Online bipartite matching in offline time

Bartłomiej Bosek[1] **Dariusz Leniowski**[2]
Piotr Sankowski[2] Anna Zych[2]

[1]Jagiellonian University

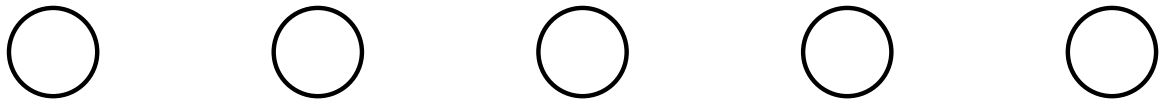
[2]University of Warsaw

FIT 2015, 31 January 2015

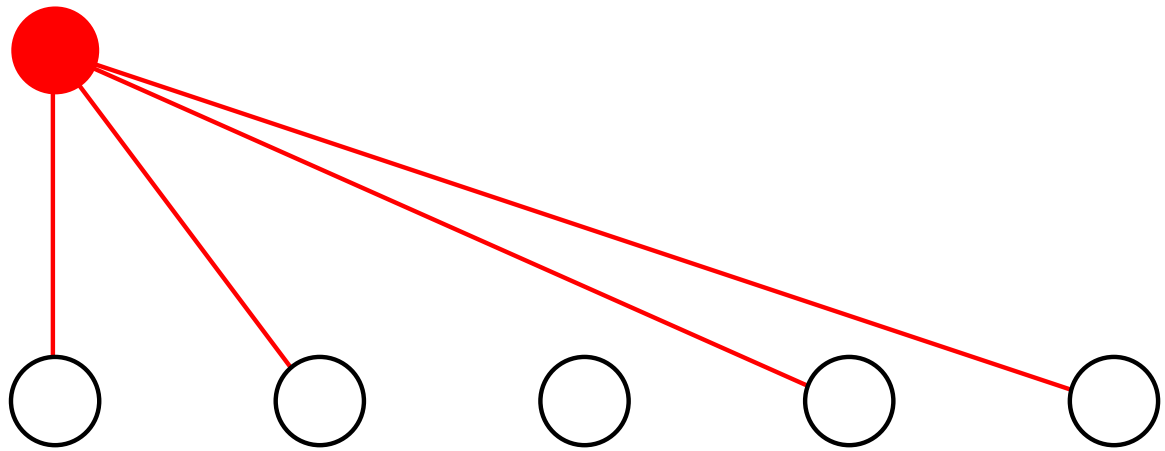
Problem statement

- A bipartite graph $G = \langle U \uplus V, E \rangle$ is revealed online.
- The vertices in U are fixed.
- The vertices from V arrive one per turn, each with all its incident edges.
- We want to *maintain* the maximum cardinality matching, and if possible, keep the number of reallocations low.

