/\varepsilon))$ scales of the cost-scaling algorithm.

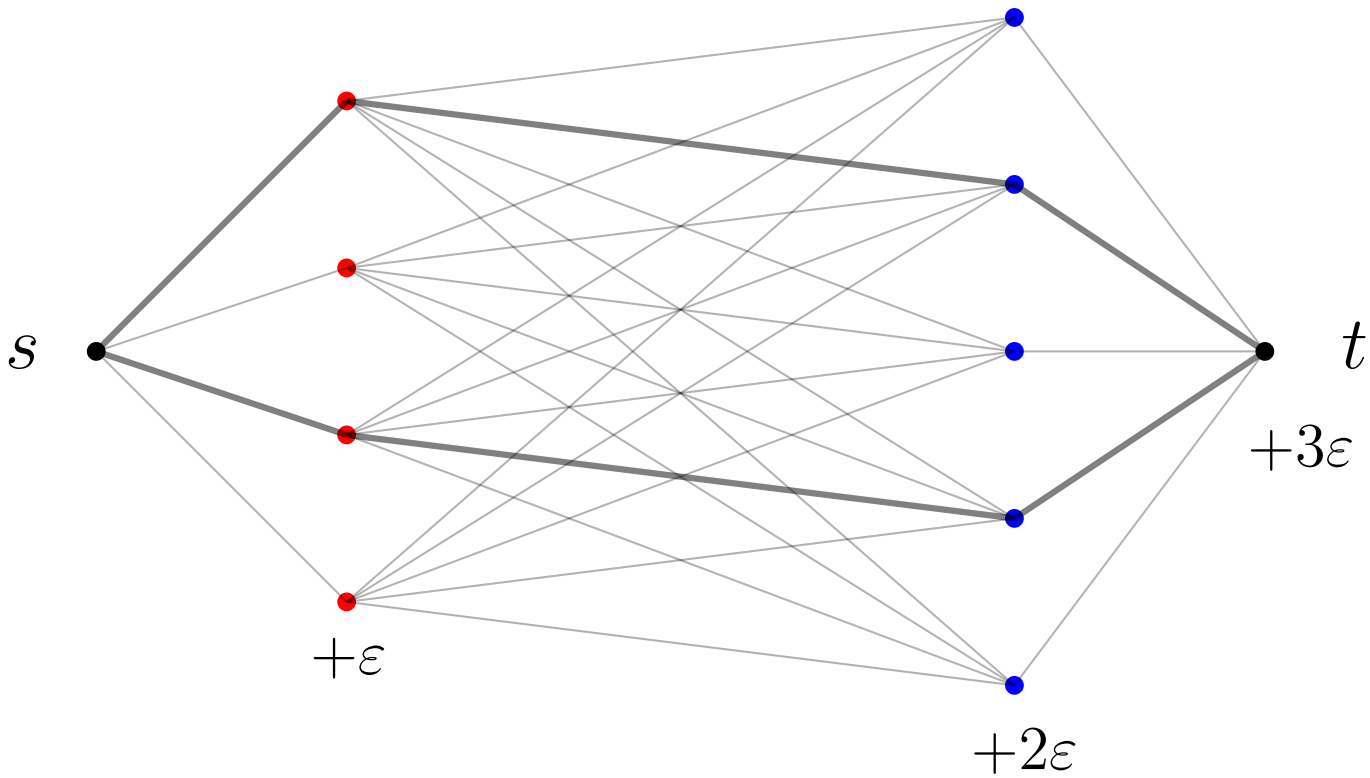
- Focus on solving each scale efficiently.

Initial θ-optimal pseudoflow of a scale



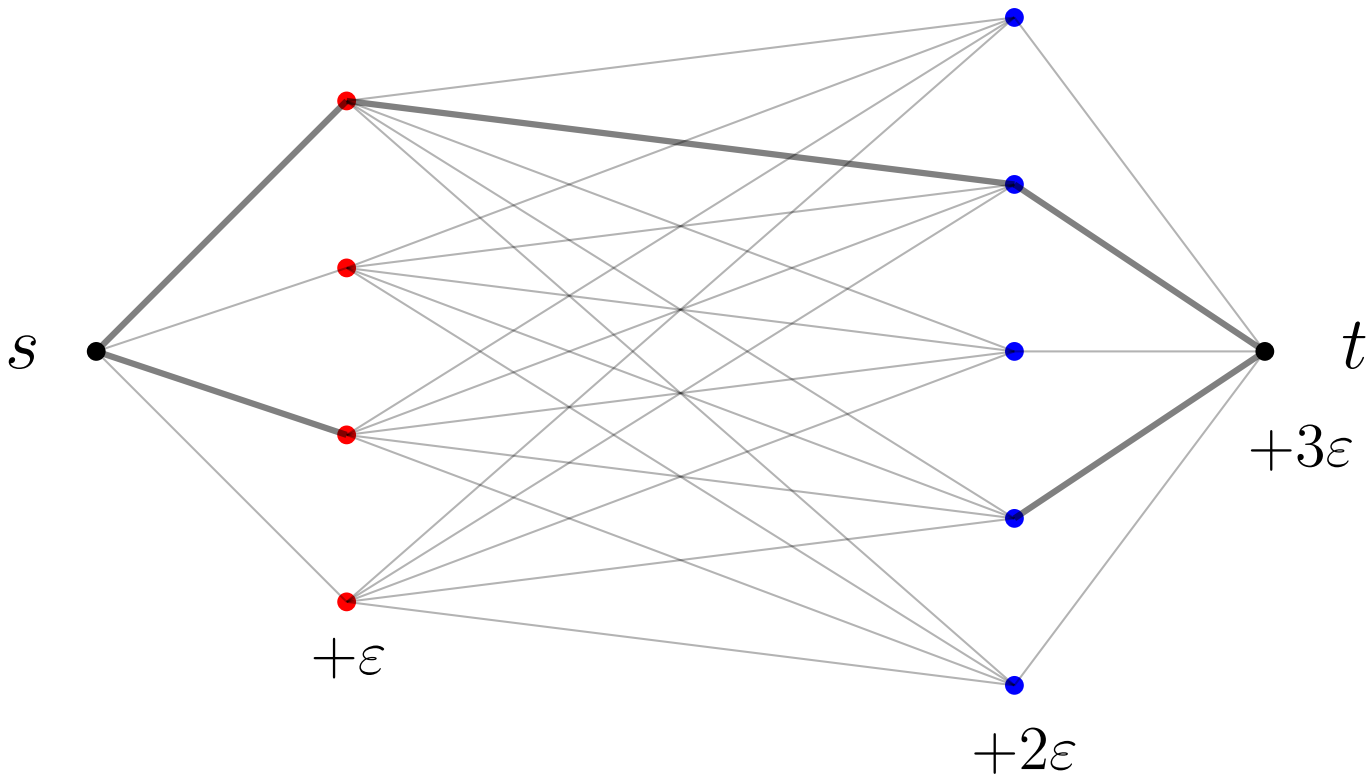
- Input circulation: $c_\pi(v, w) \geq -2\theta$ for all residual arcs.
- Forward arcs have reduced cost raised by $+\theta$ — must be θ-optimal.
- Only $O(k)$ reverse arcs, so check each one manually.

Initial θ-optimal pseudoflow of a scale



- Input circulation: $c_\pi(v, w) \geq -2\theta$ for all residual arcs.
- Forward arcs have reduced cost raised by $+\theta$ — must be θ-optimal.
- Only $O(k)$ reverse arcs, so check each one manually.

Initial θ-optimal pseudoflow of a scale



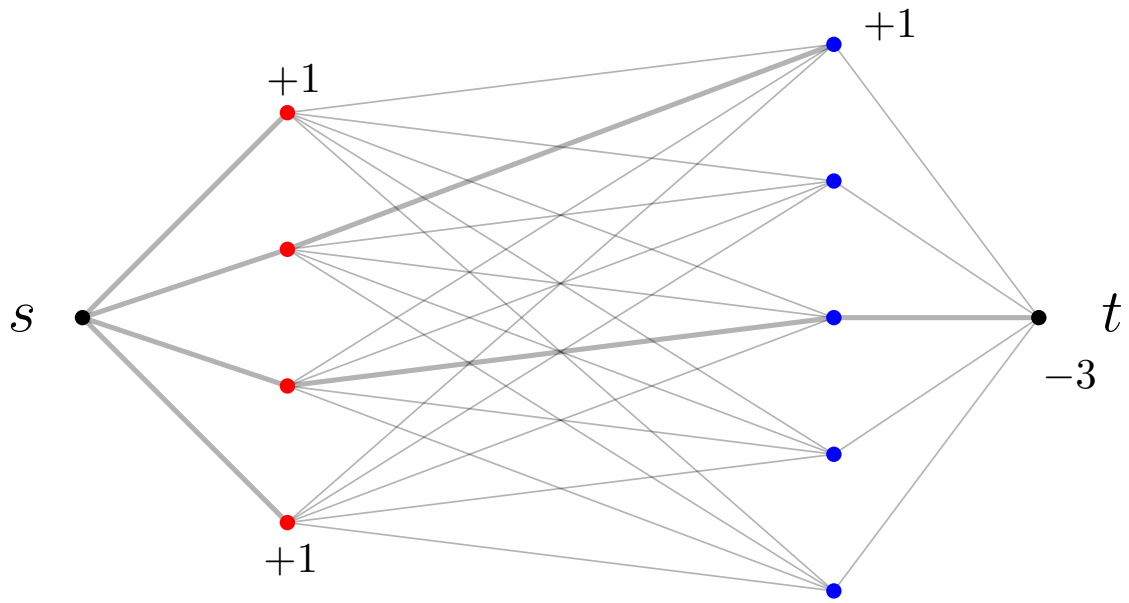
- Input circulation: $c_\pi(v, w) \geq -2\theta$ for all residual arcs.
- Forward arcs have reduced cost raised by $+\theta$ — must be θ-optimal.
- Only $O(k)$ reverse arcs, so check each one manually.

