Lecture 12

TTIC 41000: Algorithms for Massive Data

Toyota Technological Institute at Chicago

Spring 2021

Instructor: Sepideh Mahabadi

This Lecture

- Sequence sortedness
- Set Cover in Sublinear Time

Sublinear Time Algorithms

- The input is so huge that even reading all of it is not feasible
- Solve the problem accessing a *small* portion of the input
  - Need to specify the access model: what queries can be asked?
    - Random Access
      - E.g., For an array, given i, return the ith entry of a matrix, i.e., A[i]
      - For a graph, query the adjacency graph: given u,v, return A[u][v], i.e., does there exist an edge between u and v
      - Adjacency List: given u, i, return the ith neighbor of the vertex u (or Null if deg(u)<i)
    - Sample
      - Algorithm receives a random sample from a specific distribution
  - Parameters of interest
    - Number of queries asked
    - Actual runtime (could be sublinear, polynomial, or even exponential)

Example Goals

- Estimate the solution to a problem
  - E.g. what is the average degree in the graph
  - E.g. what is the size of the minimum set cover
- Property Testing: Testing whether the input has a property P, or is far from having the property
  - does the input need to change a lot to have the property
  - total variation distance between a distribution and the closest distribution having the property

Sortedness of a sequence

Problem Definition

- **Input:** a list of n numbers: $a_1, \ldots, a_n$
- **Output:** distinguish if
  - The list is sorted
  - Far from being sorted: at least ϵn elements need to be deleted so that the list becomes sorted.
    - In other words: the length of the longest increasing sequence is $< (1-\epsilon)n$
- **Query model:** given i, what is a_i
- Randomization
  - If sorted, output PASS
  - If far from being sorted, output FAIL w.p. at least $\geq 3/4$

Simple Idea 1

- For a number of iterations
  - sample i and if $a_i > a_{i+1}$, output FAIL
- Otherwise output PASS

- How many iterations?
- Bad example?
  - 1,2,…,n/2, 1,2,…,n/2
  - Needs $\Omega(n)$ queries
  - It is ½ -far from being sorted

Simple Idea 2

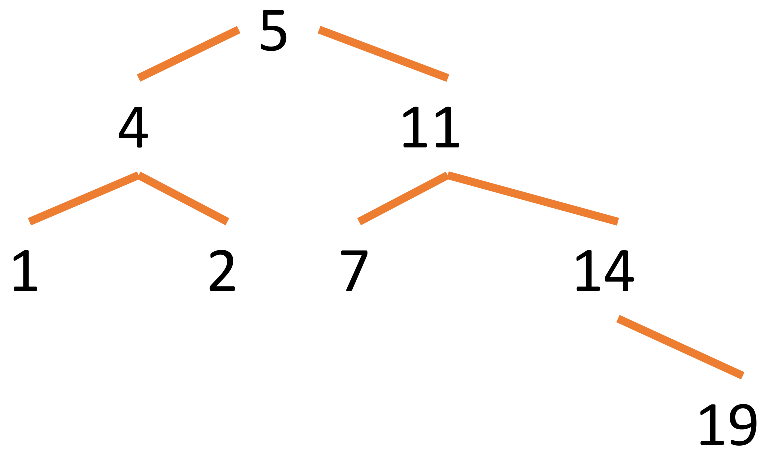
- For a number of iterations
  - sample $i < j$ and if $a_i > a_j$, output FAIL
- Otherwise output PASS
- How many iterations?
- Bad example?
  - 2,1,4,3,6,5,…,n,n-1
  - Must sample two elements from one pair to detect
  - Needs $\Omega(\sqrt{n})$ queries
  - It is ½ -far from being sorted
- Goal: $O(\frac{\log n}{\epsilon})$

Algorithm

- For $O(\frac{1}{\epsilon})$ iterations
  - Sample random a_i
  - Binary search on a_i
  - If Binary Search finds any inconsistencies output FAIL
- Output PASS
- Runtime: $O\left(\frac{\log n}{\epsilon}\right)$
- Correctness:
  - If the list is sorted, it outputs pass
  - Need to show: if it passes the test there are $(1-\epsilon)n$ elements that are sorted