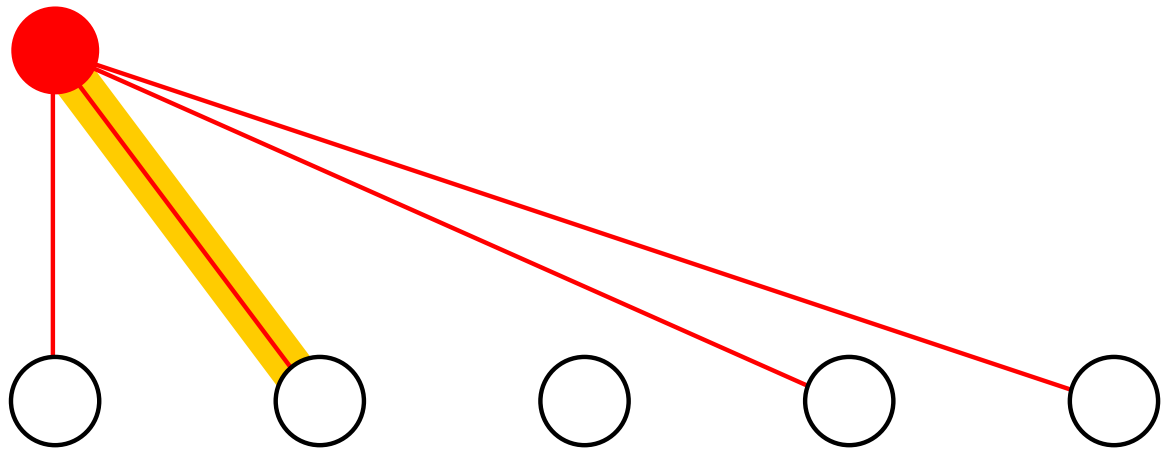
Problem statement — example



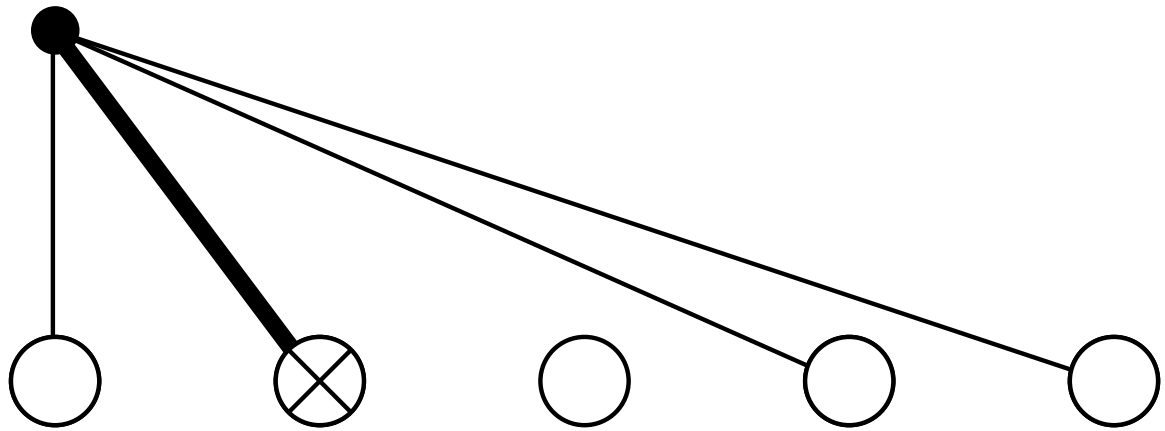
Problem statement — example



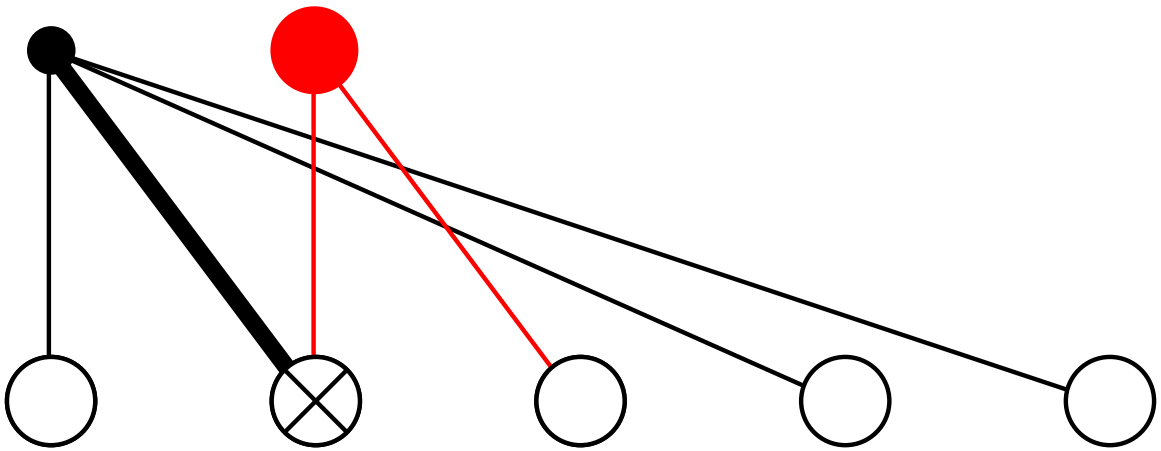
Problem statement — example



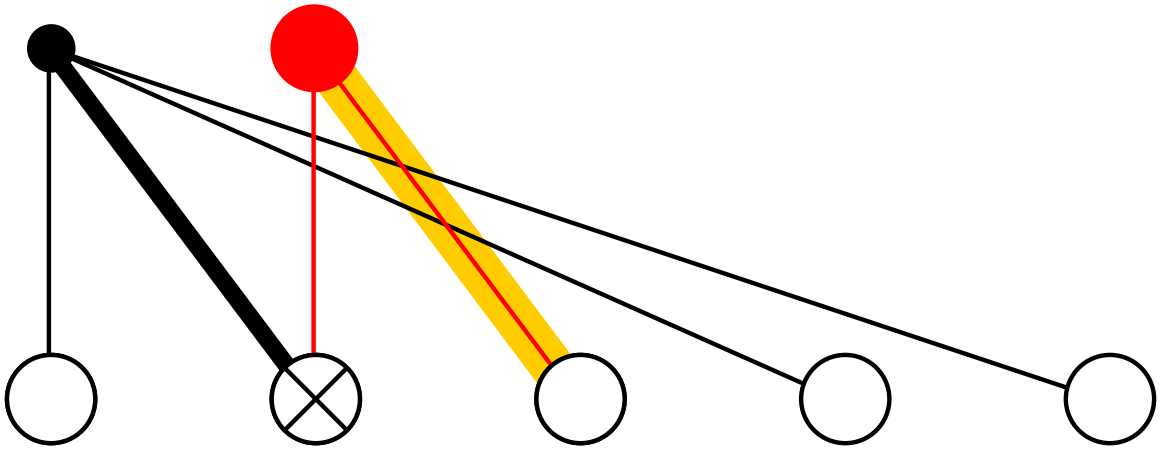
Problem statement — example



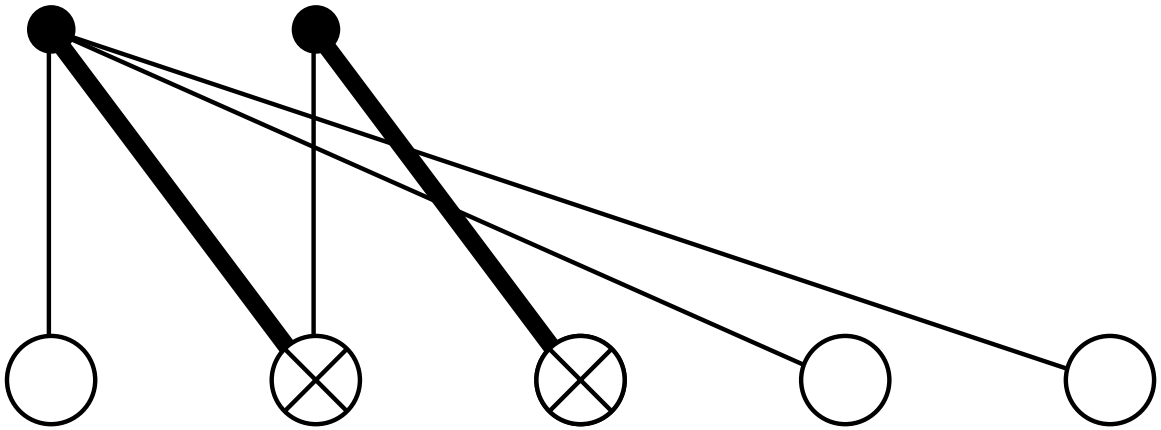
Problem statement — example



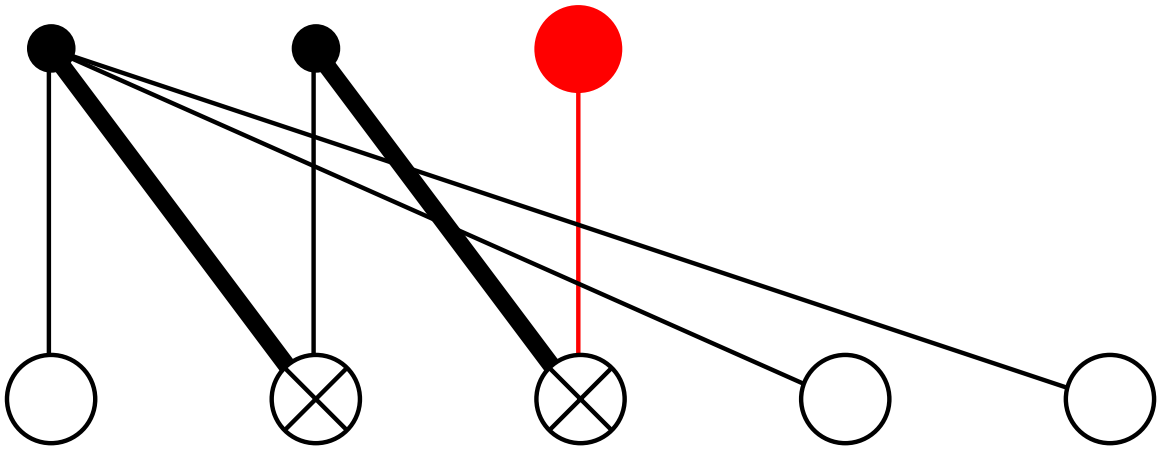
Problem statement — example



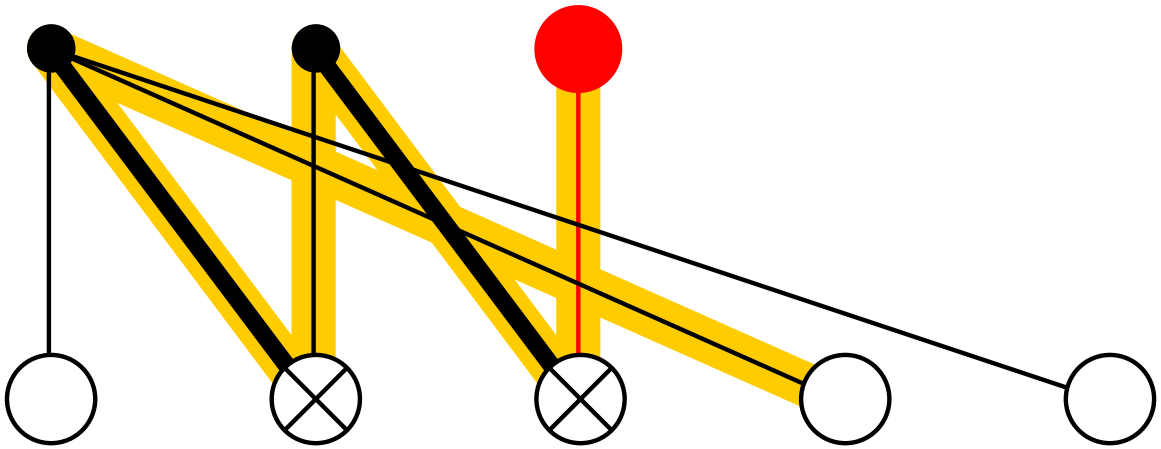
Problem statement — example



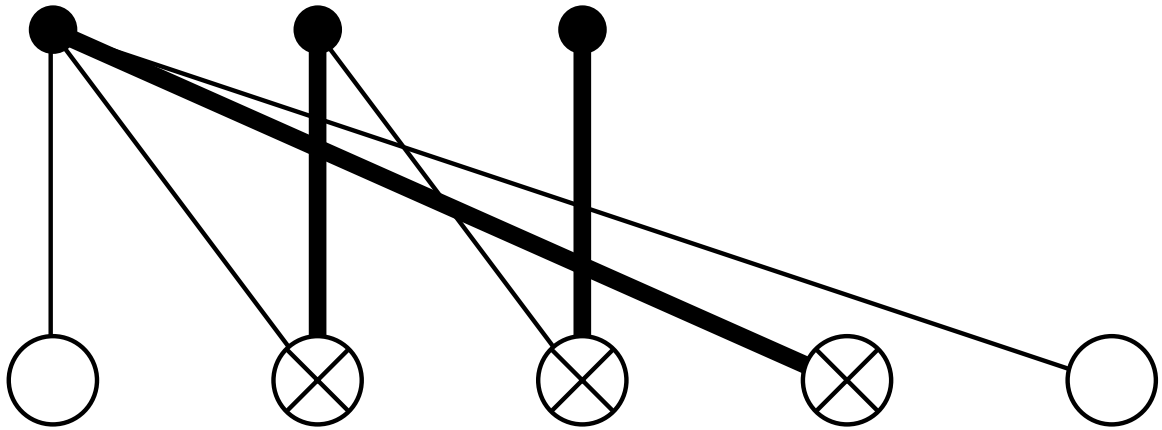
Problem statement — example



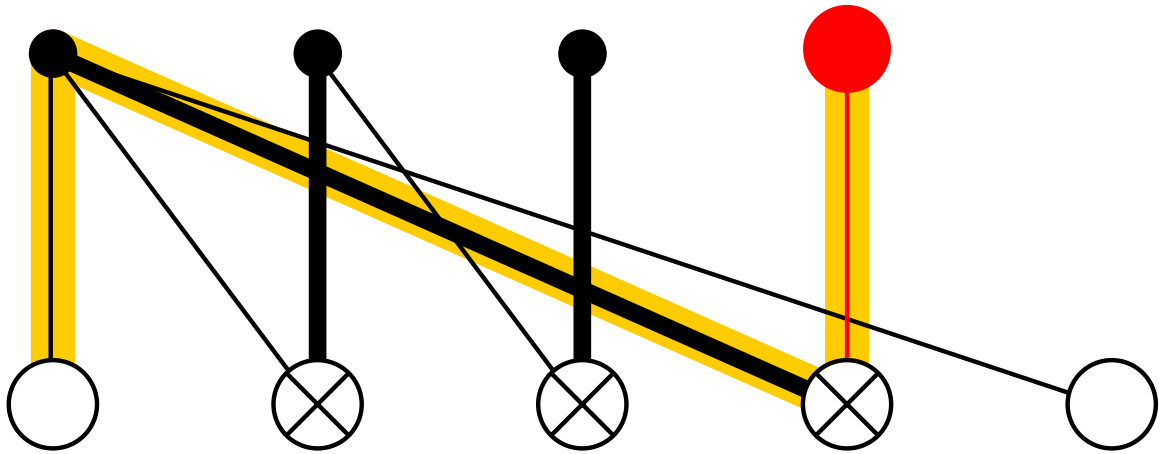
Problem statement — example



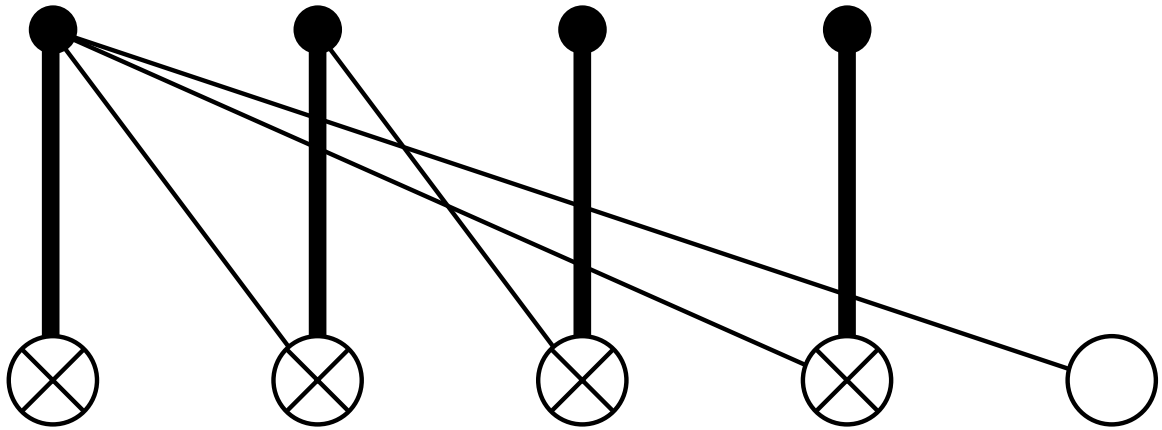
Problem statement — example



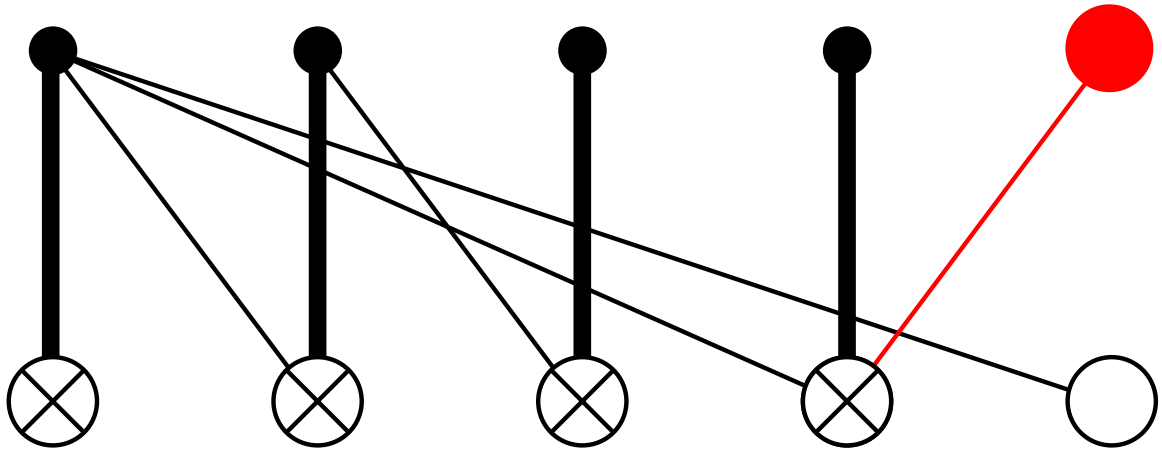
Problem statement — example



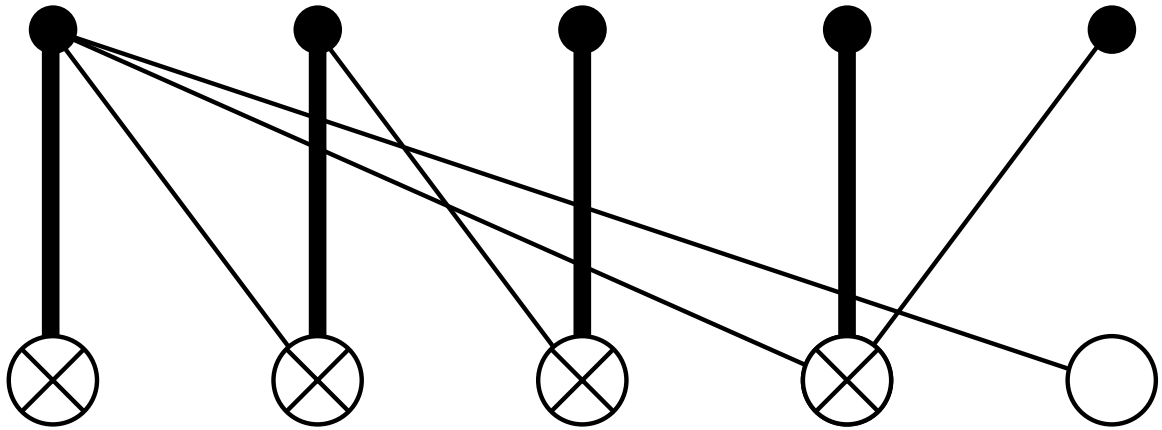
Problem statement — example



Problem statement — example



Problem statement — example



Our results

- Bound: it is possible to maintain maximum matching using $2\sqrt{n}$ reallocations per vertex.
- Exact: in $\mathcal{O}(m\sqrt{n})$ total time maintains the maximum cardinality matching.
- Approximation: in $\mathcal{O}(\varepsilon^{-1}m)$ total time maintains $(1-\varepsilon)$-approximation factor.
- Reductions: vertex-decremental (same bounds), weighted (pseudopolynomial bound).

Motivation

- Servers and new jobs to process, edges denote eligibility.
- We want to admit as many requests as possible — *we reallocate them when necessary*.
- Applications include streaming content delivery, web hosting, remote data storage, job scheduling, hashing, etc.

History

- Grove, Kao, Krishnan and Vitter, '95:
  at most 2 servers per client using $\mathcal{O}(\log n)$
  reallocations per vertex, $\mathcal{O}(n \log n)$ total.
- Chaudhuri, Daskalakis, Kleinberg and Lin '09:
  forests or particular random graphs,
  total bound of $\mathcal{O}(n \log n)$.
- General case:
  only the trivial $\mathcal{O}(n^2)$ bound was known.