Refinement by blocking flows

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.

   Dijkstra-like search changing π in units of θ until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

   Augment by an admissible blocking flow g.



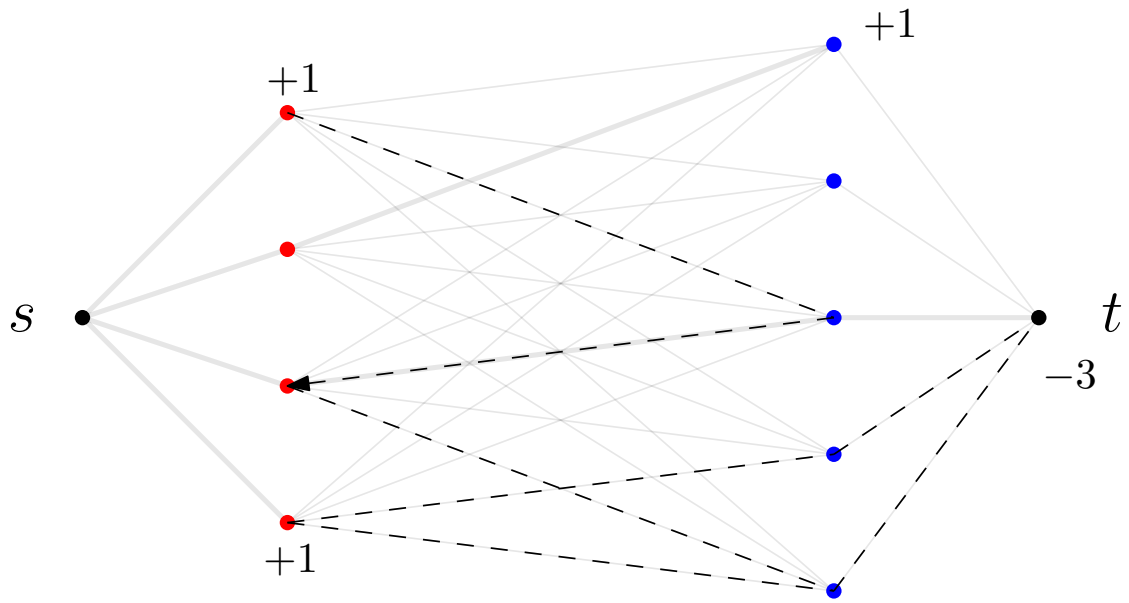
- Find by DFS through the admissible residual arcs.

Refinement by blocking flows

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.

   Dijkstra-like search changing π in units of θ until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

   Augment by an admissible blocking flow g.



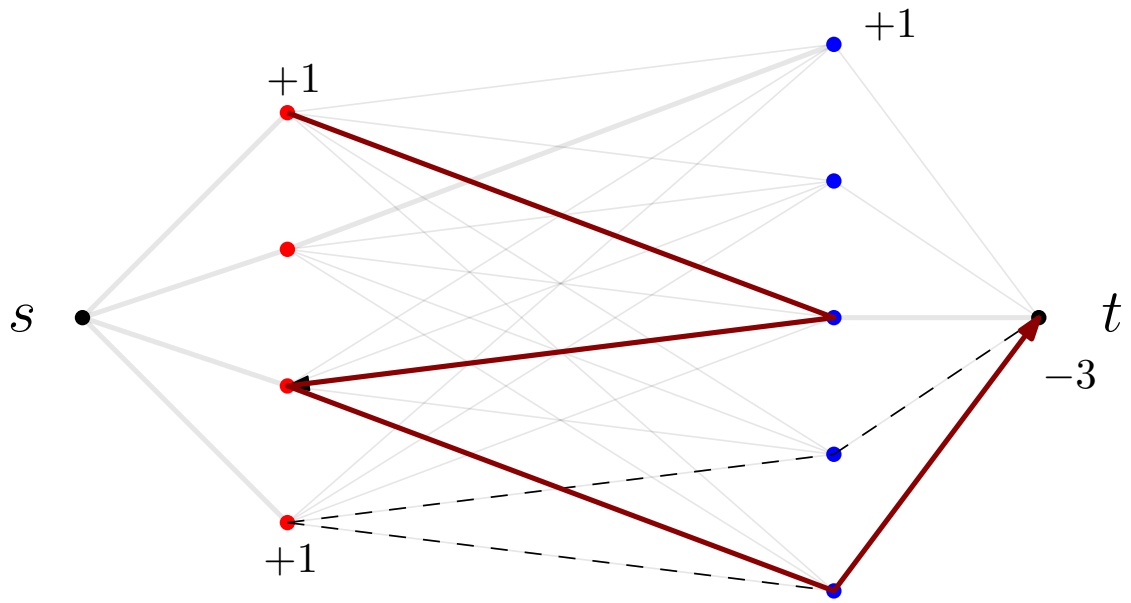
- Find by DFS through the admissible residual arcs.

Refinement by blocking flows

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.

   Dijkstra-like search changing π in units of θ until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

   Augment by an admissible blocking flow g.



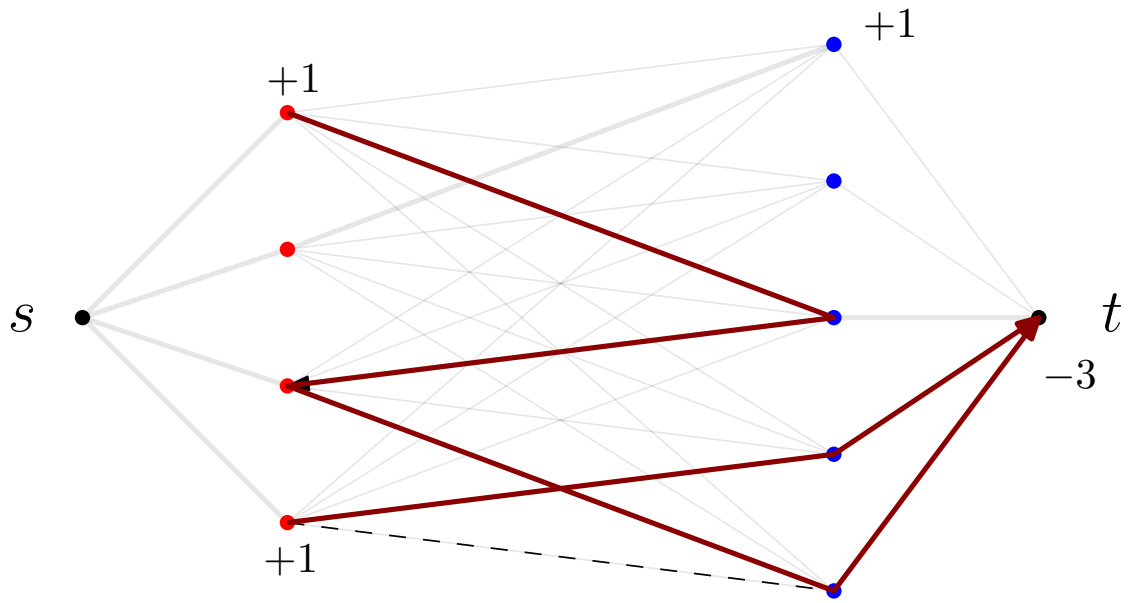
- Find by DFS through the admissible residual arcs.

Refinement by blocking flows

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.

   Dijkstra-like search changing π in units of θ until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

   Augment by an admissible blocking flow g.



