Approximation Algorithms

Max-3SAT, Max-k-Coverage, Set Cover, Max-Cut

Max-3SAT

[Max-3SAT]

- Input: a 3-CNF Boolean formula ϕ
- Output: an assignment satisfying maximum number of clauses

Assumption:

1. Each clause contains exactly 3 literals
2. Each clause contains 3 distinct variables

What if we assign values randomly?

- For each x_i, assign
  - $x_i = \text{true}$ with probability 0.5;
  - $x_i = \text{false}$ with probability 0.5.
- What is the probability that a clause is satisfied?
- What is the number of satisfied clauses in expectation?

Linearity of Expectation

- **Theorem**. Let
  - $X_1, \dots, X_n$ be n random variables that may be dependent, and
  - $c_1, \dots, c_n$ be n constants.
- We have

$$\mathbb{E}\left[\sum_{i=1}^{n} c_i X_i\right] = \sum_{i=1}^{n} c_i \mathbb{E}[X_i].$$

Max-3SAT Random Assignment

- For each $i = 1, \ldots, m$, define random variable
$$Y_i = \begin{cases} 1, & \text{if } i\text{th clause is satisfied} \\ 0, & \text{otherwise} \end{cases}$$
- We have $\mathbb{E}[Y_i] = 1 \times \Pr(Y_i = 1) + 0 \times \Pr(Y_i = 0) = \frac{7}{8}$.
- $Y = \sum_{i=1}^{m} Y_i$: total number of satisfied clauses
- We want to compute $\mathbb{E}[Y]$.
- By Linearity of Expectation:
$$\mathbb{E}[Y] = \mathbb{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbb{E}[Y_i] = \frac{7}{8}m.$$

A $\frac{7}{8}$-Approximation Algorithm?

- m is clearly an upper bound to OPT.
- If we can satisfied $\geq \frac{7}{8}m$ clauses, we get a $\frac{7}{8}$-Approximation Algorithm!

Let's try to assign value to x_1

- We have
$$\mathbb{E}[Y] = \mathbb{E}[Y|x_1 = \text{true}] \cdot \Pr(x_1 = \text{true}) + \mathbb{E}[Y|x_1 = \text{false}] \cdot \Pr(x_1 = \text{false})$$
$$= \frac{1}{2} \cdot \mathbb{E}[Y|x_1 = \text{true}] + \frac{1}{2} \cdot \mathbb{E}[Y|x_1 = \text{false}]$$
- which implies
$$\mathbb{E}[Y|x_1 = \text{true}] + \mathbb{E}[Y|x_1 = \text{false}] = 2 \cdot \mathbb{E}[Y].$$
- Thus, either $\mathbb{E}[Y|x_1 = \text{true}] \geq \mathbb{E}[Y]$ or $\mathbb{E}[Y|x_1 = \text{false}] \geq \mathbb{E}[Y]$.
- The two conditional expectations can be computed in $O(m)$ time.
- We can assign value to x_1 with larger conditional expectation!

Example

- $\phi = (x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4)$
- Assigning $x_1 = \text{true}$ results in
  - $\phi = \text{true} \land \text{true} \land (\neg x_2 \lor x_4)$
  - $\mathbb{E}[Y|x_1 = \text{true}] = 1 + 1 + \frac{3}{4} = 2.75$
- Assigning $x_1 = \text{false}$ results in
  - $\phi = (x_3 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land \text{true}$
  - $\mathbb{E}[Y|x_1 = \text{false}] = \frac{3}{4} + \frac{3}{4} + 1 = 2.5$
- We shall assign $x_1 = \text{true}$.

Continue for x_2...

- After assigning some value for x_1:
- $x_1 = v_1$ where $v_1 \in \{\text{true}, \text{false}\}$
- We assign value for x_2 by comparing
- $\mathbb{E}[Y|x_1 = v_1, x_2 = \text{true}], \mathbb{E}[Y|x_1 = v_1, x_2 = \text{false}]$
- Assign $x_2 = v_2 \in \{\text{true}, \text{false}\}$ with larger conditional expectation.

An Approximation Algorithm

1. for $i = 1, \dots, n$:
2. compute $\mathbb{E}[Y | x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{true}], \mathbb{E}[Y | x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{false}]$
3. assign $x_i = v_i \in \{\text{true}, \text{false}\}$ with the larger conditional expectation
4. endfor

Expectation Monotonicity

In each iteration:

$$\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$$
$$= \frac{1}{2}\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{true}] + \frac{1}{2}\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{false}]$$

Thus, either

- $\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{true}] \geq \mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$, or
- $\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{false}] \geq \mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$

The algorithm always choose $x_i = v_i \in \{\text{true}, \text{false}\}$ with larger expectation:

$$\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = v_i] \geq \mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$$

The conditional expectation for Y is non-decreasing!

Expectation Monotonicity

- The conditional expectation for Y is non-decreasing!
- Thus, $\mathbb{E}[Y|x_1 = v_1, \dots, x_n = v_n] \geq \mathbb{E}[Y] = \frac{7}{8}m$.
- $\mathbb{E}[Y|x_1 = v_1, \dots, x_n = v_n]$ is already deterministic.
  - With assignment $x_1 = v_1, \dots, x_n = v_n$, this is exactly the number of satisfied clauses!
- We have a $\frac{7}{8}$-approximation algorithm!
- Running Time: $O(mn)$

Possible Improvements?

- Can this algorithm do better than $\frac{7}{8}$-approximation?
- No…
- Easy to come up with a tight example…

Possible Improvements?

- Exist other better algorithms?
- Assuming **P** $\neq$ **NP**, no…
- [Håstad, 2001] Max-3SAT is NP-hard to approximate to within $\frac{7}{8}+\varepsilon$ for any $\varepsilon > 0$.

Maximum Independent Set (Clique)

- For any $\varepsilon > 0$, Maximum Independent Set/Clique is NP-hard to approximate to within factor $(|V|^{1-\varepsilon})$.
  – [Håstad, 1999], [Khot, 2001] and [Zuckerman, 2006]
- Can you give a $|V|$-approximation algorithm?
- An $O\left(\frac{|V|(\log\log|V|)^2}{(\log|V|)^3}\right)$-approximation algorithm…
  – [Feige, 2004]

Greedy-Based Approximation Algorithm

- Greedy algorithm may not output optimal solutions for some optimization problems.
- However, it may be a good approximation algorithm!

Max-k-Coverage and Set Cover Problems

- Let $U = \{1, \ldots, n\}$ be a ground set of elements.
- Let $T = \{A_1, A_2, \ldots, A_m\}$ be a collection of subsets of U with $\bigcup_{A_i \in T} A_i = U$.
- [Set Cover] Find a sub-collection $S \subseteq T$ with minimum $|S|$ such that $\bigcup_{A_i \in S} A_i = U$.
- [Max-k-Coverage] Given $k \in \mathbb{Z}^+$, find a sub-collection $S \subseteq T$ with $|S| \leq k$ that maximizes $\left|\bigcup_{A_i \in S} A_i\right|$.

NP-Hardness

- Given $k \in \mathbb{Z}^+$, it is NP-complete to decide if there exists $S \subseteq T$ with $|S| \leq k$ such that $\bigcup_{A_i \in S} A_i = U$.
- Exercise: Prove it!
- Therefore, both max-k-coverage and set cover are NP-hard.

Notation

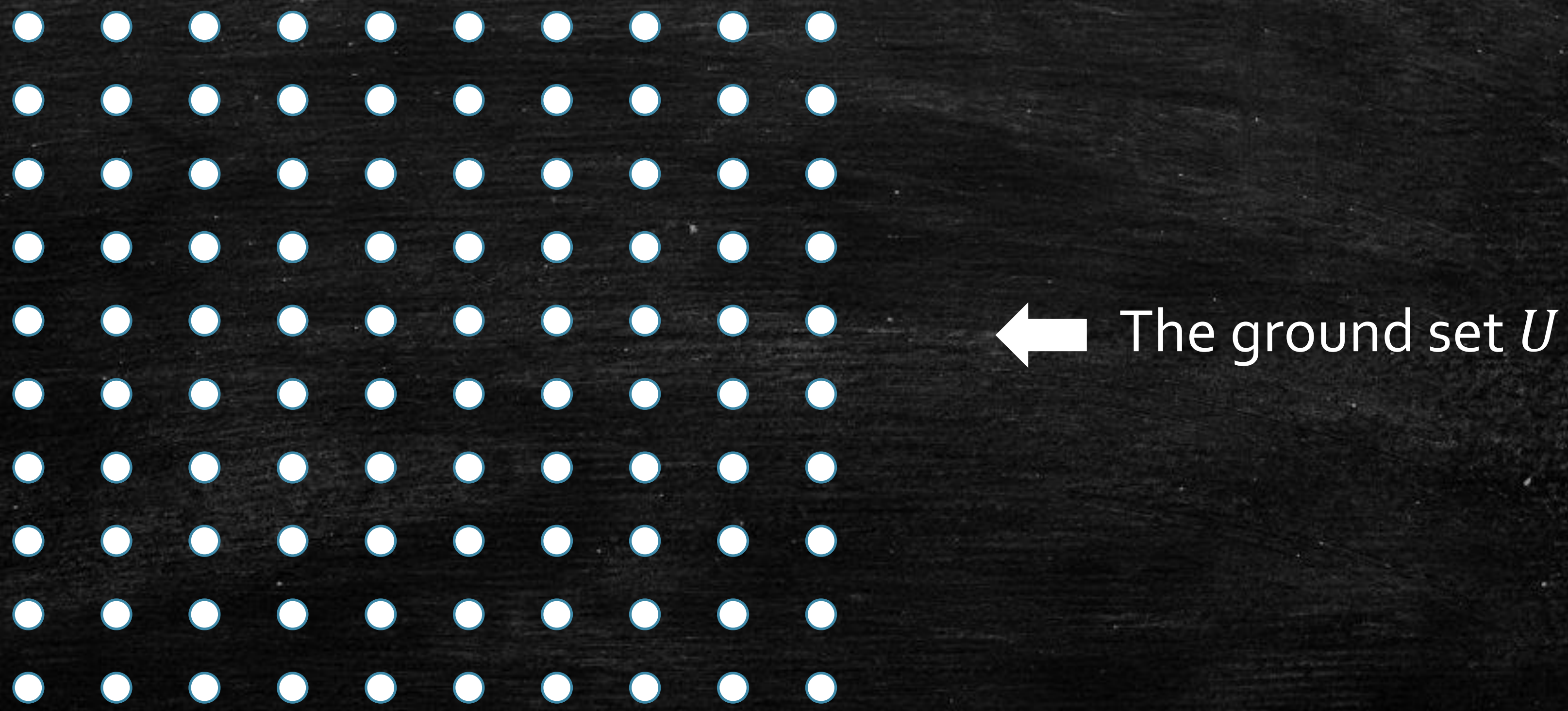
- Denote $f(S) = \left|\bigcup_{A_i \in S} A_i\right|$: the number of elements covered by S.
- [Set Cover] Find minimum-sized S with $f(S) = |U| = n$.
- [Max-k-Coverage] Maximize $f(S)$ subject to $|S| \leq k$.