Relation to other models

- Karp, Vazirani and Vazirani, '90: no reallocations, $\mathcal{O}(m)$ total time, approximation factor of $(1 - 1/e) \approx 0.63$.
- Gupta, Peng '13: $(1 - \varepsilon)$-approximation in $\mathcal{O}(m\sqrt{m})$ total time.
- Our result: $(1 - \varepsilon)$-approximation in $\mathcal{O}(\varepsilon^{-1}m)$ total time, exact in $\mathcal{O}(m\sqrt{n})$ time, matches the running time of the offline Hopcroft-Karp algorithm.

Main idea

- For each $u \in U$ we keep track of its *rank*, i.e., the number of times it was used by augmenting paths up to this point.
- Suppose that vertex of V arrives.
- Pick an augmenting path that minimizes the maximum rank for its every suffix.

Ranks and tiers

Rank

Given a vertex $u \in U$

$$\begin{aligned} \mathrm{rank}_t(u) &= \text{how many times augmenting paths used } u, \\ \mathrm{rank}_t(P) &= \max_{u \in P} \mathrm{rank}_t(u). \end{aligned}$$

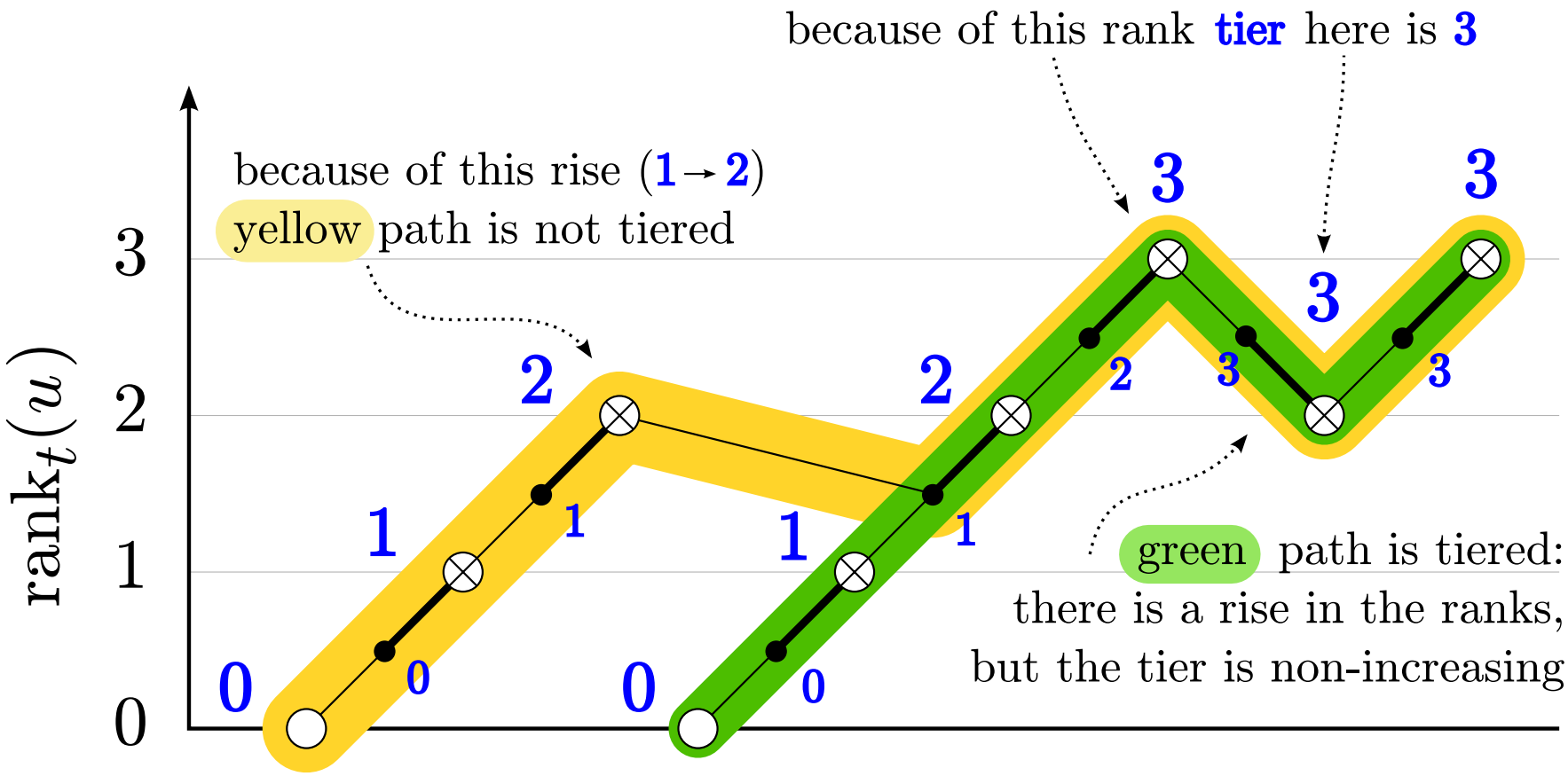
Tier

Given a vertex $w \in V \uplus U$,

$$\begin{aligned} \mathrm{tier}_t(w) &= \text{the rank of the lowest-ranked alternating} \\ &\quad \text{path to any unmatched vertex } u \in U \\ &= \min_{\text{unmatched } u \in U} \; \min_{\text{alternating path } P \,:\, w \to u} \mathrm{rank}_t(P). \end{aligned}$$

Tiered paths

A path is called *tiered* if the tiers are non-increasing.



Tiers — a few notes

- Tiers do not decrease in time.
- $\text{tier}(w) \leq \text{dist}_{\text{ALT}}(w, \text{unmatched } u)$.
- Tiers are hard to maintain.
- Instead, for any w we keep $\text{tierLB}(w)$,
  a number such that $\text{tierLB}(w) \leq \text{tier}_t(w)$;
  set $\text{tierLB}(w) = 0$ and update as necessary.
- **Theorem:** both tiers and ranks are $\mathcal{O}(\sqrt{n})$.