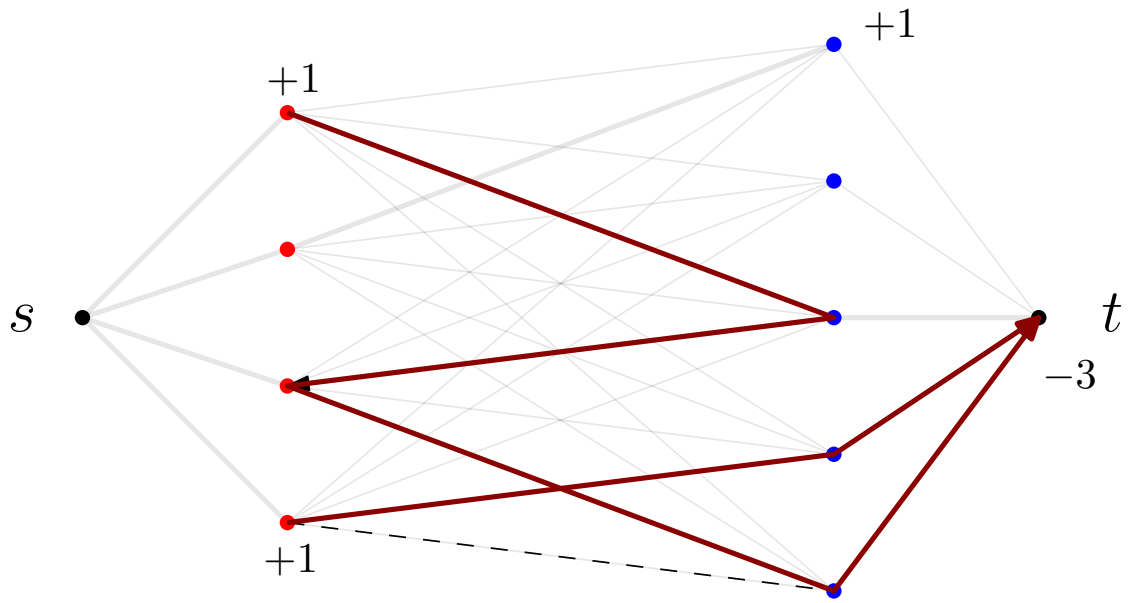
- Find by DFS through the admissible residual arcs.

Refinement by blocking flows

1. Hungarian search: Dijkstra-like search changing π until $\exists$ an admissible augmenting path.

   Dijkstra-like search changing π in units of θ until $\exists$ an admissible augmenting path.
2. Augment by the admissible augmenting path you found.

   Augment by an admissible blocking flow g.



- Find by DFS through the admissible residual arcs.

Refinement by blocking flows

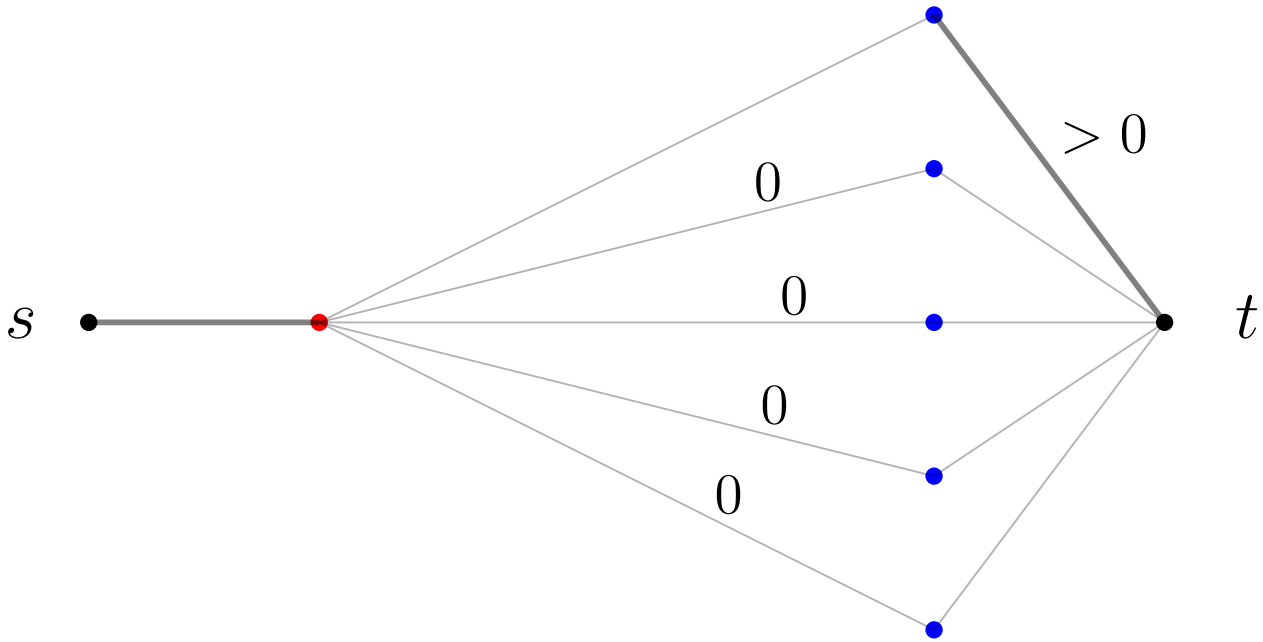
1. Hungarian search: Dijkstra-like search changing π in units of θ until $\exists$ an admissible augmenting path.
2. Augment by an admissible blocking flow g.

Lemma

After $O(\sqrt{k})$ blocking flows, f is a circulation.

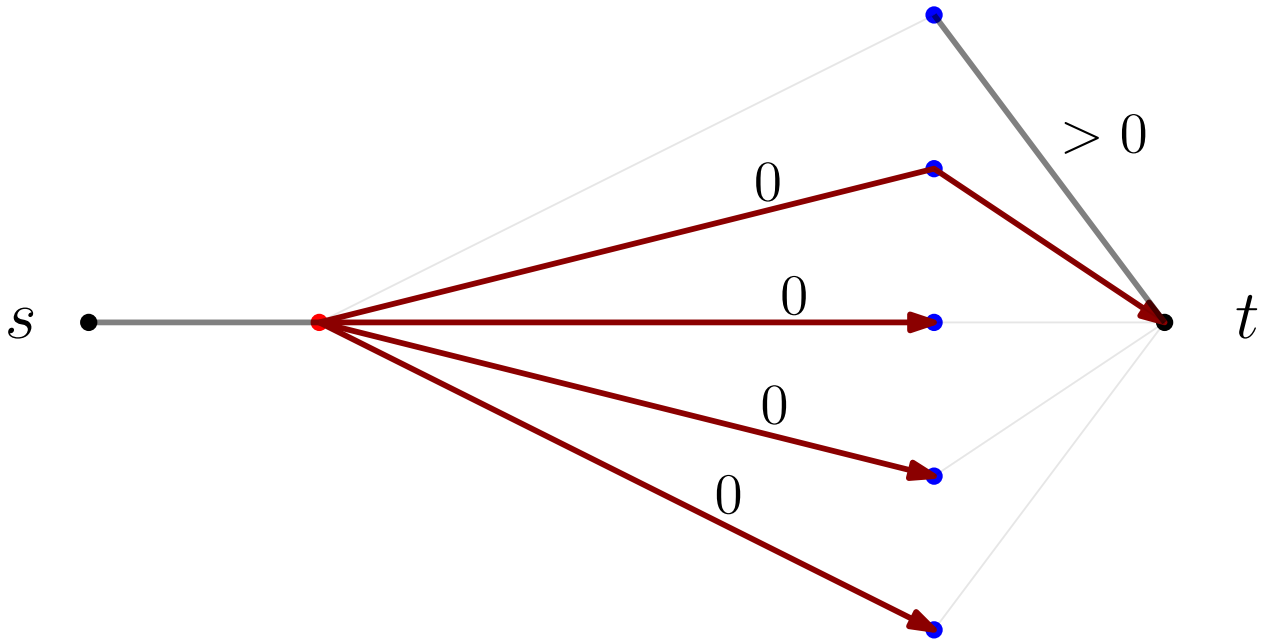
Hungarian search dead ends

- There can be $\Omega(n)$ relaxations even if k is very small.



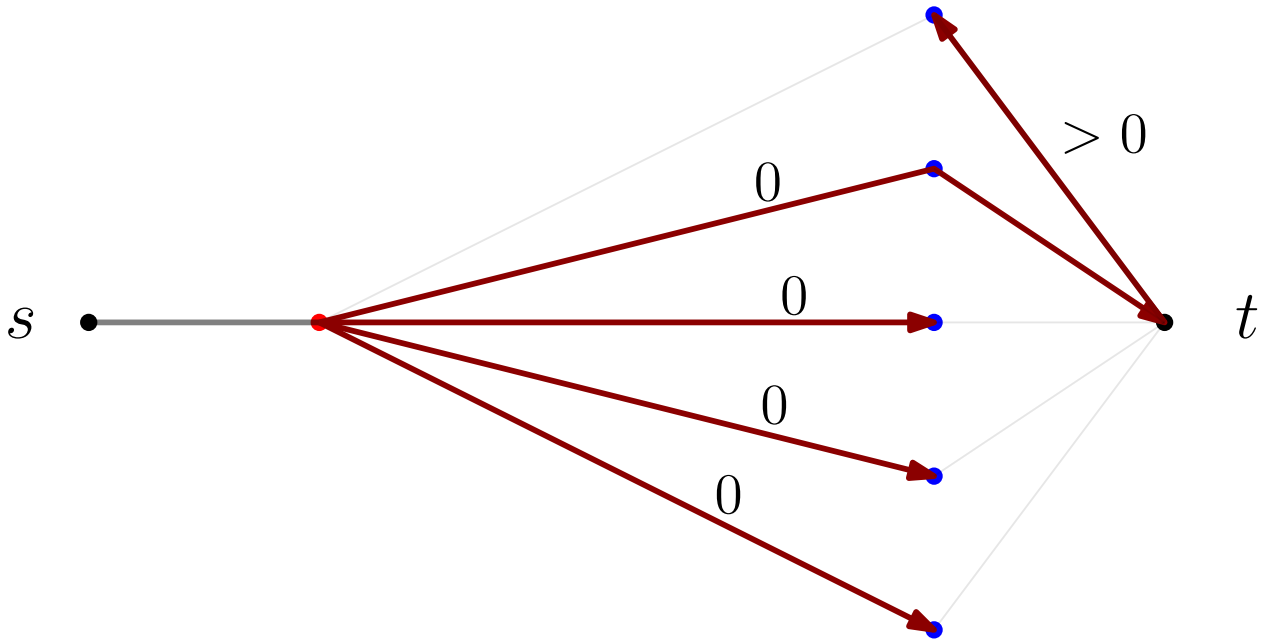
Hungarian search dead ends

- There can be $\Omega(n)$ relaxations even if k is very small.