Greedy Algorithm

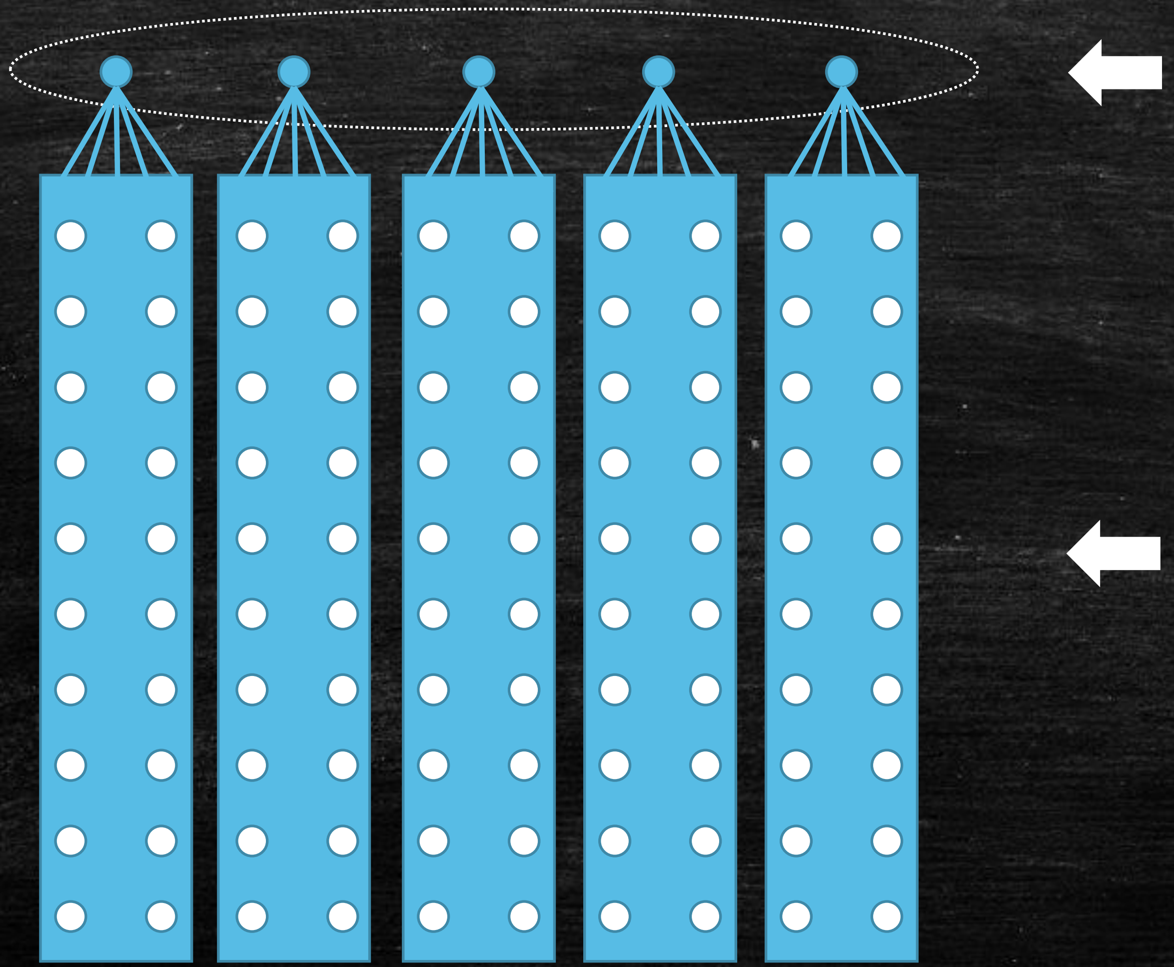
1. Initialize $S \leftarrow \emptyset$
2. Repeat the followings:
3. find $A \in T \setminus S$ that maximizes $f(S \cup \{A\}) - f(S)$
4. update $S \leftarrow S \cup \{A\}$
5. Until:
   - $f(S) = |U| = n$ (for set cover)
   - $|S| = k$ (for max-k-coverage)
6. Return S

Performance of Greedy Algorithm

- $U = \{1, \ldots, n\}$: ground set of elements
- $T = \{A_1, A_2, \ldots, A_m\}$: a collection of subsets of U