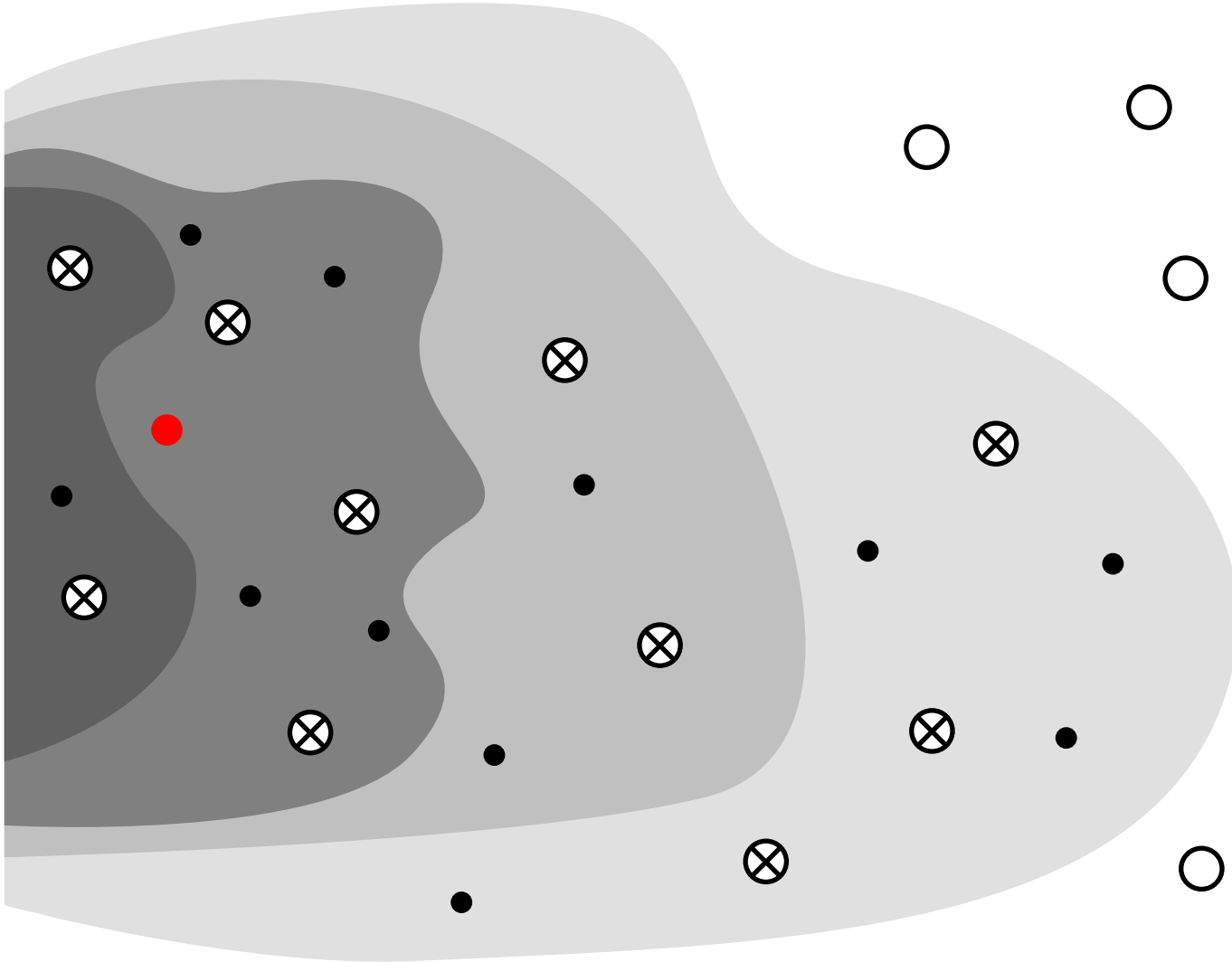
Algorithm — a sketch

1. Add a vertex v_t and assume its tierLB is 0.
2. Try to find a tiered augmenting path of rank at most tierLB(v_t).

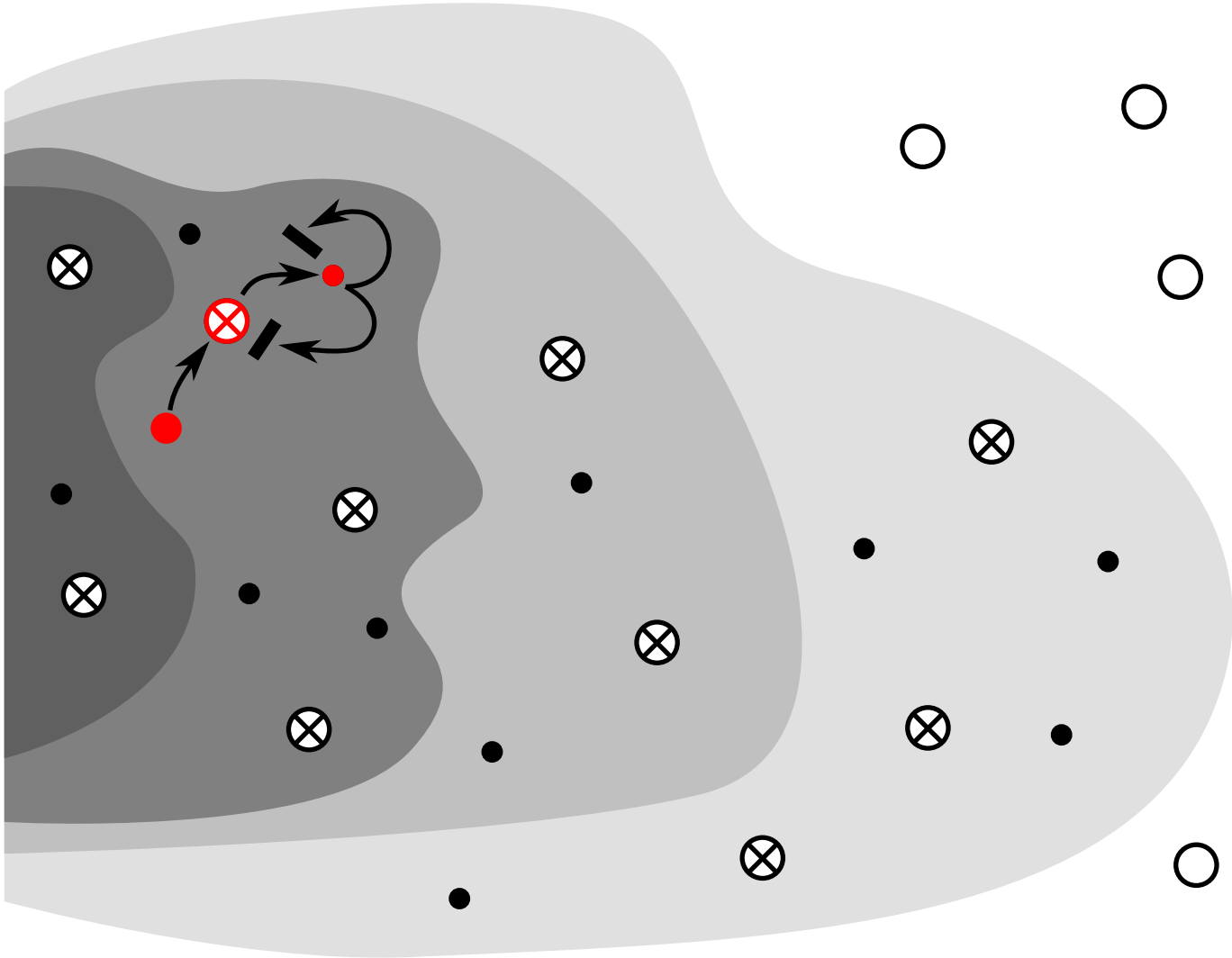
3a. Failed search means that some tierLB was too low, so raise it and repeat.

3b. For a successful search raise the ranks.