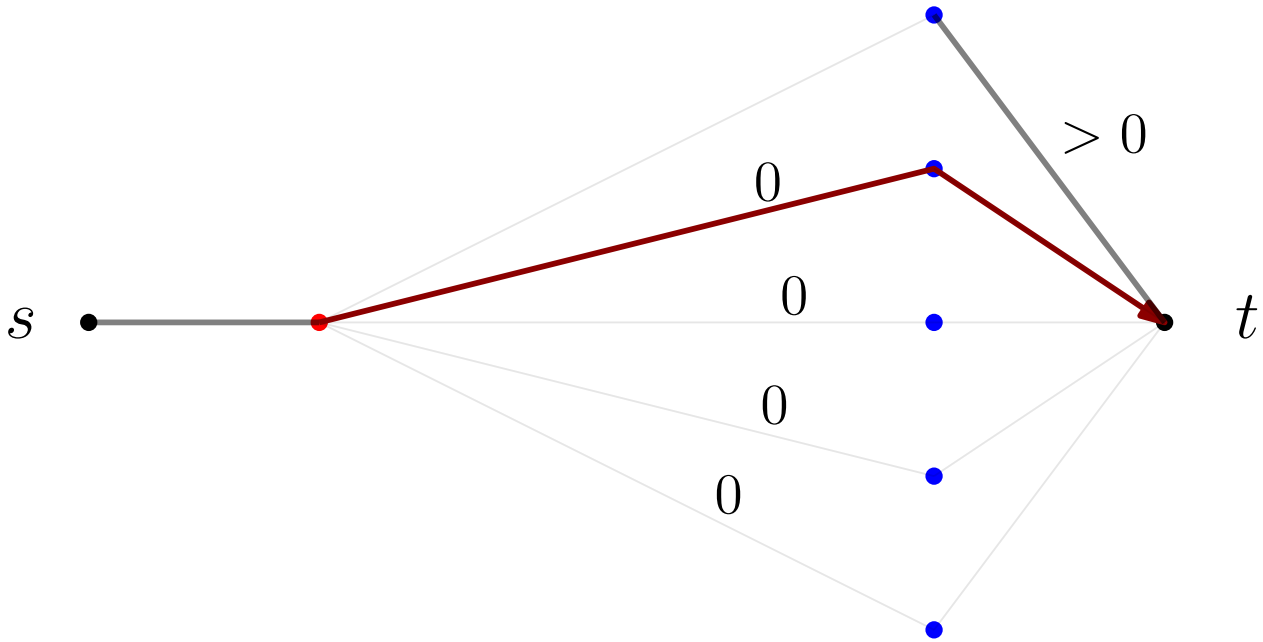
Hungarian search dead ends

- There can be $\Omega(n)$ relaxations even if k is very small.



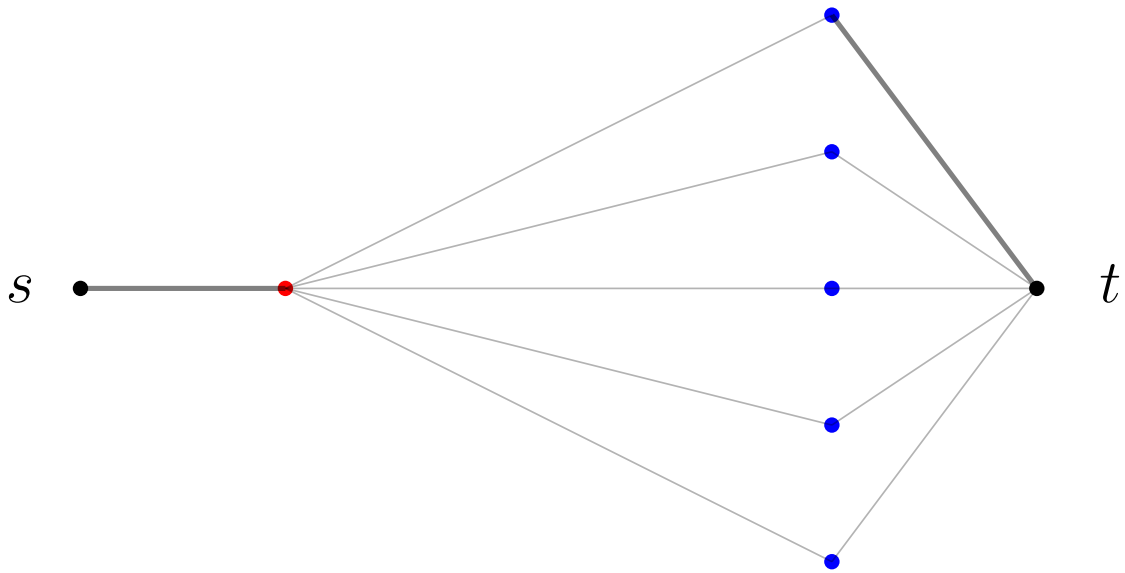
Hungarian search dead ends

- There can be $\Omega(n)$ relaxations even if k is very small.



Dead or alive

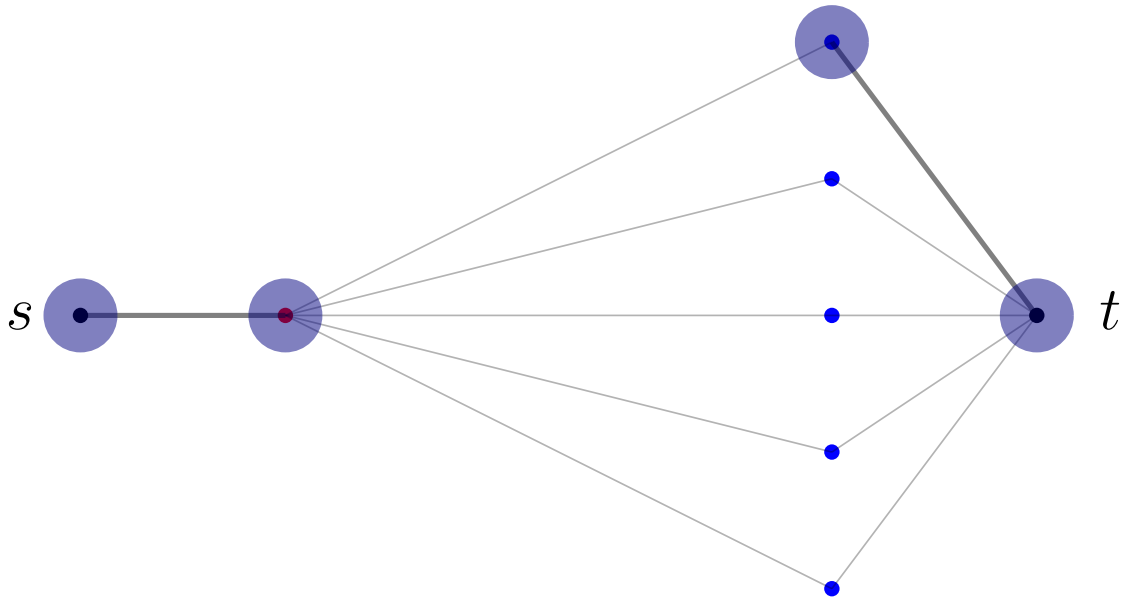
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- 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 or alive

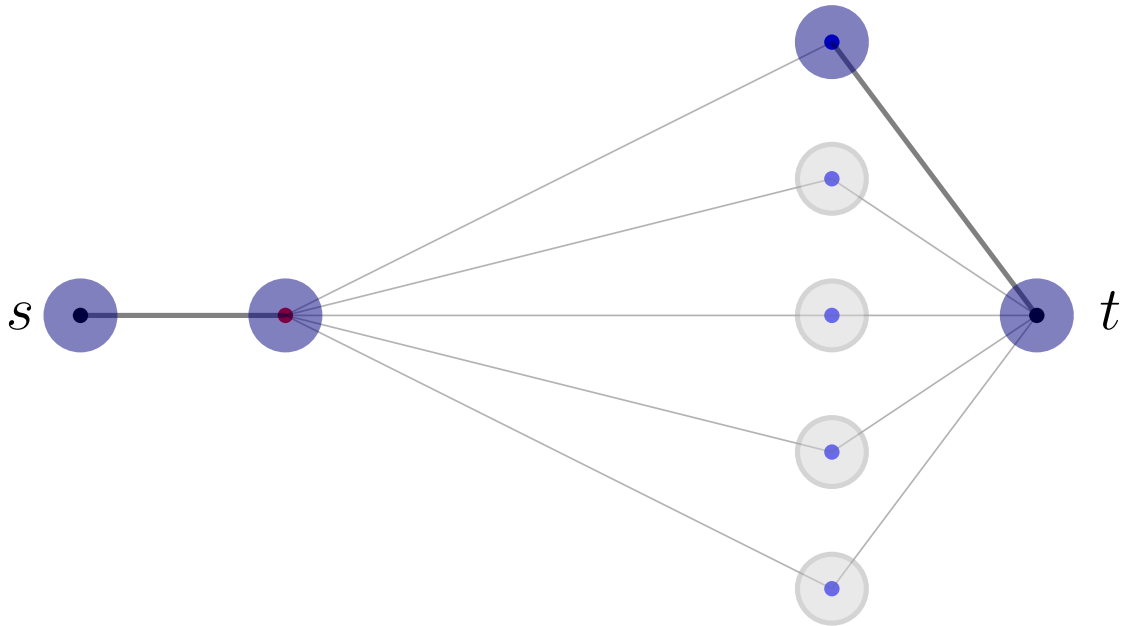
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- 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 or alive