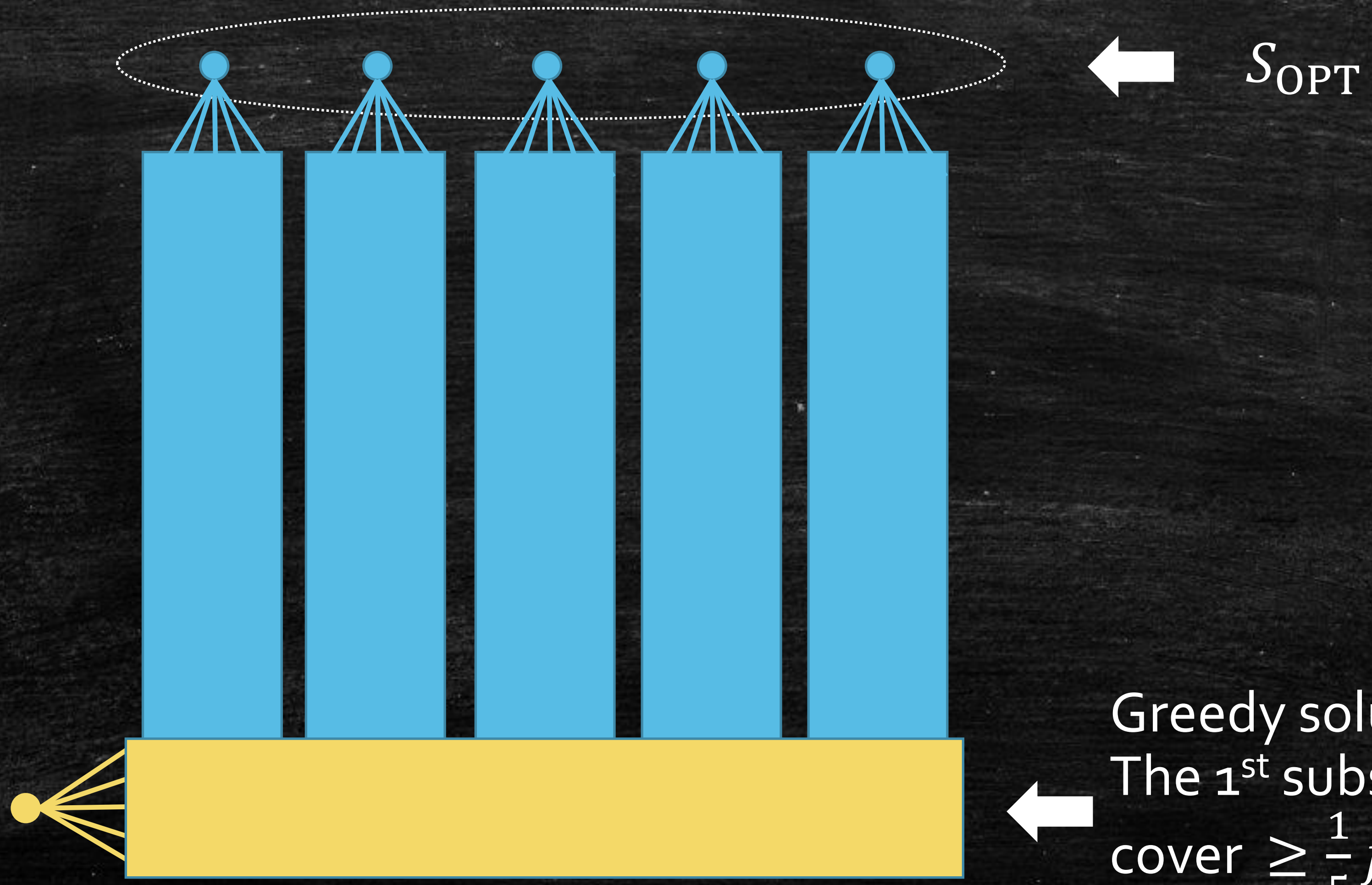
Performance of Greedy Algorithm



S_{OPT}

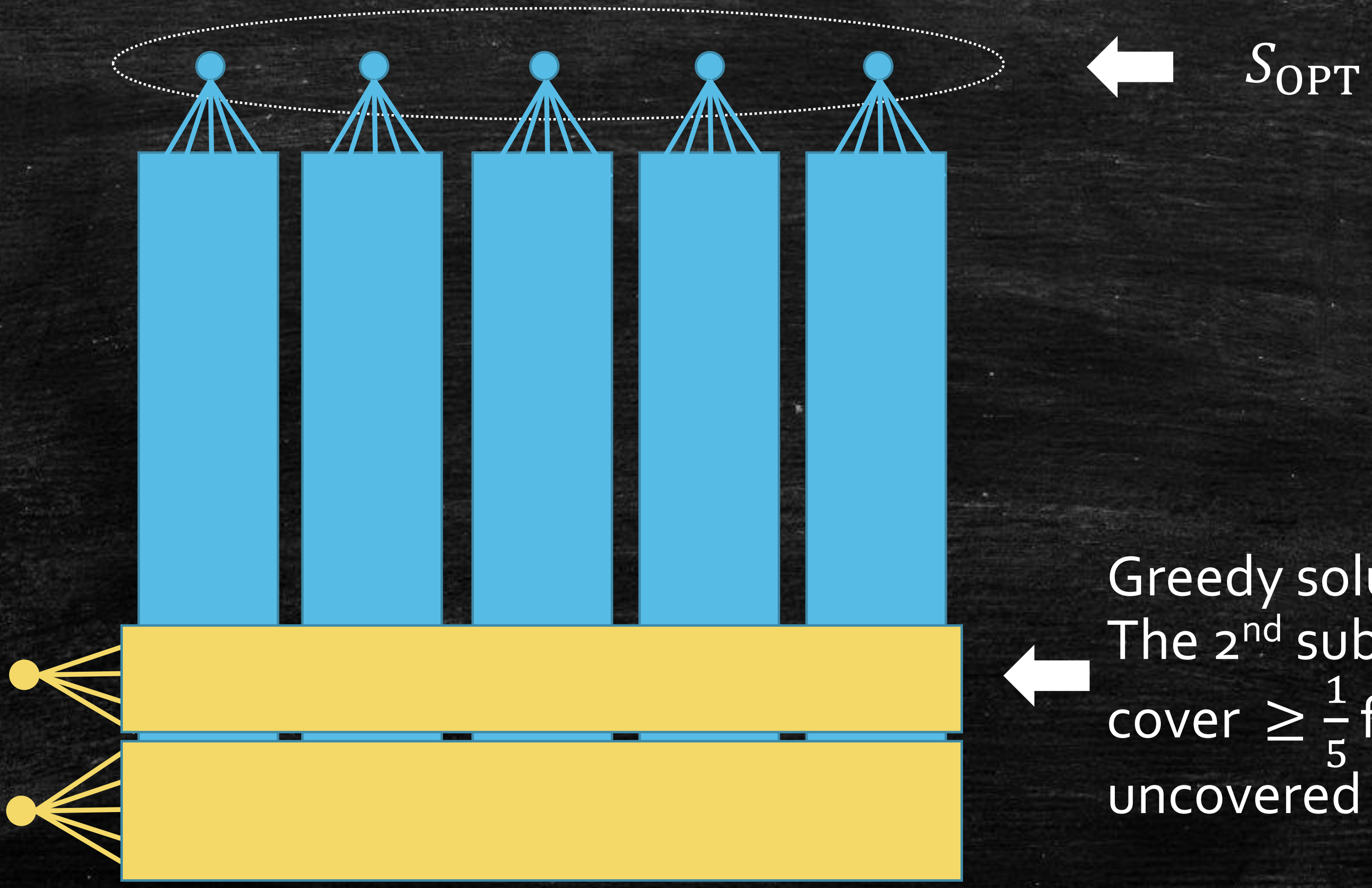
Optimal solution:
5 subsets covers all elements

Performance of Greedy Algorithm



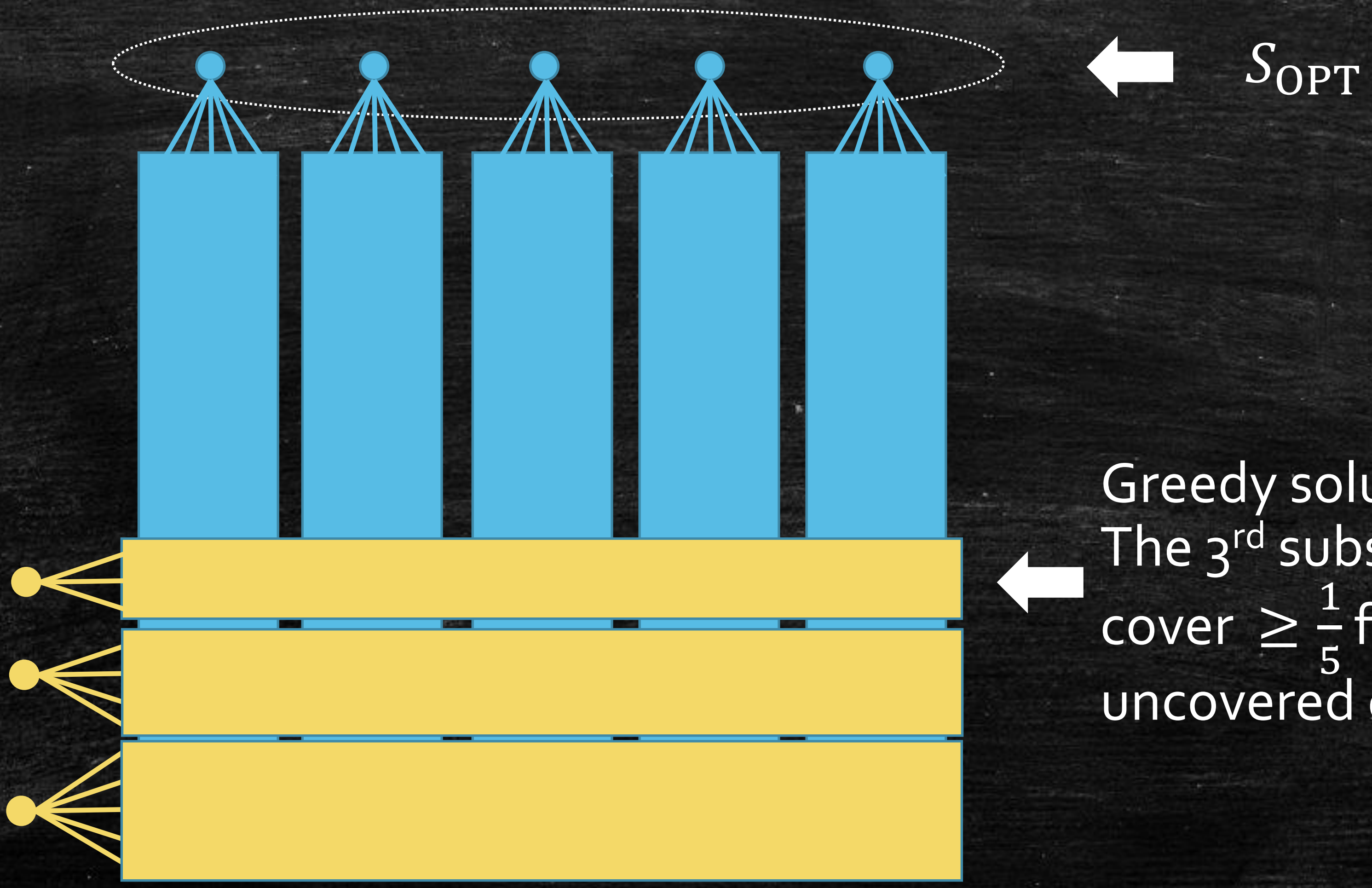
Greedy solution:
The 1st subset selected must cover $\geq \frac{1}{5} f(S_{\text{OPT}})$ elements