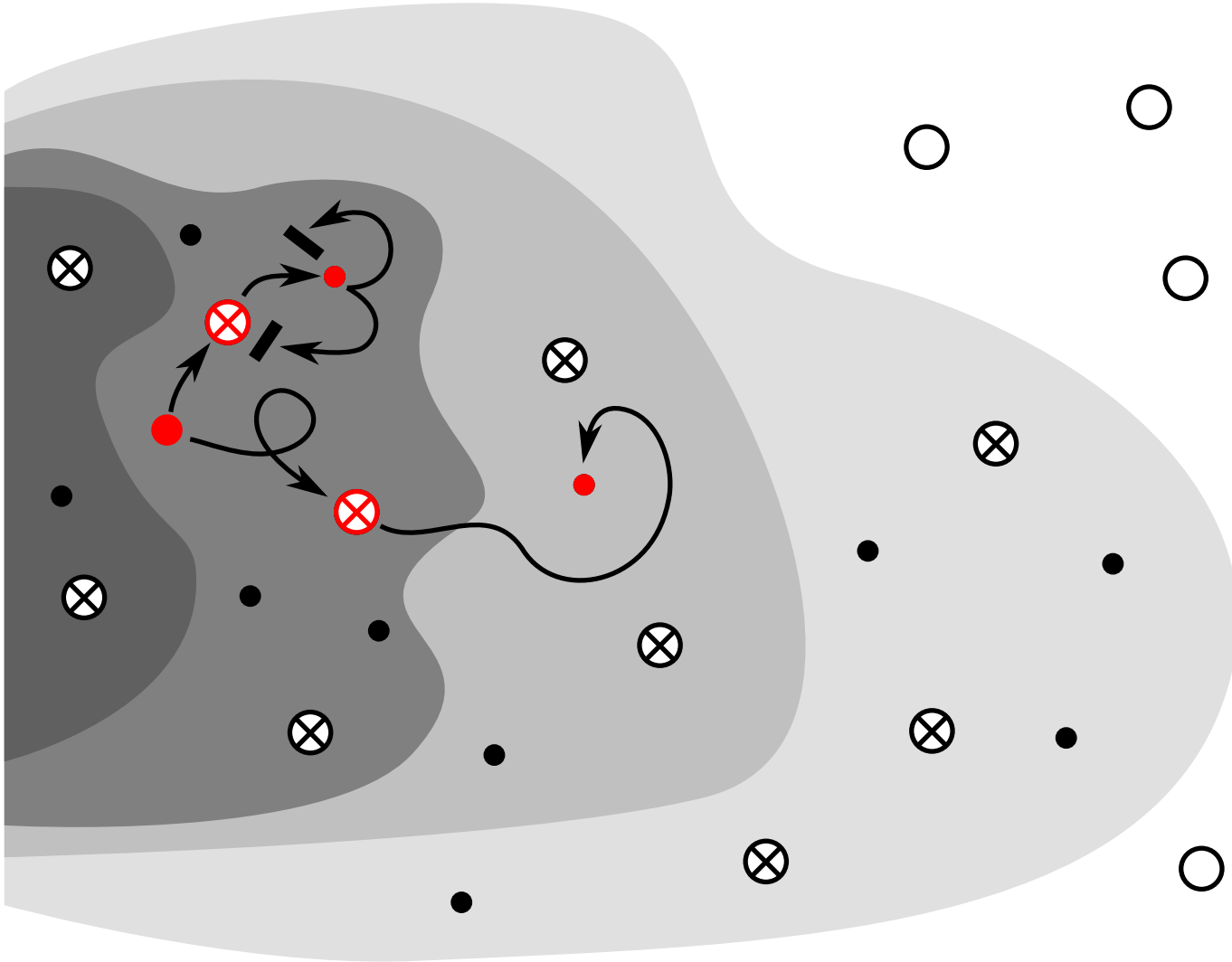
Algorithm — example



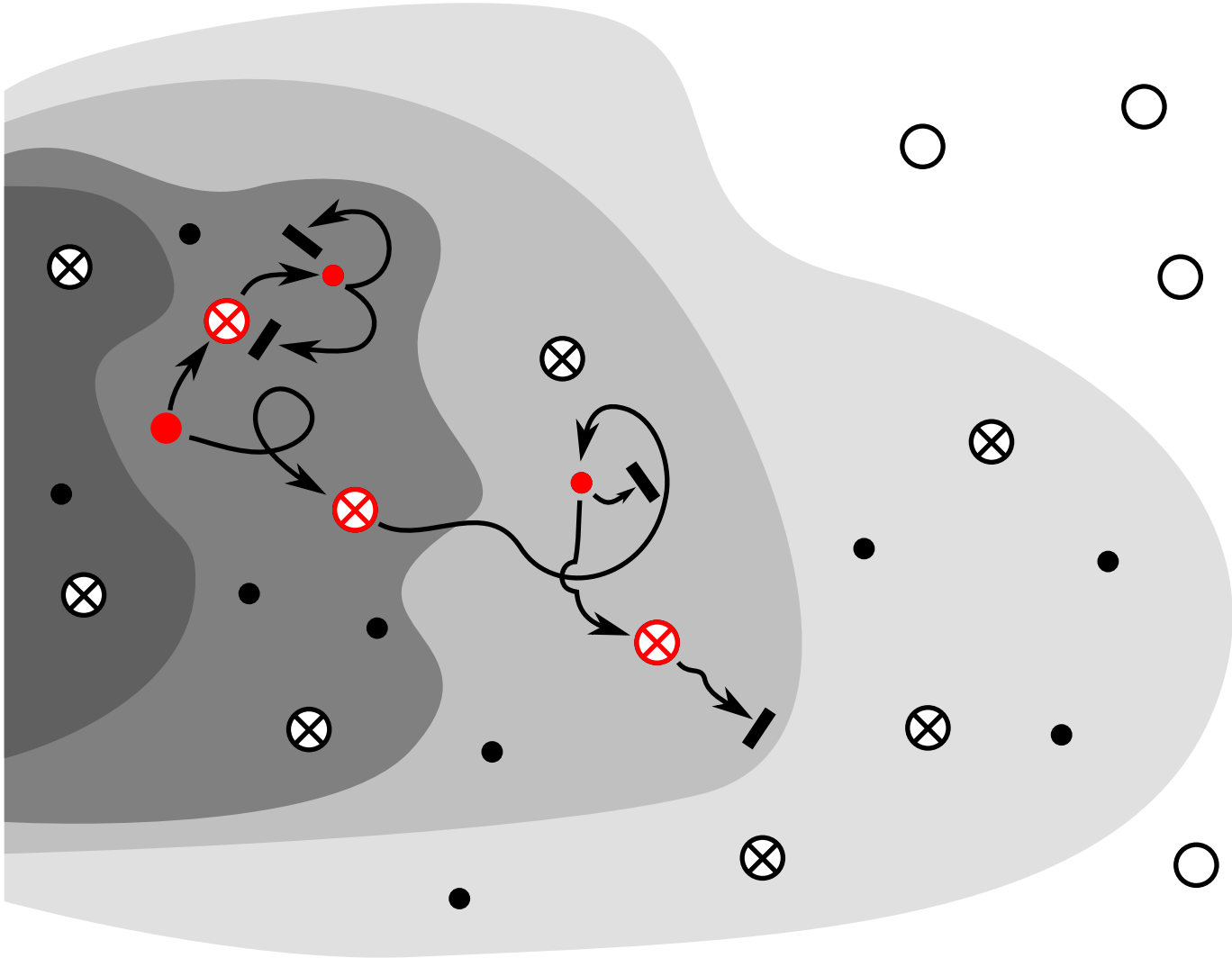
Algorithm — example



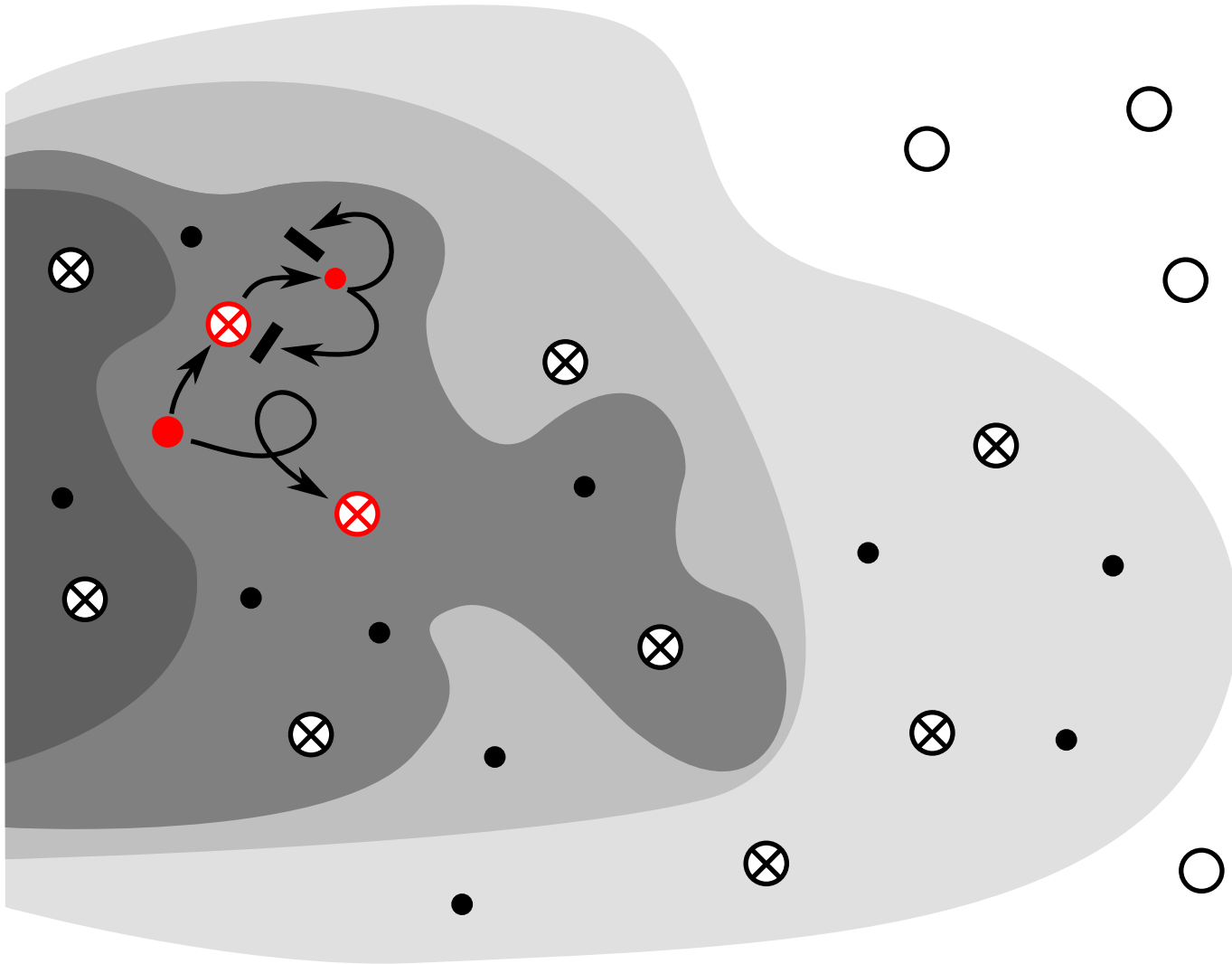
Algorithm — example



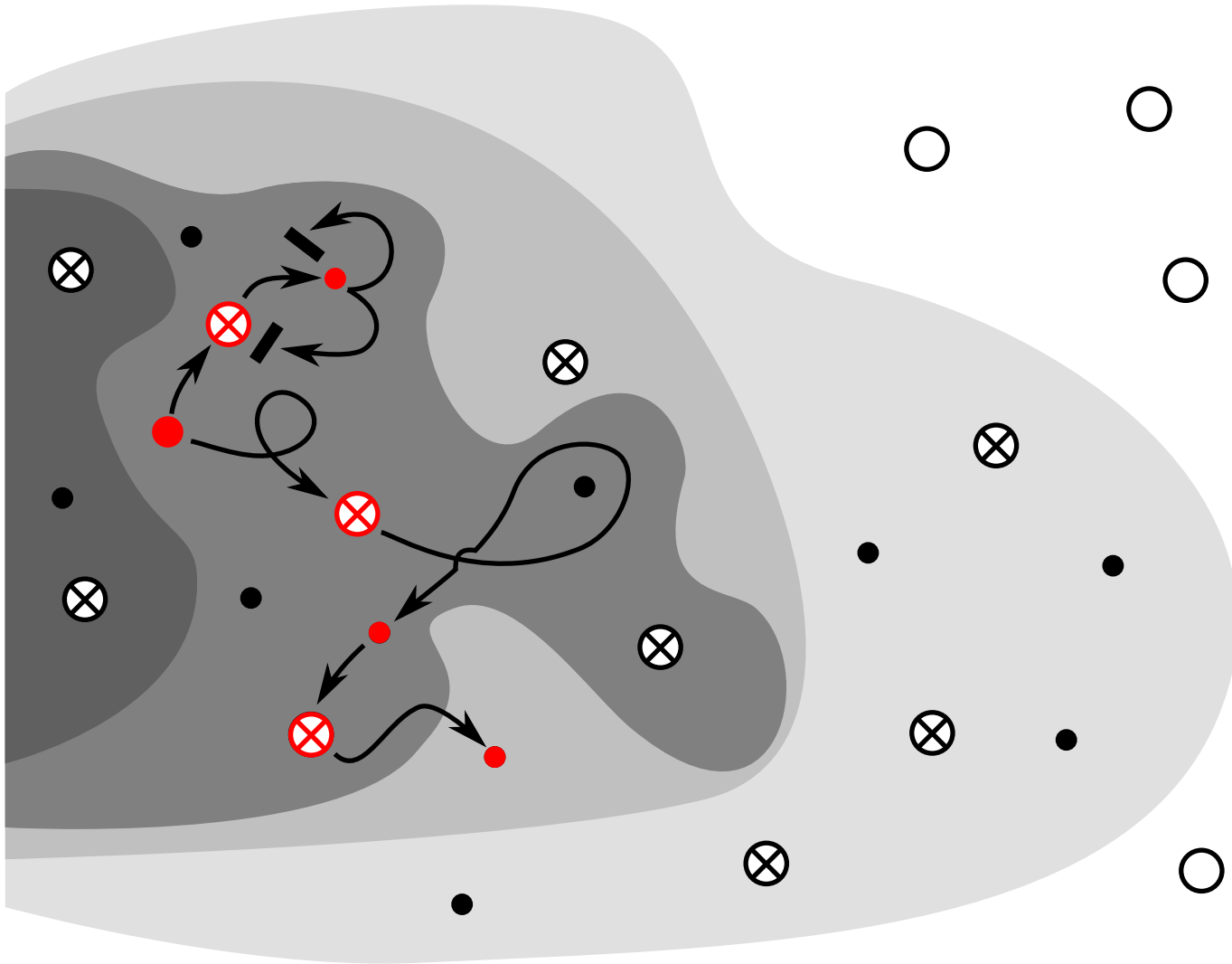
Algorithm — example



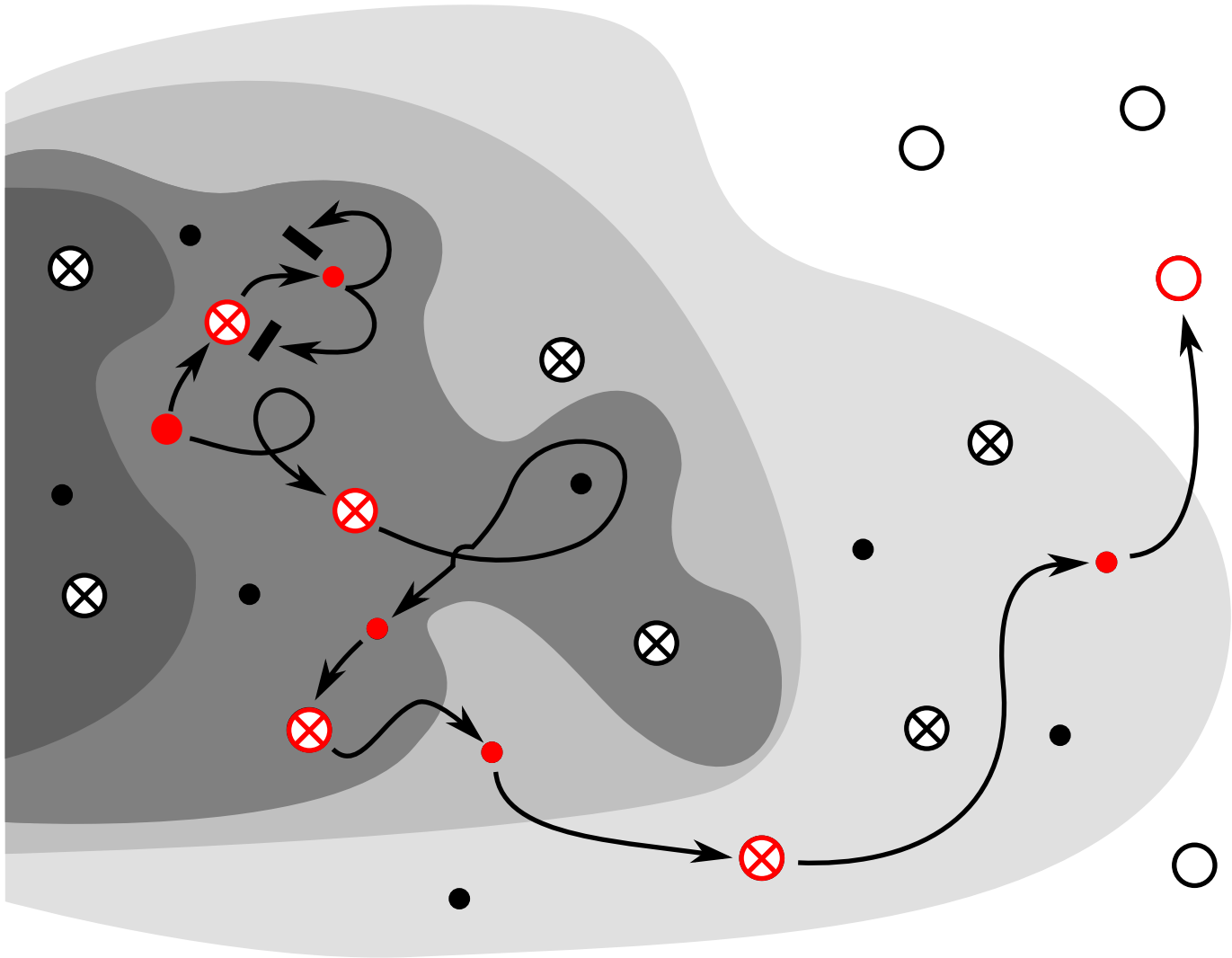
Algorithm — example



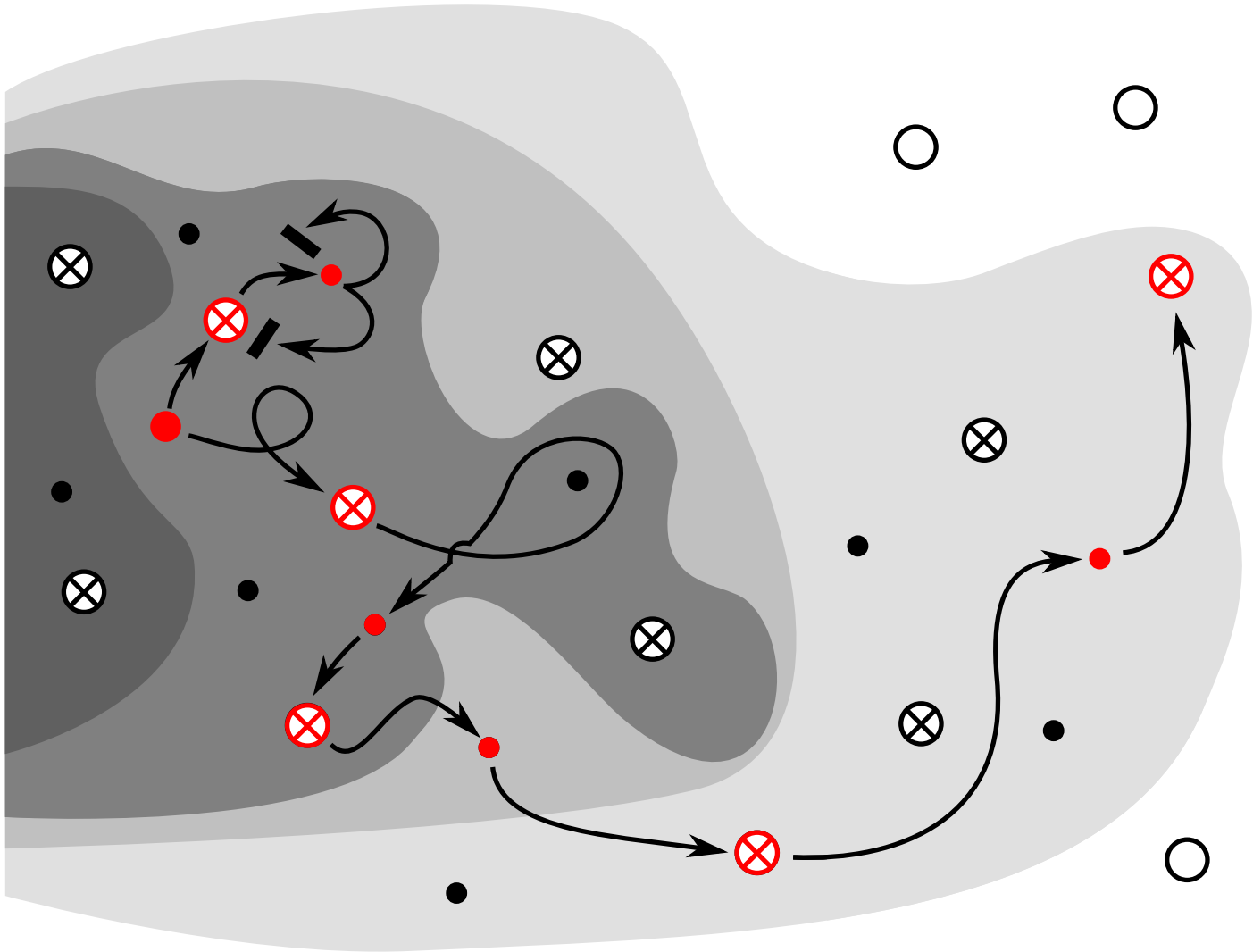
Algorithm — example



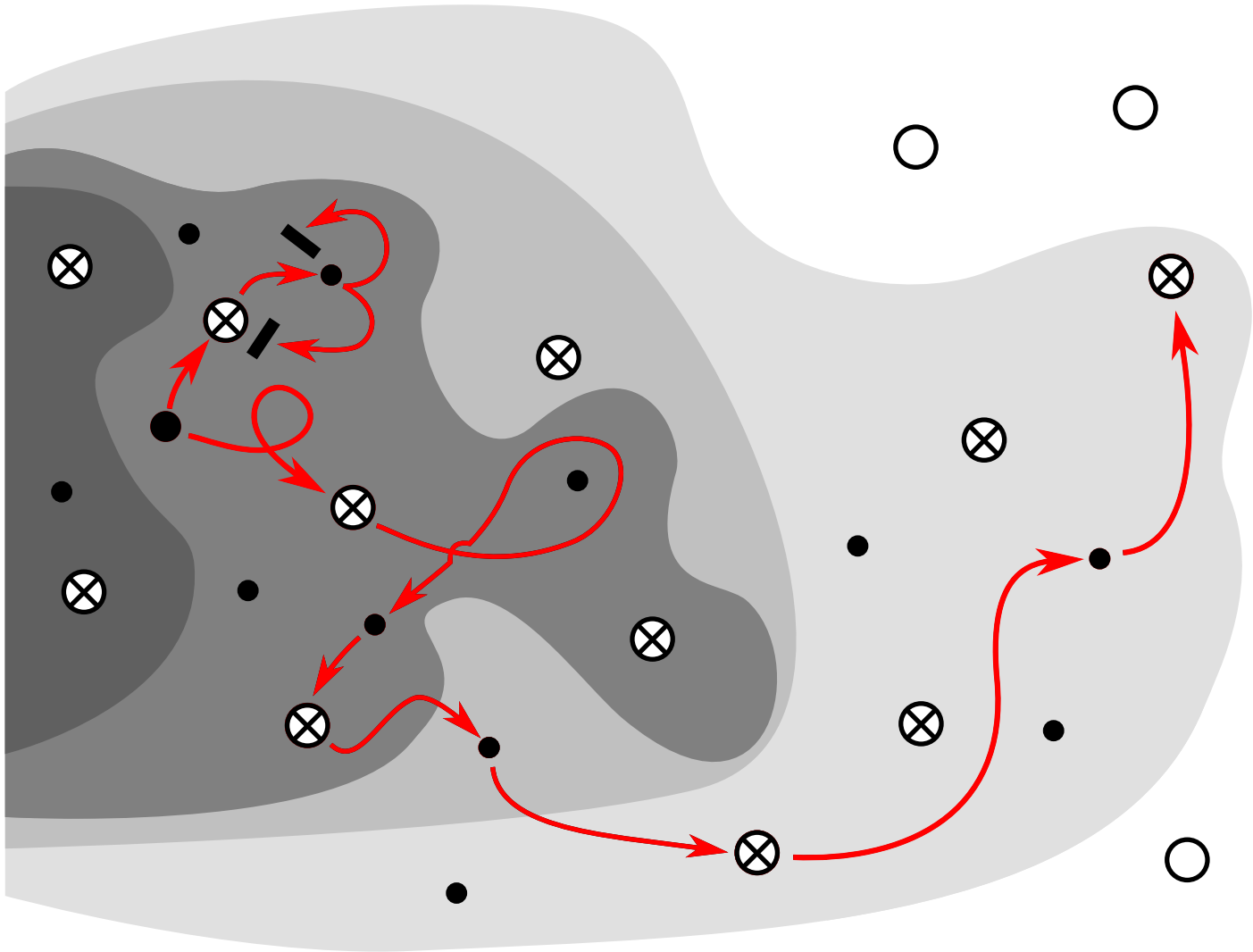
Algorithm — example



Algorithm — example



Algorithm — example



Algorithm — approximation

- We set an artificial upper bound tierLB $\leq k$.
- Vertices that went over are left unmatched.
- If the ranks of all augmenting paths are $\geq k$, the matching is $(1 - 2/k)$-approximate.
- With $\mathcal{O}(\varepsilon^{-1})$ changes per vertex, we get $(1 - \varepsilon)$-approximation in $\mathcal{O}(\varepsilon^{-1}m)$ time.

Concluding remarks

- New type of augmenting path algorithms.
- The reallocation problem has bound $2\sqrt{n}$.
- Incremental or decremental,
  exact or approximate versions.
- It's simple and combinatorial.
- Can the bound of $\mathcal{O}(n \log n)$ be reached?

Concluding remarks

- New type of augmenting path algorithms.
- The reallocation problem has bound $2\sqrt{n}$.
- Incremental or decremental,
  exact or approximate versions.
- It's simple and combinatorial.
- Can the bound of $\mathcal{O}(n \log n)$ be reached?

Thank You!