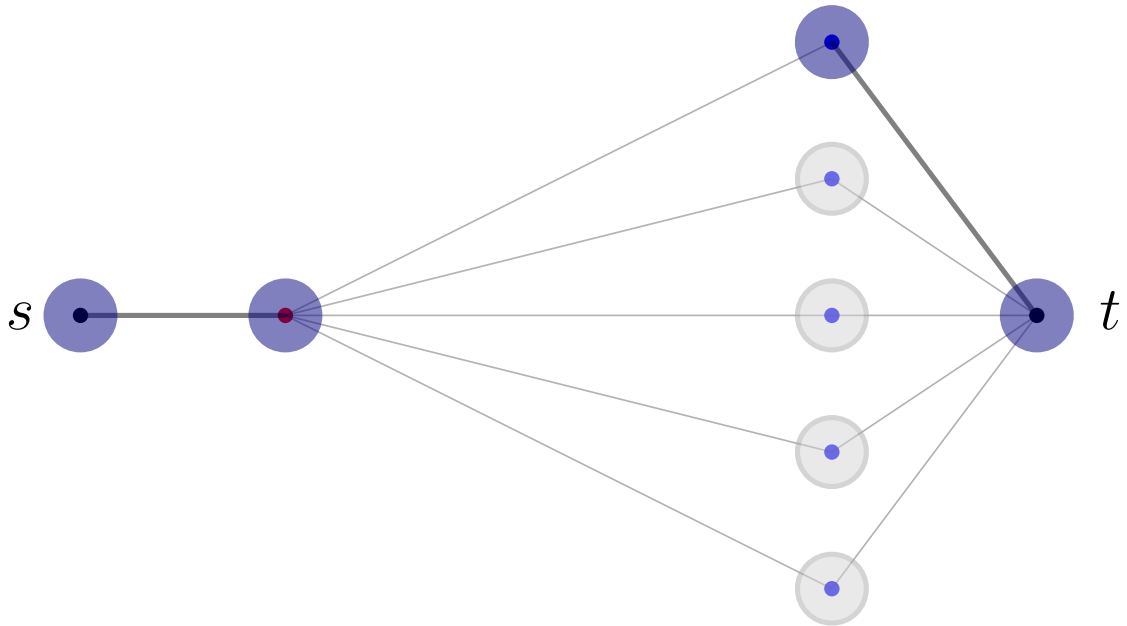
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- 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 or alive

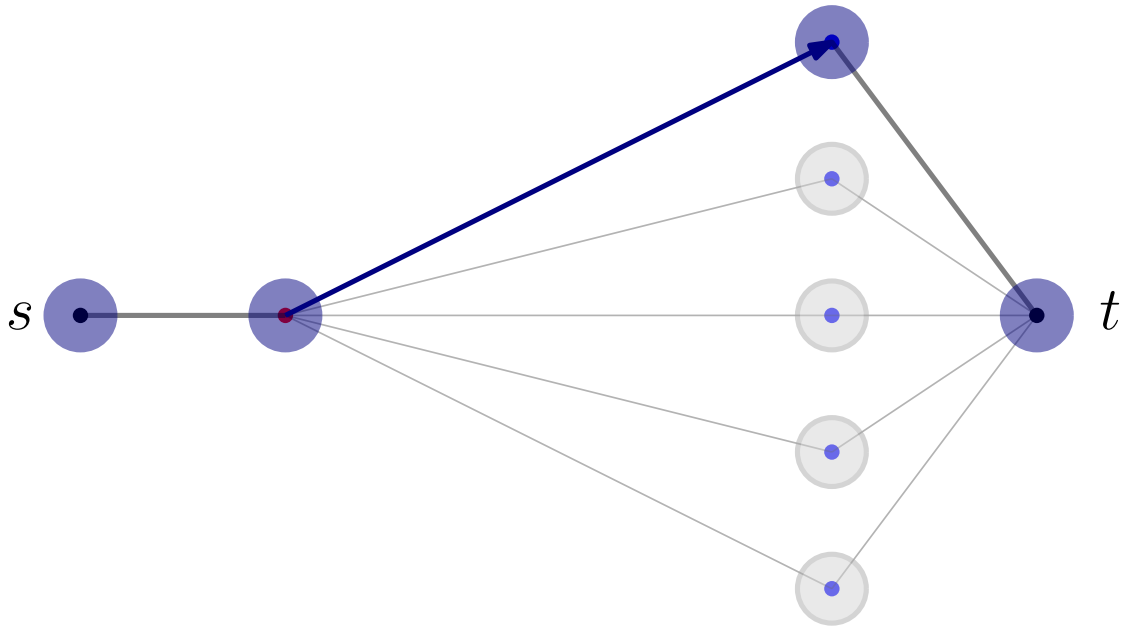
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- 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 or alive

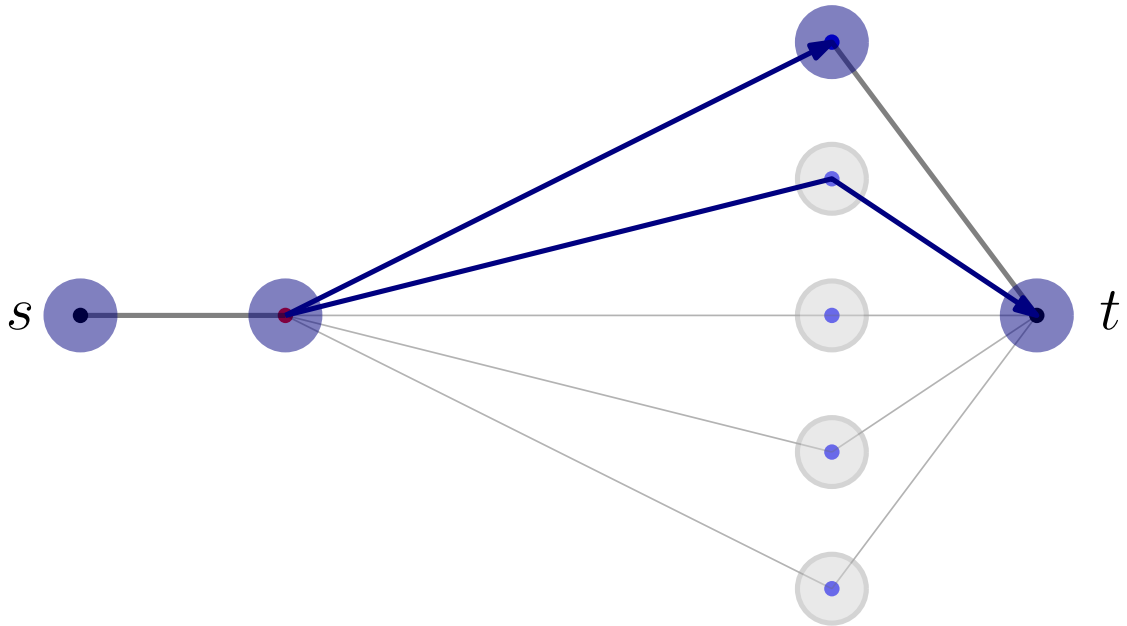
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- 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 or alive