Performance of Greedy Algorithm



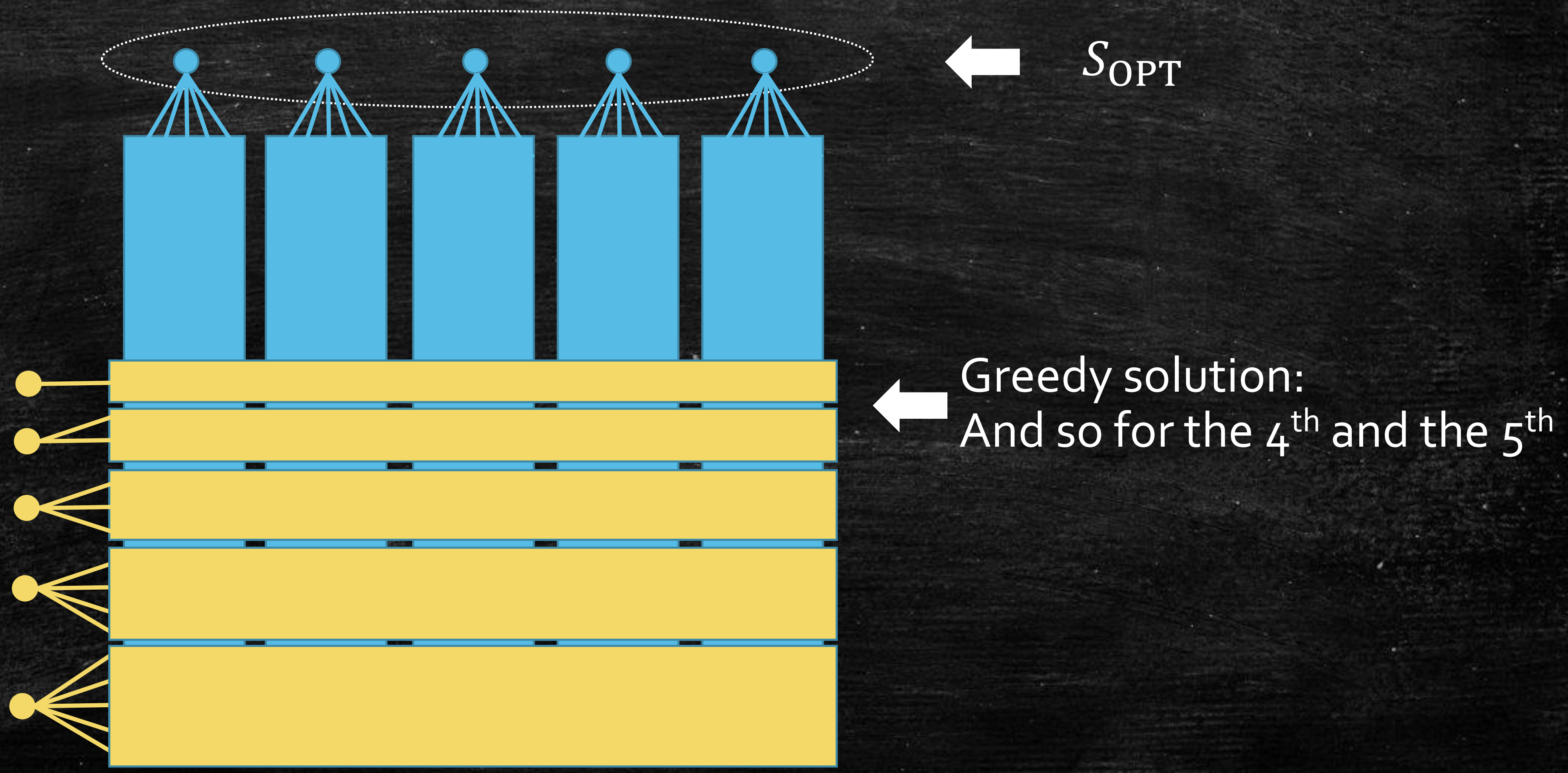
Greedy solution:
The 2nd subset selected must cover $\geq \frac{1}{5}$ fraction of remaining uncovered elements

Performance of Greedy Algorithm



Greedy solution:
The 3rd subset selected must cover $\geq \frac{1}{5}$ fraction of remaining uncovered elements

Performance of Greedy Algorithm



Performance of Greedy Algorithm

- Let $S = (A_1, \ldots, A_5)$ be the output of the greedy algorithm.
- $\{A_1\}$ covers $\frac{1}{5}$ fraction
- $\{A_1, A_2\}$ covers $\frac{1}{5} + \frac{1}{5}\left(1 - \frac{1}{5}\right) = 1 - \left(1 - \frac{1}{5}\right)^2$ fraction
- $\{A_1, A_2, A_3\}$: $1 - \left(1 - \frac{1}{5}\right)^2 + \frac{1}{5}\left(1 - \left(1 - \left(1 - \frac{1}{5}\right)^2\right)\right) = 1 - \left(1 - \frac{1}{5}\right)^3$
- $\{A_1, A_2, A_3, A_4\}$: $1 - \left(1 - \frac{1}{5}\right)^4$
- $\{A_1, A_2, A_3, A_4, A_5\}$: $1 - \left(1 - \frac{1}{5}\right)^5$

Performance of Greedy Algorithm

- Let $S^* = \{O_1, O_2, \ldots, O_k\}$ be any collection of k subsets.
- Let $S = \{A_1, A_2, \ldots, A_\ell\}$ be the output of greedy after ℓ iterations.
- **Lemma**. $f(S) \geq \left(1 - \left(1 - \frac{1}{k}\right)^\ell\right) f(S^*)$.
- Greedy gives a $\left(1 - \frac{1}{e}\right)$-approximation for max-k-coverage:
  - For optimal S^*, we have $f(S) \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) f(S^*) \geq \left(1 - \frac{1}{e}\right) f(S^*)$.
- Greedy gives a $(\ln n)$-approximation for set cover:
  - Suppose S^* with $|S^*| = k$ is optimal.
  - For $\ell = k \cdot \ln n$, $f(S) \geq \left(1 - \left(1 - \frac{1}{k}\right)^{k \cdot \ln n}\right) f(S^*) > \left(1 - \frac{1}{e^{\ln n}}\right) f(S^*) = n - 1$
  - This implies $f(S) = n$, as $f(S) \in \mathbb{Z}^+$

Proving $f(S) \geq \left(1 - \left(1 - \frac{1}{k}\right)^{\ell}\right) f(S^*)$

- Let $S_t = \{A_1, \dots, A_t\}$
- Prove lemma by Induction…
- Base Step $\ell = 1$:
- By greedy nature, $f(S_1 = \{A_1\}) \geq f(\{O_i\})$ for all O_i.
- Thus, $f(S_1) \geq \frac{1}{k} \sum_{i=1}^{k} f(\{O_i\}) \geq \frac{1}{k} f(S^*) = \left(1 - \left(1 - \frac{1}{k}\right)^{1}\right) f(S^*)$
- Middle inequality: Elements in more than one O_i is counted more than once in $\sum_{i=1}^{k} f(\{O_i\})$, and only once in $f(S^*)$.

Proving $f(S) \geq \left(1-\left(1-\frac{1}{k}\right)^{\ell}\right) f(S^*)$

- Now, $S_t = \{A_1, \ldots, A_t\}$ after t iterations.
- For each O_i, consider $\Delta(O_i \mid S_t) = f(S_t \cup \{O_i\}) - f(S_t)$.
- By greedy nature, $\Delta(A_{t+1} \mid S_t) \geq \Delta(O_i \mid S_t)$ for each O_i.
- $\Delta(A_{t+1} \mid S_t) \geq \frac{1}{k} \sum_{i=1}^{k} \Delta(O_i \mid S_t) \geq \frac{1}{k} \Delta(S^* \mid S_t)$

Proving $f(S) \geq \left(1 - \left(1 - \frac{1}{k}\right)^{\ell}\right) f(S^*)$

- We have $\Delta(A_{t+1} | S_t) \geq \frac{1}{k} \sum_{i=1}^{k} \Delta(O_i \mid S_t) \geq \frac{1}{k} \Delta(S^* \mid S_t)$
- Inductive step: $f(S_{t+1}) - f(S_t) \geq \frac{1}{k}\left(f(S^* \cup S_t) - f(S_t)\right)$ (yellow)
- $\geq \frac{1}{k}\left(f(S^*) - f(S_t)\right)$ (monotonicity of f)
- $f(S_{t+1}) \geq \frac{1}{k} f(S^*) + \left(1 - \frac{1}{k}\right) f(S_t)$ (rearranging inequality)
- $\geq \frac{1}{k} f(S^*) + \left(1 - \frac{1}{k}\right)\left(1 - \left(1 - \frac{1}{k}\right)^{t}\right) f(S^*)$ (induction hypothesis)
- $= \left(1 - \left(1 - \frac{1}{k}\right)^{t+1}\right) f(S^*)$