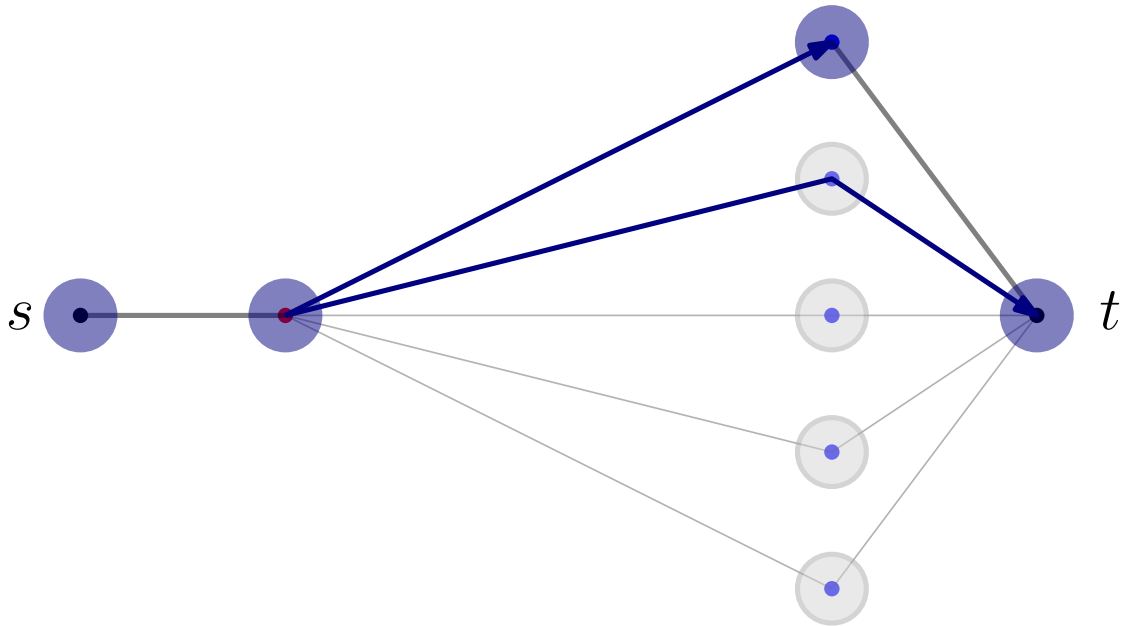
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



- 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 or alive

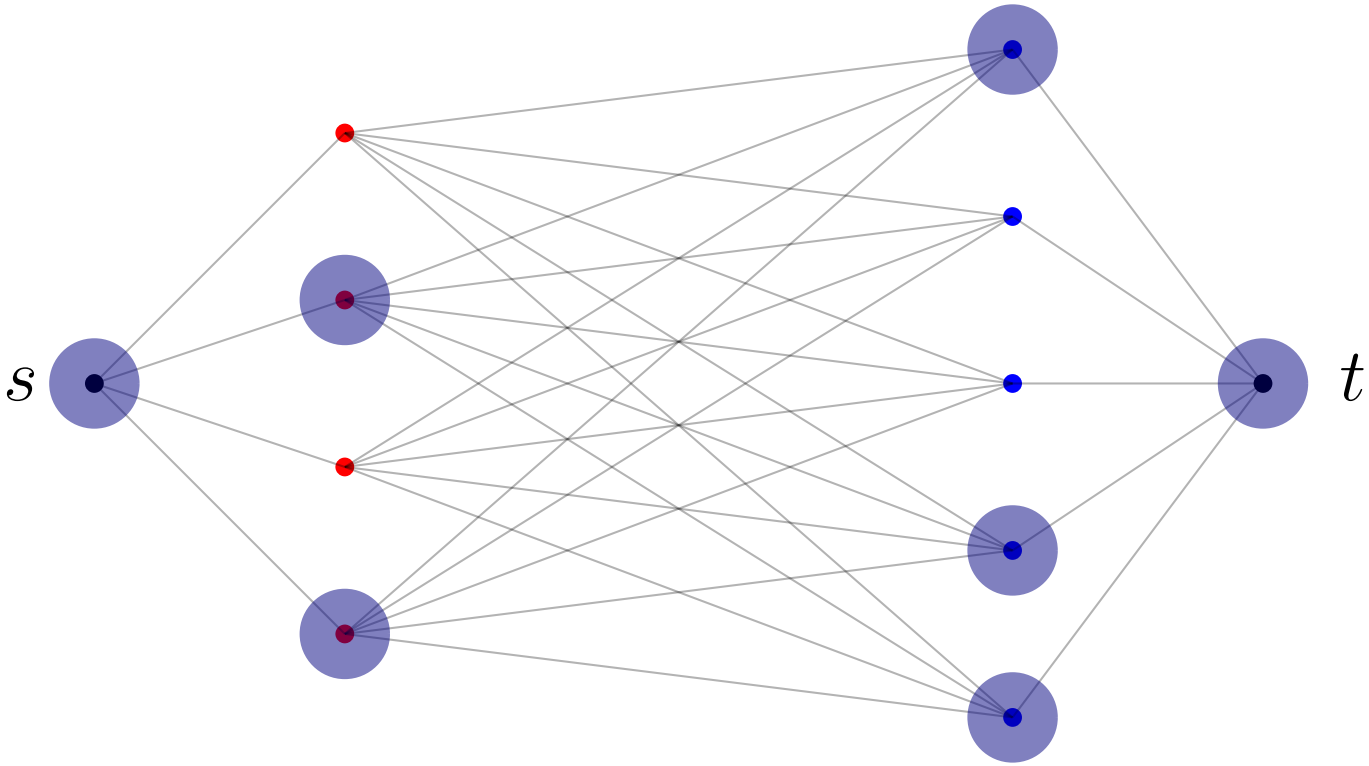
- Alive nodes: nonzero excess/deficit, or adjoining flow support arcs.
- Dead nodes: ones which aren't alive.



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

Hungarian search over alive paths

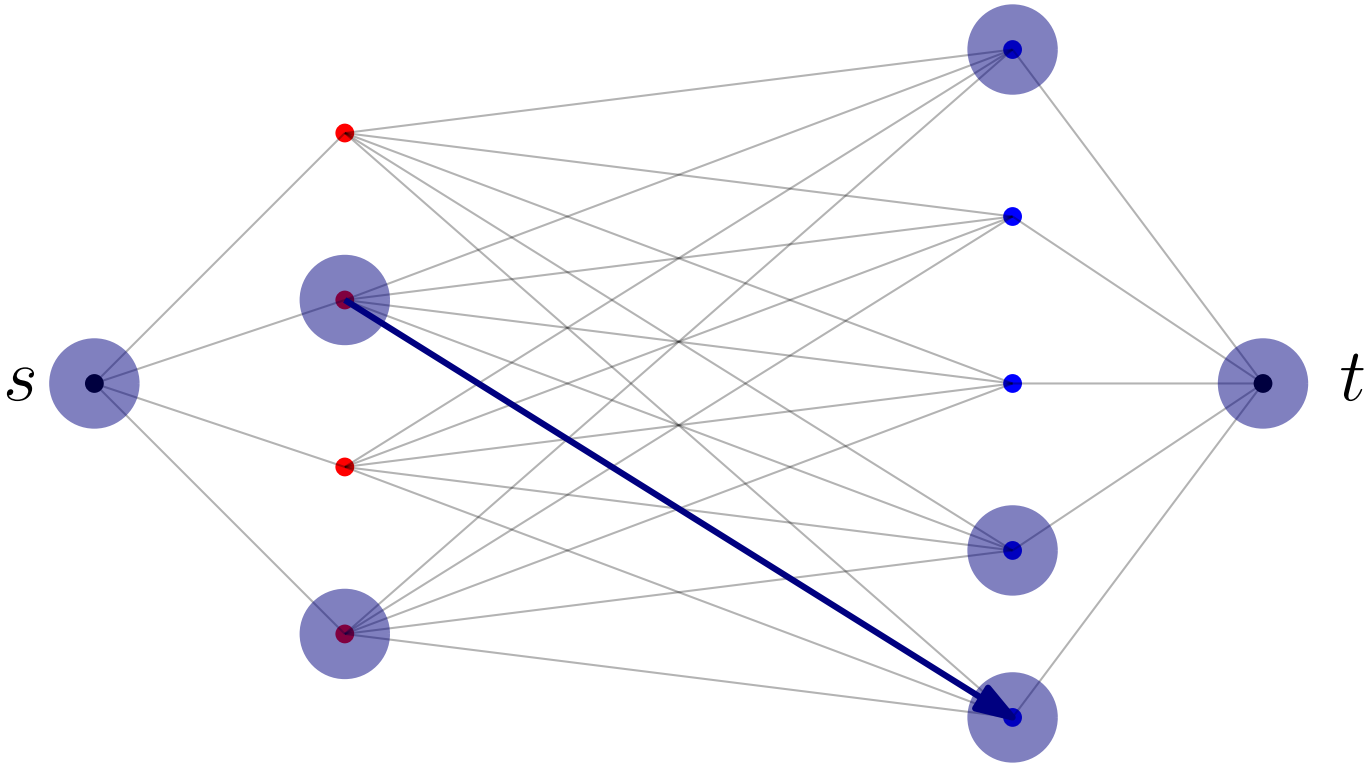
- Alive paths have length 1, 2, or 3.



- Telescoping on alive paths: $c_\pi(s \to a \to b) = c(a,b) - \pi(s) + \pi(b)$
- Can still use BCP.