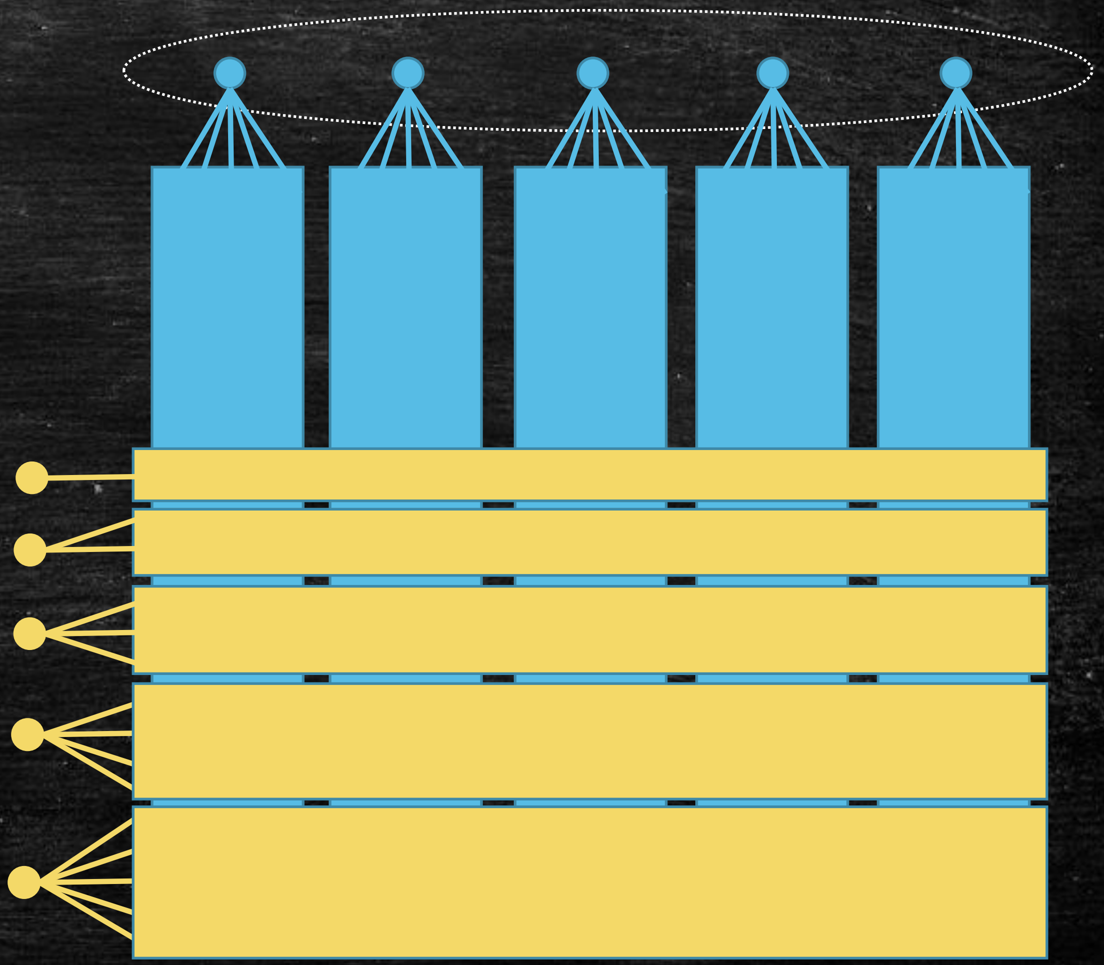
Performance of Greedy Algorithm

- Greedy gives a $\left(1-\frac{1}{e}\right)$-approximation for max-k-coverage.
  - For optimal S^*, we have $f(S) \geq \left(1-\left(1-\frac{1}{k}\right)^k\right) f(S^*) \geq \left(1-\frac{1}{e}\right) f(S^*)$.
- Greedy gives a $(\ln n)$-approximation for set cover.
  - Suppose S^* with $|S^*| = k$ is optimal.
  - For $\ell = k \cdot \ln n$, $f(S) \geq \left(1-\left(1-\frac{1}{k}\right)^{k \cdot \ln n}\right) f(S^*) > \left(1-\frac{1}{e^{\ln n}}\right) f(S^*) = n-1$
  - This implies $f(S) = n$, as $f(S) \in \mathbb{Z}^+$

Can greedy do better (by better analysis)?

This is also a Tight Example:

- Max-k-Coverage:
  - Greedy can do at best $1 - \frac{1}{e}$
- Set Cover:
  - Greedy can do at best $\ln n$



Better Algorithms?

Max-k-Coverage

- No $\left(1-\frac{1}{e}+\varepsilon\right)$-approximation algorithm unless **P** = **NP**.
  - [Feige, 1998]

Set Cover

- No $(1-o(1))\ln n$-approximation algorithm unless **NP** $\subseteq$ $\mathrm{DTIME}\left(n^{\mathrm{O}(\log\log n)}\right)$.
  - [Feige, 1998]
- No $(1-o(1))\ln n$-approximation algorithm unless **P** = **NP**.
  - [Moshkovitz, 2012] [Dinur & Steurer, 2014]

Local Search

- Start with an arbitrary solution.
- Improve it by “local updates”.
- Until no more update improves the objective.

Max-Cut

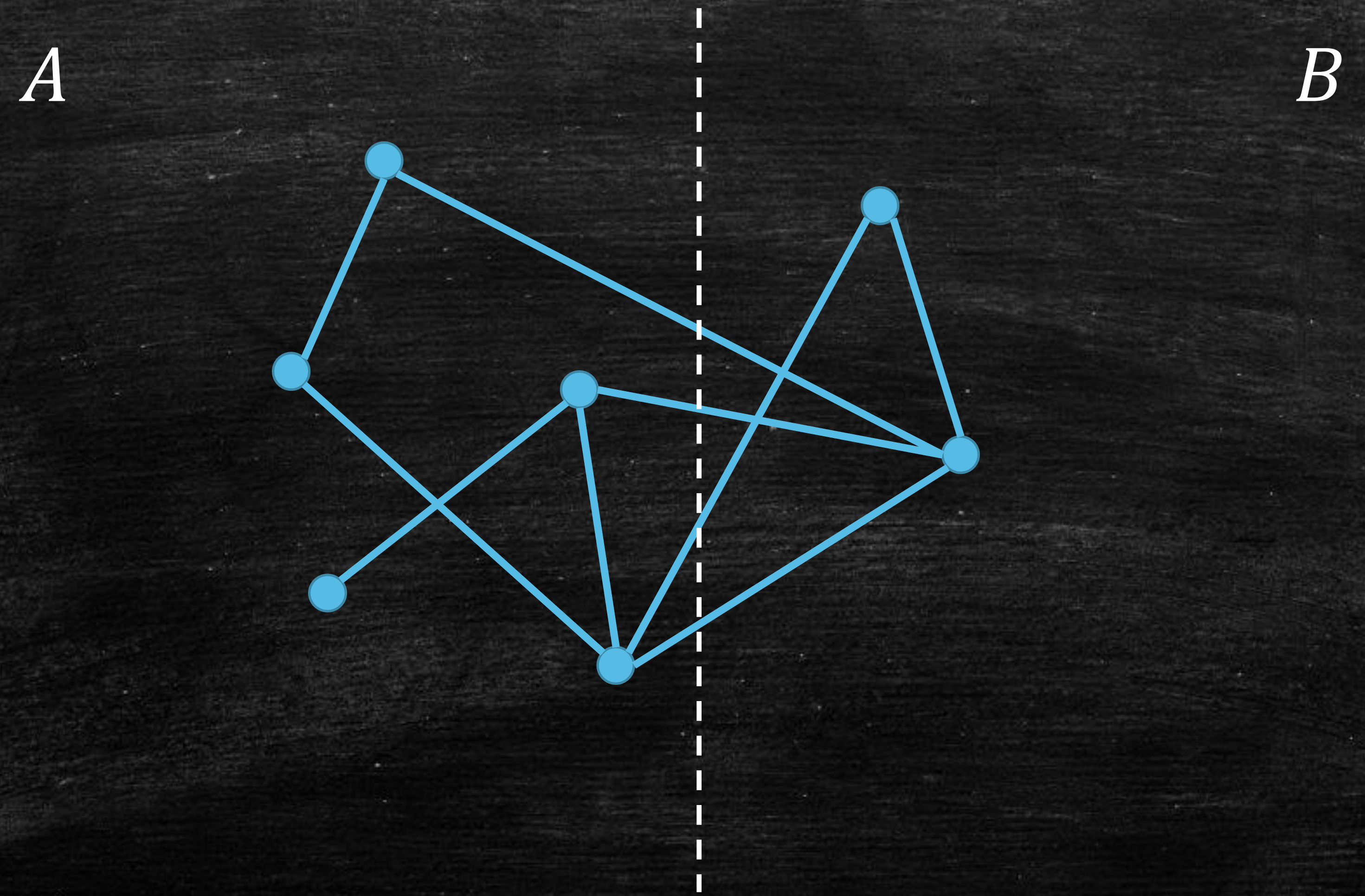
- [Max-Cut] Given an undirected graph $G = (V, E)$, find a cut (A, B) with maximum value $c(A, B) = |E(A, B)|$.

- [Karp, 1972] Max-Cut is NP-hard.

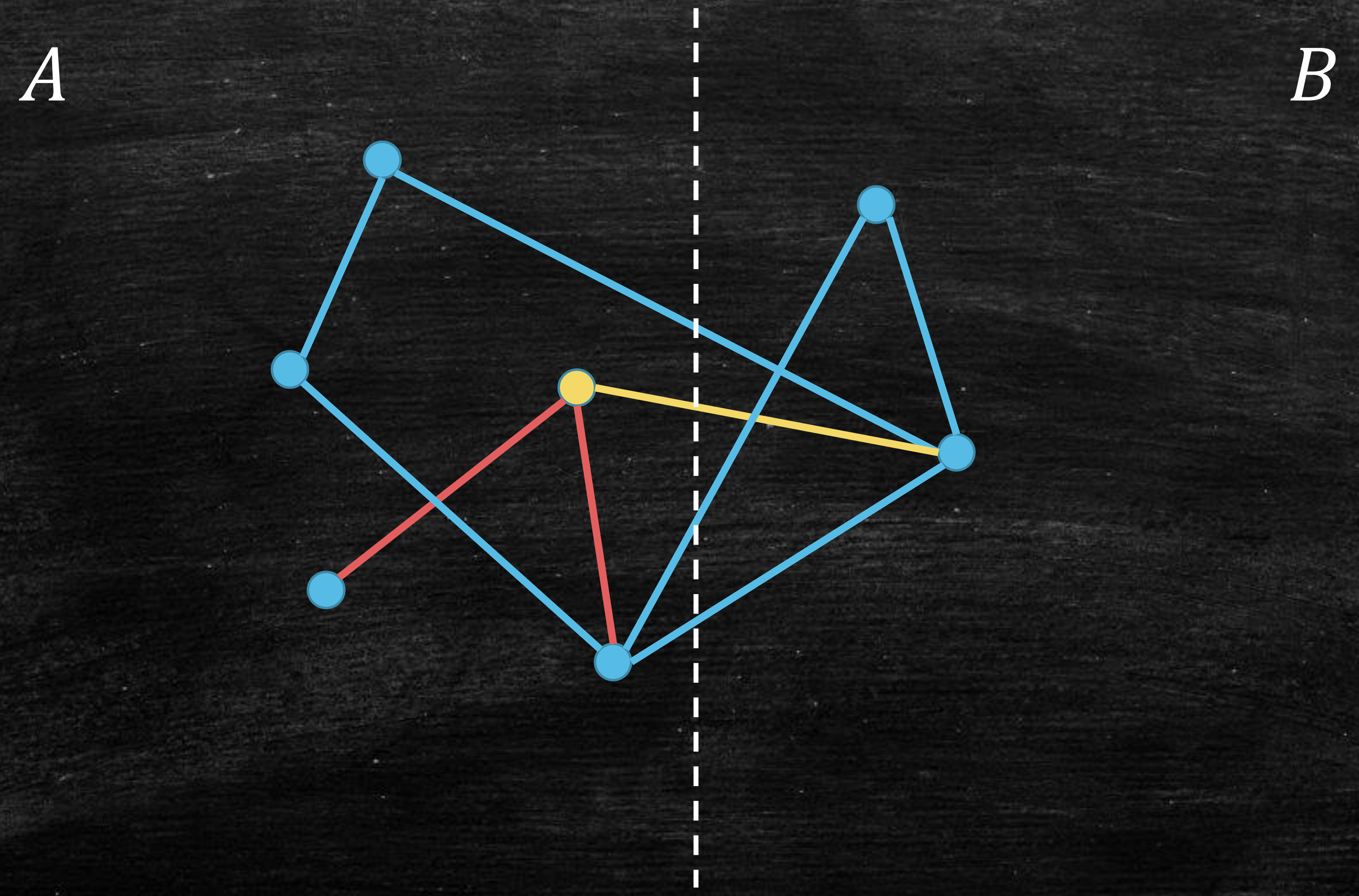
A Local Search Algorithm

1. Start with any partition (A, B).
2. If moving a vertex u from A to B or from B to A increases $c(A, B)$, move it.
3. Terminate until no such movement is possible.