Hungarian search over alive paths

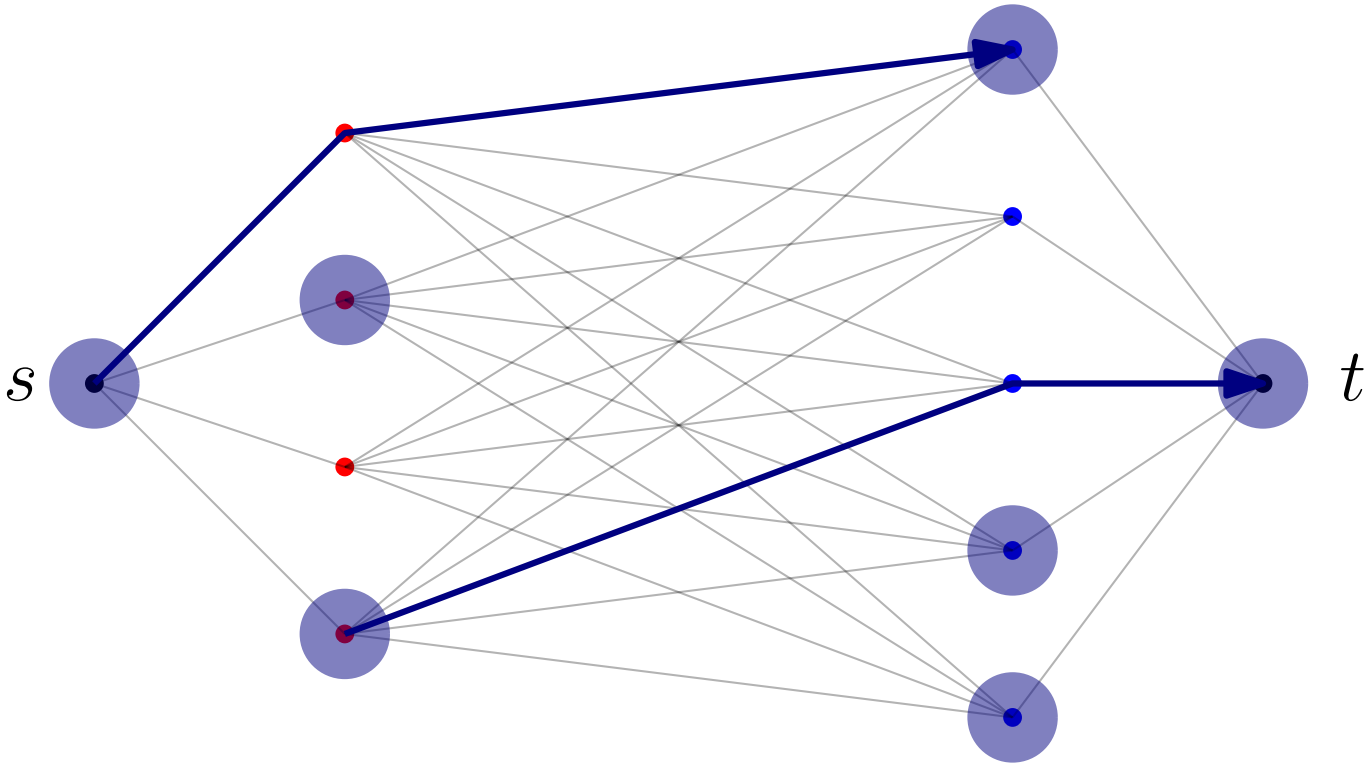
- Alive paths have length 1, 2, or 3.



- Telescoping on alive paths: $c_\pi(s \to a \to b) = c(a,b) - \pi(s) + \pi(b)$
- Can still use BCP.

Hungarian search over alive paths

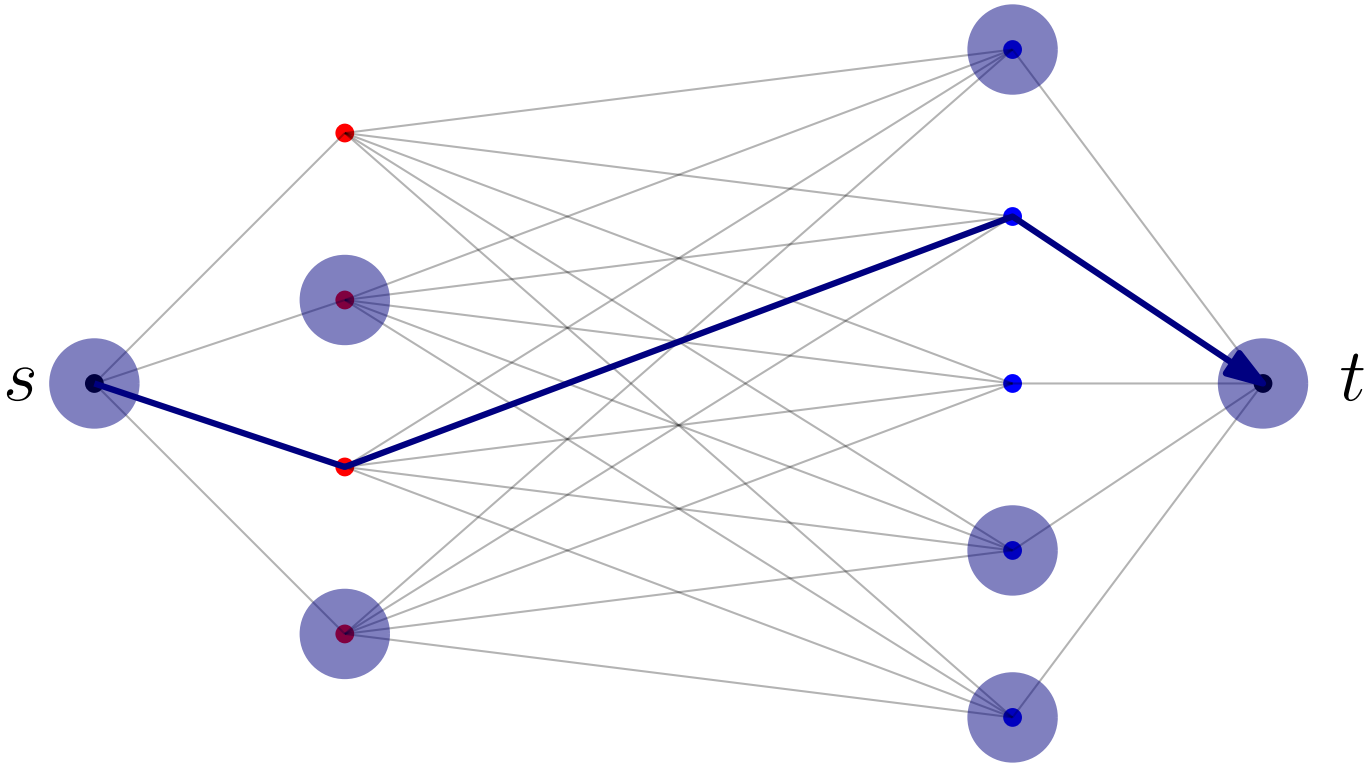
- Alive paths have length 1, 2, or 3.



- Telescoping on alive paths: $c_\pi(s \to a \to b) = c(a,b) - \pi(s) + \pi(b)$
- Can still use BCP.

Hungarian search over alive paths

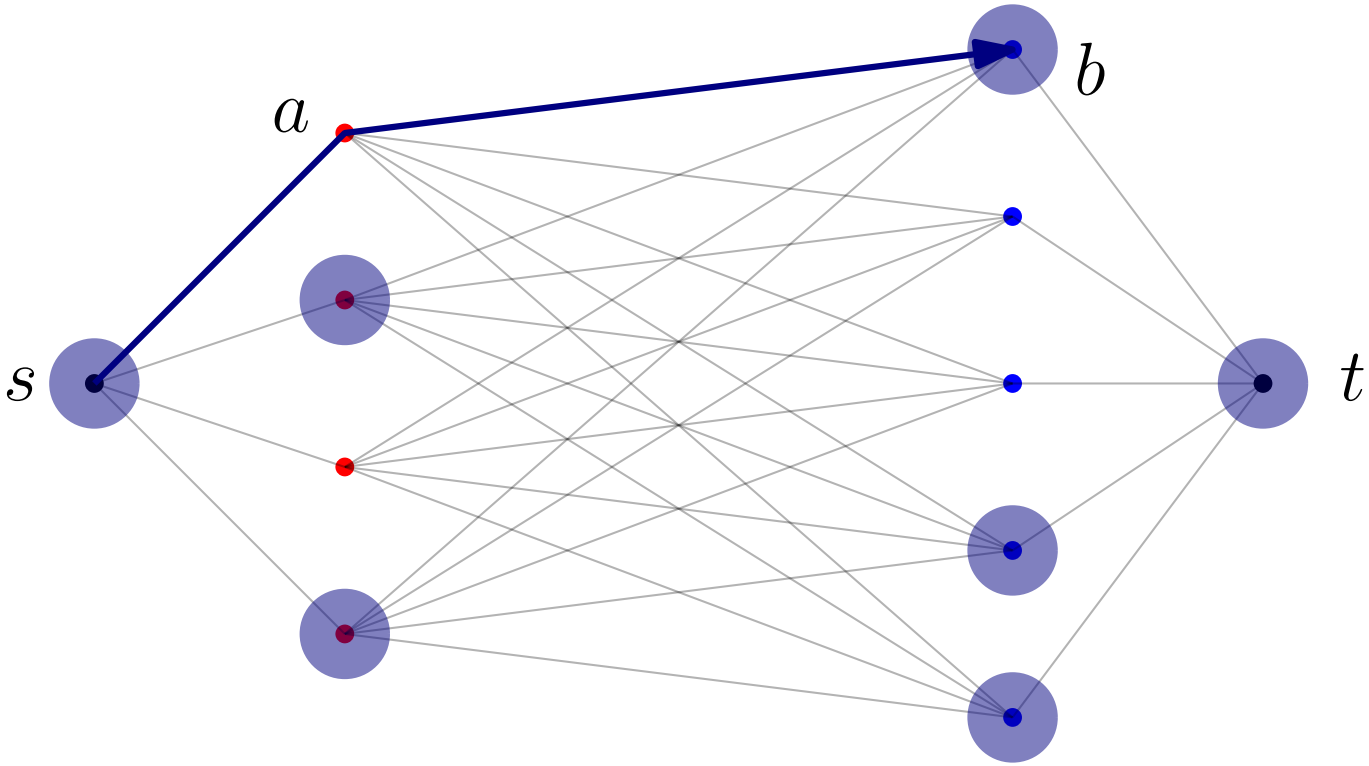
- Alive paths have length 1, 2, or 3.