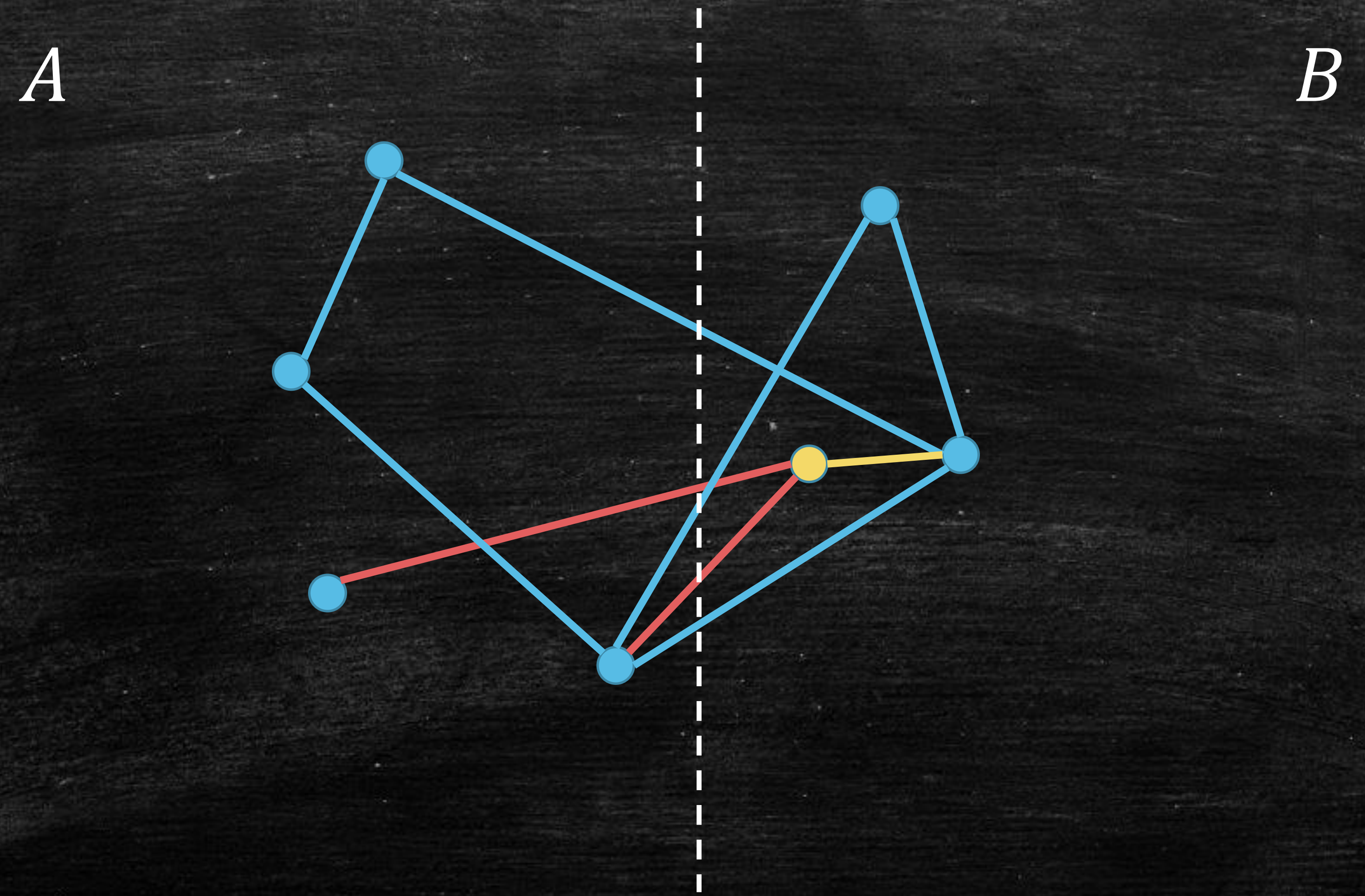
Example



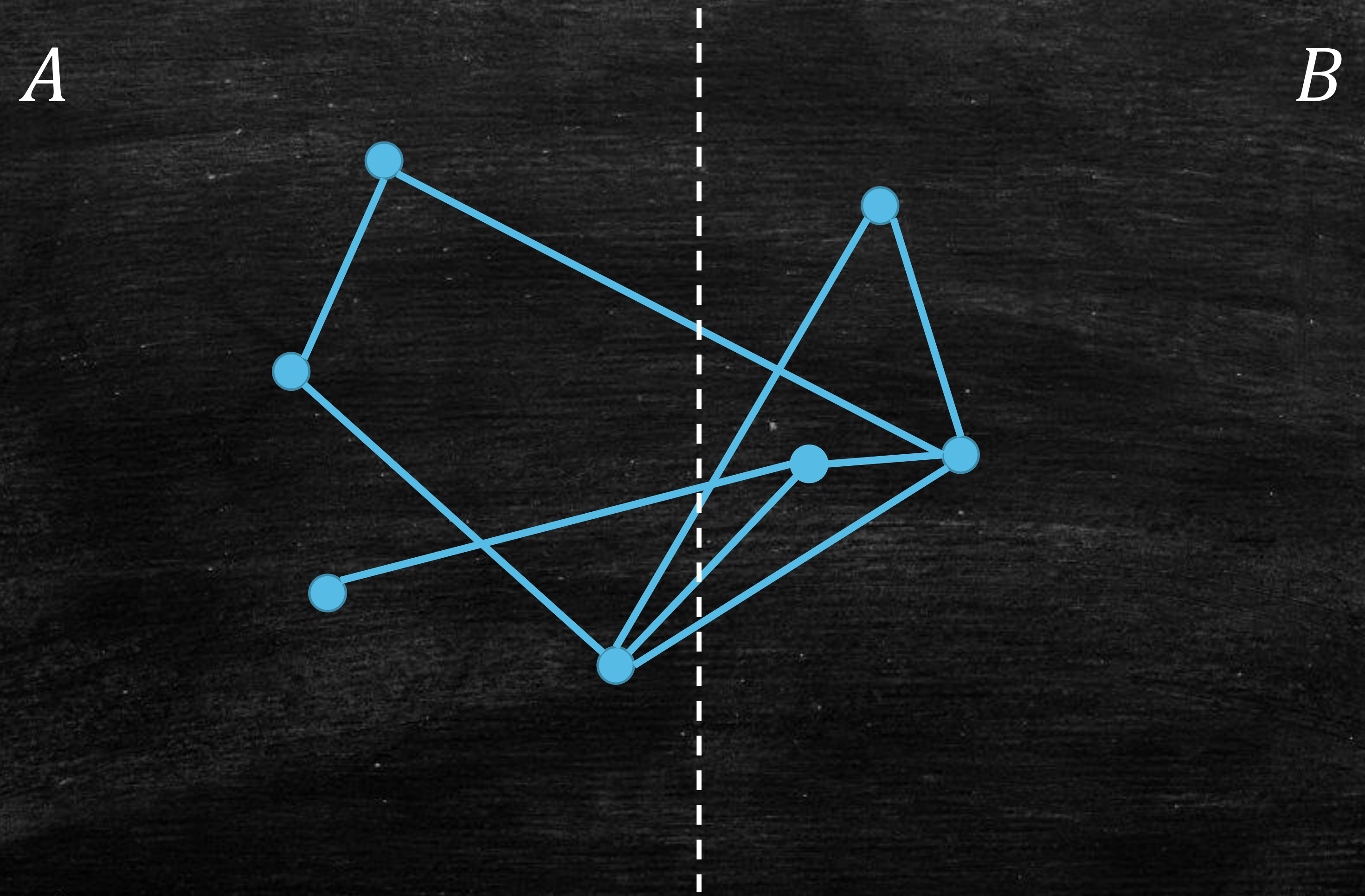
Example



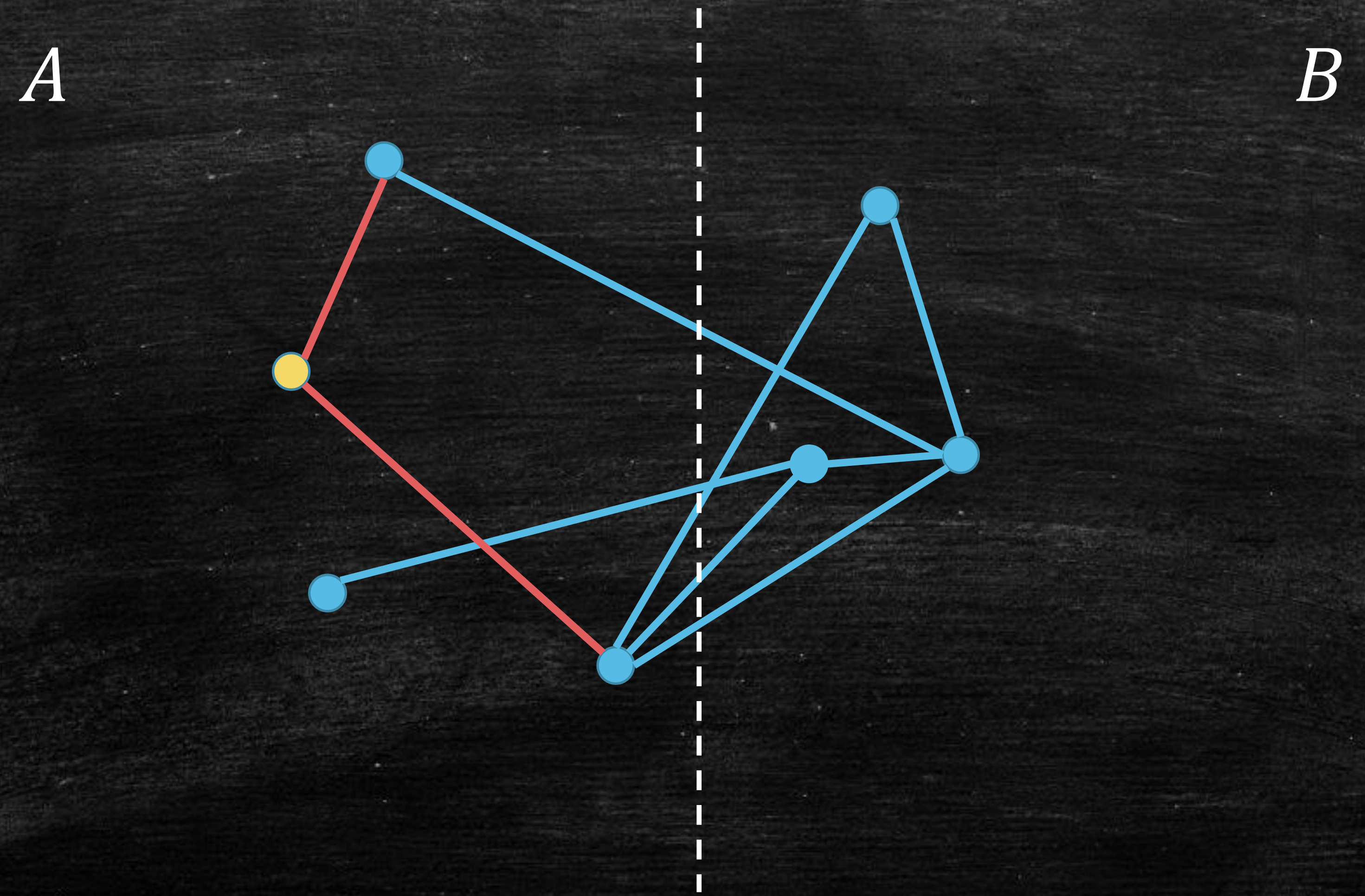
Example



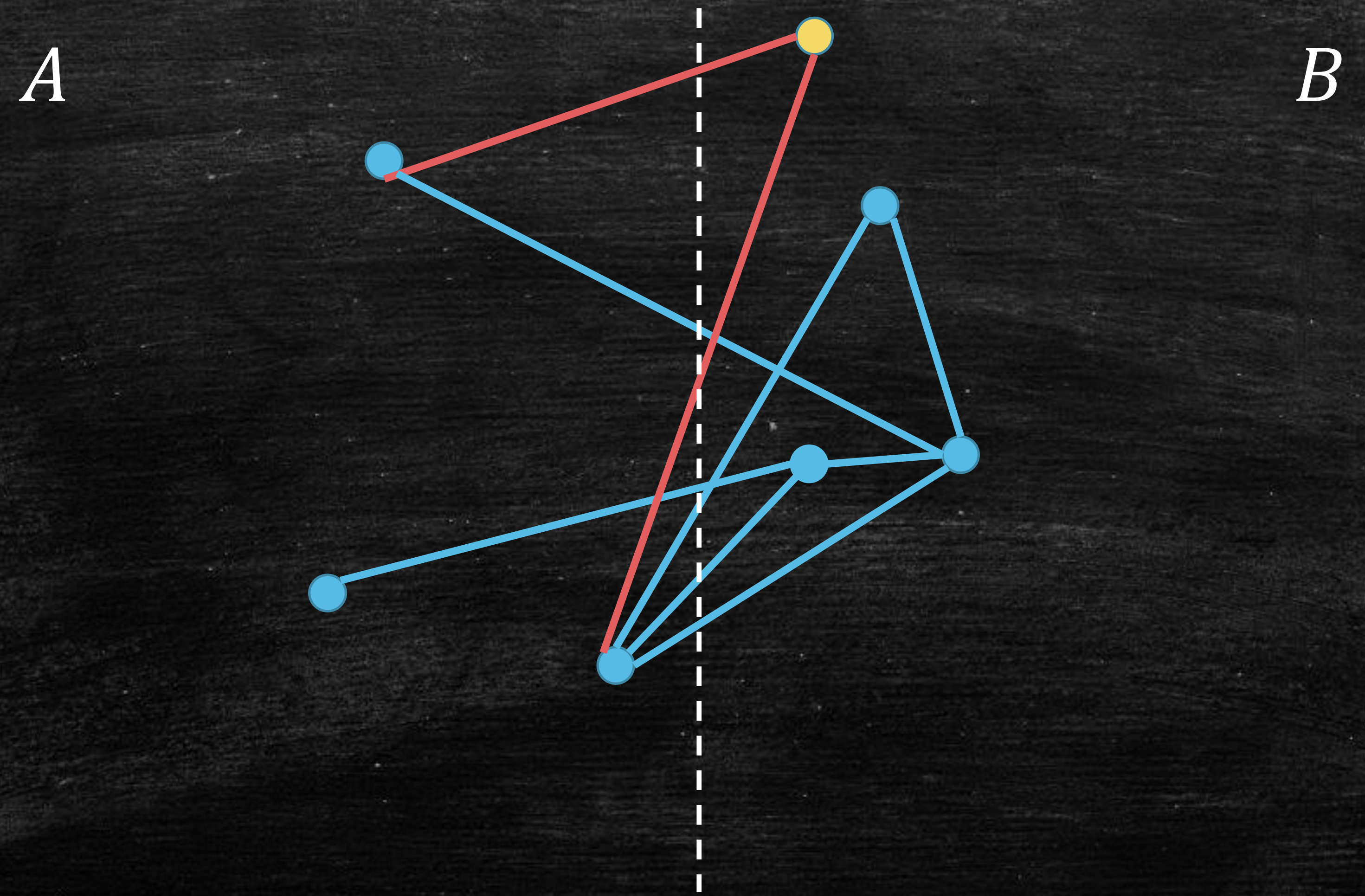
Example



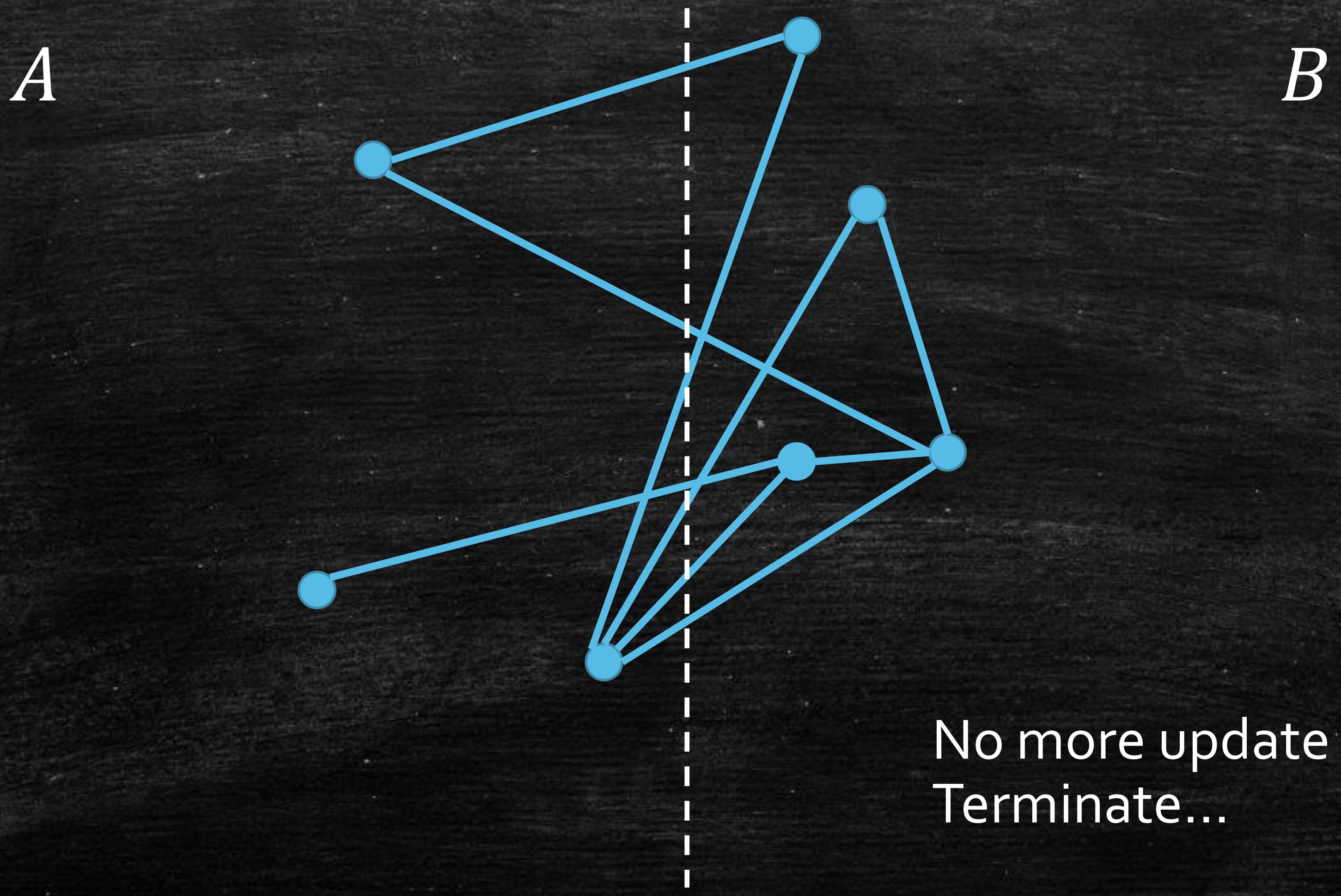
Example



Example



Example



No more update can improve.
Terminate…

Time Complexity?

- Each update searches for at most $O(|V|)$ vertices.
- For each vertex, decide if the update is beneficial takes at most $O(|E|)$ time.
- Total number of updates is at most $|E|$.
  - Each update increases the cut size by at least 1.
- Overall: $O\left(|V||E|^2\right)$ - polynomial time!

Approximation Guarantee?

- Each vertex u: at least $\frac{1}{2}\deg(u)$ incident edges in the cut.
- Thus,
$$c(A, B) \geq \frac{1}{2}\sum_{u \in V}\frac{1}{2}\deg(u) = \frac{1}{2}|E|.$$
- $|E|$ is an obvious upper bound to OPT.
- Therefore, the local search algorithm is a 0.5-approximation.

Can the algorithm do better than 0.5-approximation?

- No…
- Can you give a tight example?

Are there better approximation algorithms?

- Yes!
- Next lecture…

Approximability Spectrum

EASY

- Poly-time Solvable: Shortest-Path, Max-Flow, Min-Cut, Matching, LP
- FPTAS (fully poly-time approximation scheme): Knapsack
  - $(1 \pm \varepsilon)$-approximation for any $\varepsilon > 0$, running time $\mathrm{poly}(n, 1/\varepsilon)$
- PTAS (poly-time approximation scheme): Makespan minimization, Euclidean TSP
  - $(1 \pm \varepsilon)$-approximation for any constant $\varepsilon > 0$, running time may be something like $n^{1/\varepsilon}$
- Constant approximability: Max-3SAT, Vertex Cover, Metric TSP, Max-Cut, Max-k-Coverage, k-Means
- Sub-linear approximability: Set Cover, Dominating Set
- (Almost-)linear inapproximability: Independent Set/Clique, Longest Path on Directed Graphs
- Totally inapproximable: IP, TSP

HARD

This Lecture

- More approximation Algorithms:
  - Max-3SAT
  - Max-k-Coverage
  - Set Cover
  - Max-Cut
- Three techniques:
  - Expectation boosting
  - Greedy
  - Local Search
- For maximization problem, there is a natural "maximum possible value" as upper bound to OPT.

Extra – Naming for **P** and **NP**

- **P**: polynomial-time
- **NP**: non-deterministic polynomial-time

- Deterministic Turing Machine (the normal TM we have seen):
  - Transition $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$
- Non-deterministic Turing Machine
  - Specify two transitions δ_1, δ_2 for each state-alphabet tuple.

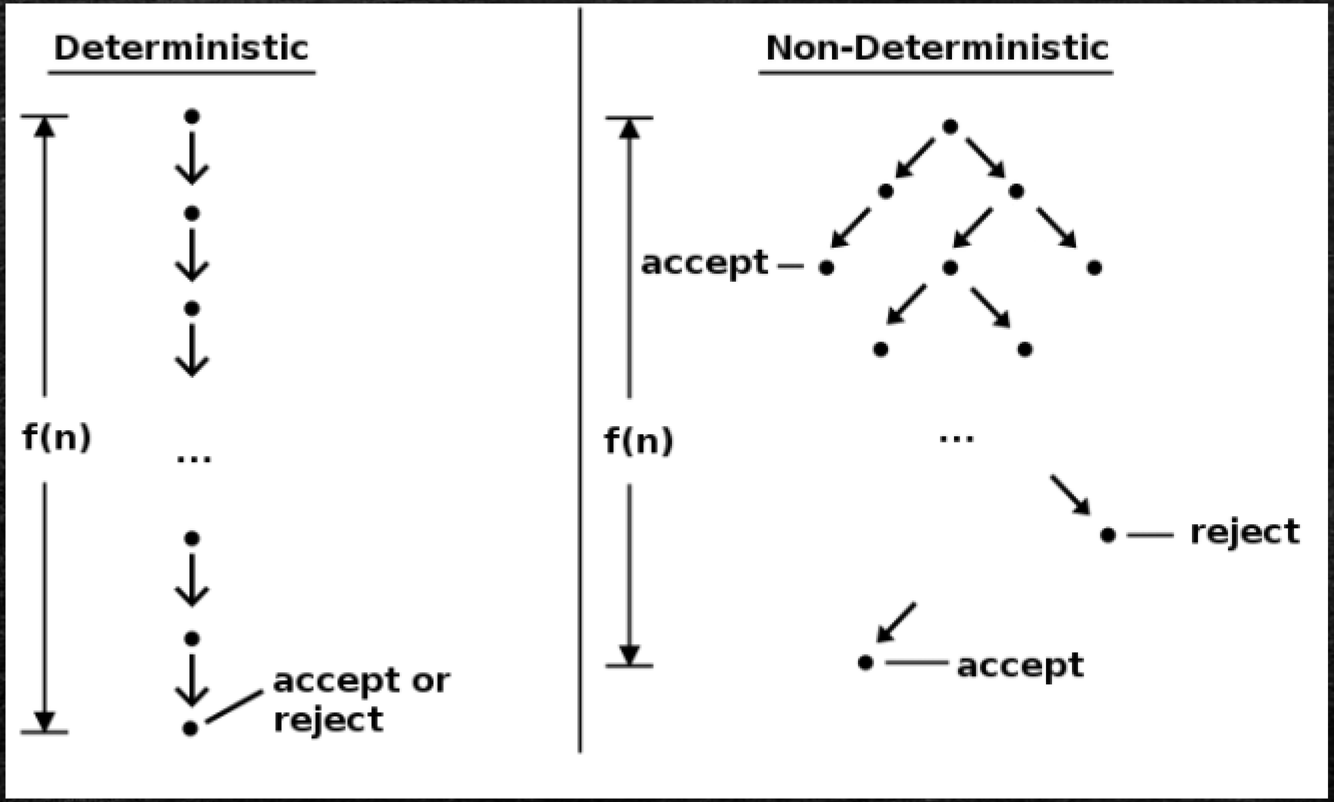


Image from: https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine

Polynomial Time NTM

- A non-deterministic Turing machine runs in polynomial time if, upon receiving input x, all branches reach halting states within $O(|x|^c)$ steps for some constant $c > 0$.

Original Definition for **NP**

- Definition. A decision problem $f: \Sigma^* \to \{0,1\}$ is in **NP** if there is a polynomial time NTM $\mathcal{A}$ such that
  - There is a branch of $\mathcal{A}(x)$ that reaches the accepting state if $f(x) = 1$
  - All branches of $\mathcal{A}(x)$ reach the rejecting state if $f(x) = 0$

- This definition is equivalent to the "certificate definition":
  - Each bit of the certificate corresponds to the "instruction" for which of δ_1, δ_2 we are following.
  - For the yes instance, the certificate "instructs" us to move along the branch that reach the accepting state.
  - For the no instance, no "instruction" can help us reach the accepting state.

SAT ∈ **NP**

- We consider the NTM that enumerates the values of $x_1, \dots, x_n$ in the first n steps.
- Now we have 2^n "terminals" after first n steps.
- For each terminal, verify if ϕ is satisfied; go to the accepting state if it is, and go to the rejecting state if not.