- Telescoping on alive paths: $c_\pi(s \to a \to b) = c(a,b) - \pi(s) + \pi(b)$
- Can still use BCP.

Hungarian search over alive paths

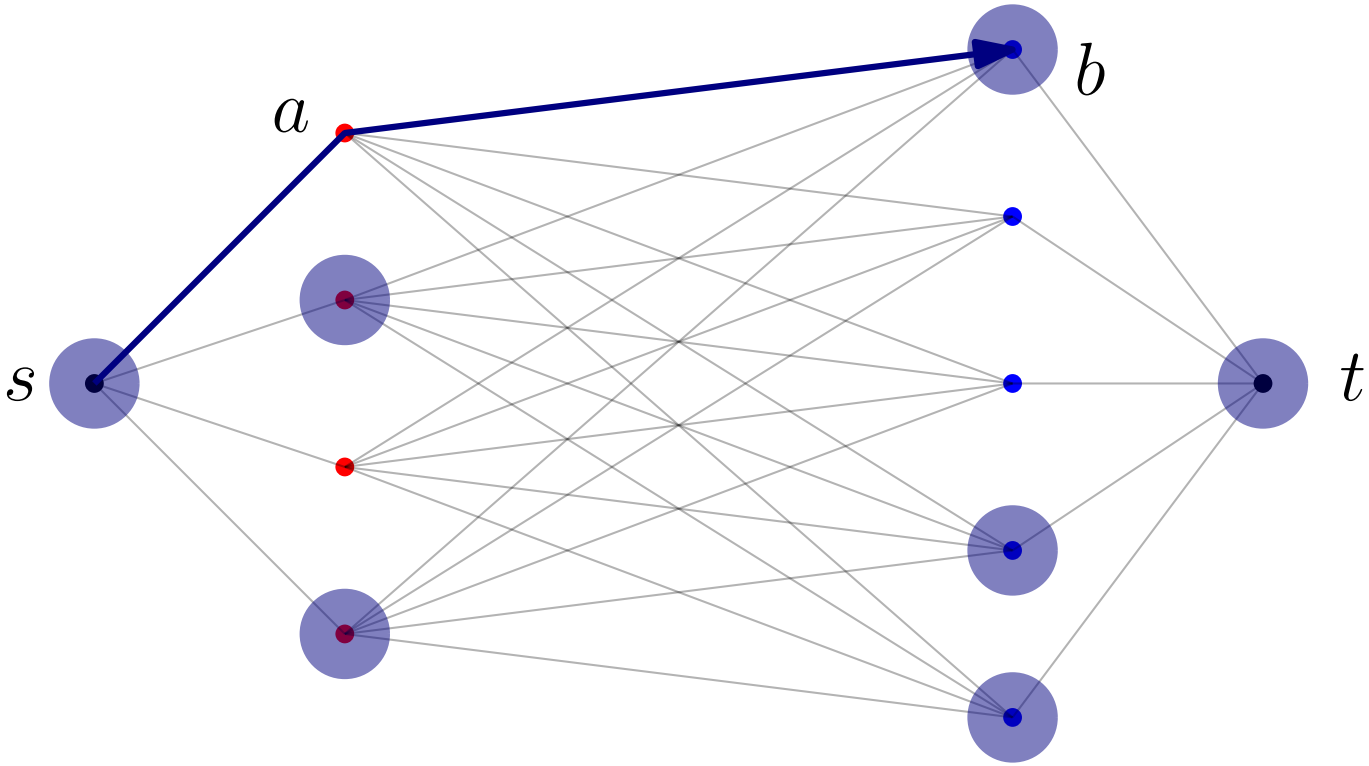
- Alive paths have length 1, 2, or 3.



- Telescoping on alive paths: $c_\pi(s \to a \to b) = c(a,b) - \pi(s) + \pi(b)$
- Can still use BCP.

Hungarian search over alive paths

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



- Telescoping on alive paths: $c_\pi(s \to a \to b) = c(a,b) - \pi(s) + \pi(b)$
- Can still use BCP.

How many relaxations now?

- $O(k)$ alive nodes.
- Every alive path relaxation adds another alive node to X.
- Thus, $O(k)$ relaxations.

Dead node potentials

- Dead nodes may become alive again after an augmentation.
- Also, at the end of a scale, we need potentials for all nodes.
- What's the right value for a dead node's potential?

$$\pi(v) = \begin{cases} \pi(s) & v \in A \\ \pi(t) & v \in B \end{cases}$$

- This choice preserves θ-optimality and ensures that the component arcs of an admissible alive path are admissible.

Dead node potentials

- Dead nodes may become alive again after an augmentation.
- Also, at the end of a scale, we need potentials for all nodes.
- What's the right value for a dead node's potential?

$$\pi(v) = \begin{cases} \pi(s) & v \in A \\ \pi(t) & v \in B \end{cases}$$

- This choice preserves θ-optimality and ensures that the component arcs of an admissible alive path are admissible.

Dead node potentials

- Dead nodes may become alive again after an augmentation.
- Also, at the end of a scale, we need potentials for all nodes.
- What's the right value for a dead node's potential?

$$\pi(v) = \begin{cases} \pi(s) & v \in A \\ \pi(t) & v \in B \end{cases}$$

- This choice preserves θ-optimality and ensures that the component arcs of an admissible alive path are admissible.

Cost-scaling time summary

- $O(\log(n^q/\varepsilon))$ scales.
- $O(\sqrt{k})$ iterations of refinement.
- Each blocking flow has size $O(k)$.

- Combine alive paths with previous ideas: rewinding and batched potential updates.
- $O(k \operatorname{polylog} n)$ time for each Hungarian search and DFS (finding blocking flow).
- $O(n \operatorname{polylog} n)$ to build the initial BCP data structures per scale.

- $O((n + k\sqrt{k}) \operatorname{polylog} n)$ time per scale, $O(\log(n^q/\varepsilon))$ scales, $(1+\varepsilon)$-approx.

Cost-scaling time summary

- $O(\log(n^q/\varepsilon))$ scales.
- $O(\sqrt{k})$ iterations of refinement.
- Each blocking flow has size $O(k)$.

- Combine alive paths with previous ideas: rewinding and batched potential updates.
- $O(k \operatorname{polylog} n)$ time for each Hungarian search and DFS (finding blocking flow).
- $O(n \operatorname{polylog} n)$ to build the initial BCP data structures per scale.

- $O((n + k\sqrt{k}) \operatorname{polylog} n)$ time per scale, $O(\log(n^q/\varepsilon))$ scales, $(1+\varepsilon)$-approx.

Cost-scaling time summary

- $O(\log(n^q/\varepsilon))$ scales.
- $O(\sqrt{k})$ iterations of refinement.
- Each blocking flow has size $O(k)$.

- Combine alive paths with previous ideas: rewinding and batched potential updates.
- $O(k \operatorname{polylog} n)$ time for each Hungarian search and DFS (finding blocking flow).
- $O(n \operatorname{polylog} n)$ to build the initial BCP data structures per scale.

- $O((n + k\sqrt{k}) \operatorname{polylog} n)$ time per scale, $O(\log(n^q/\varepsilon))$ scales, $(1+\varepsilon)$-approx.

Cost-scaling time summary

- $O(\log(n^q/\varepsilon))$ scales.
- $O(\sqrt{k})$ iterations of refinement.
- Each blocking flow has size $O(k)$.

- Combine alive paths with previous ideas: rewinding and batched potential updates.
- $O(k \operatorname{polylog} n)$ time for each Hungarian search and DFS (finding blocking flow).
- $O(n \operatorname{polylog} n)$ to build the initial BCP data structures per scale.

- $O((n + k\sqrt{k}) \operatorname{polylog} n)$ time per scale, $O(\log(n^q/\varepsilon))$ scales, $(1+\varepsilon)$-approx.

Cost-scaling time summary

- $O(\log(n^q/\varepsilon))$ scales.
- $O(\sqrt{k})$ iterations of refinement.
- Each blocking flow has size $O(k)$.

- Combine alive paths with previous ideas: rewinding and batched potential updates.
- $O(k \operatorname{polylog} n)$ time for each Hungarian search and DFS (finding blocking flow).
- $O(n \operatorname{polylog} n)$ to build the initial BCP data structures per scale.

- $O((n + k\sqrt{k}) \operatorname{polylog} n)$ time per scale, $O(\log(n^q/\varepsilon))$ scales, $(1+\varepsilon)$-approx.

The End

Thank you.