Analysis

- **Good element:** binary search is successful on it
- Algorithm guarantees that w.h.p the number of good elements is $\geq (1-\epsilon)n$
- Good elements form increasing sub-sequence
  - If $i < j$ are both good, then need to show $a_i < a_j$
  - Let k be their common ancestor
  - Search for i went left, and search for j went right of k, so $a_i \leq a_k \leq a_j$
- Example:1,4,2,5,7,11,14,19
- BST is based on the indices
- Good elements: 1,4,5,7,11,14,19
- Bad elements: 2



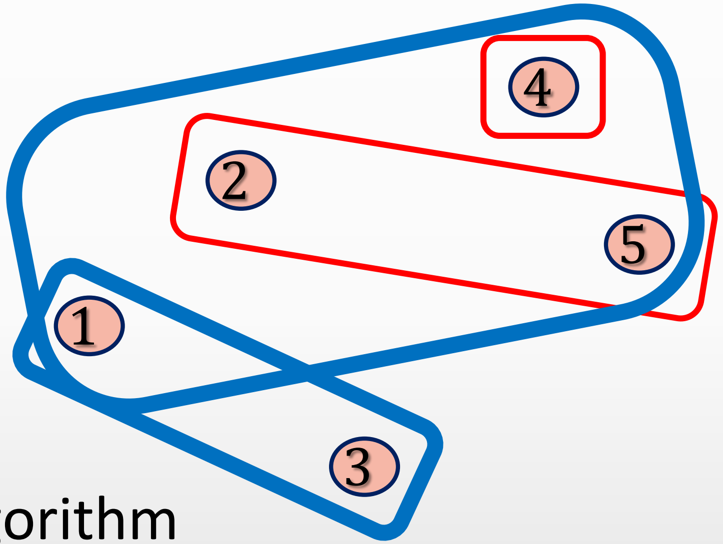
Set Cover

Set Cover Problem

Input: Collection $\mathcal{F}$ of sets $S_1, \dots, S_m$, each a subset of $\mathcal{U} = \{1, \dots, n\}$

Output: a subset $\mathcal{C}$ of $\mathcal{F}$ such that:

- $\mathcal{C}$ covers $\mathcal{U}$
- $|\mathcal{C}|$ is minimized



Complexity:

- NP-hard
- Greedy $(\ln n)$-approximation algorithm
- Can't do better unless **P=NP** [LY91][RS97][Fei98][AMS06][DS14]

*"Is it possible to solve minimum set cover in **sub-linear time**?"*

Sub-linear Time Set Cover

Data Access Model [NO'08,YYI'12]

- No assumption on the order
- Incidence list in (sub-linear) algorithms for graphs
- Sublinear in $\boldsymbol{mn}$

EltOf$(\boldsymbol{S}, \boldsymbol{i})$: ith element in $\boldsymbol{S}$
SetOf$(\boldsymbol{e}, \boldsymbol{j})$: jth set containing $\boldsymbol{e}$

$\boldsymbol{n}$ = **number of *elements*** $\boldsymbol{m}$ = **number of *sets*** $\boldsymbol{k}$ = **size of the optimal solution**

Part one: upper bound

Theorem: There exists an algorithm that with high probability finds an $O(\rho\alpha)$-approximate cover which uses $\tilde{O}(\boldsymbol{mn^{1/\alpha}} + \boldsymbol{nk})$ number of queries.

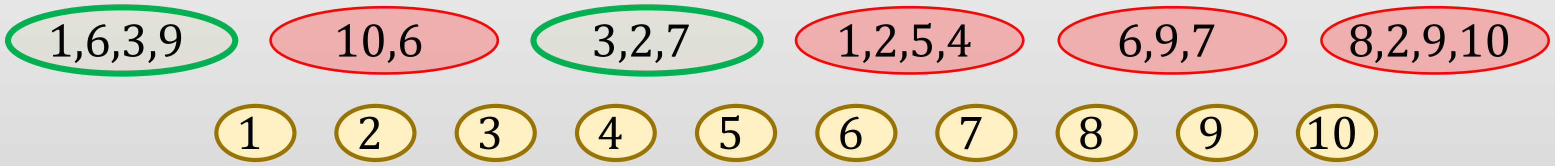
Same technique (very similar algorithm) as the streaming model

Component I: set sampling

Set Sampling: After picking ℓ sets uniformly at random, all elements with degree at least $\frac{m \log n}{\ell}$ are covered w.h.p.

- We only need to worry about low degree elements.

$\ell = 2$

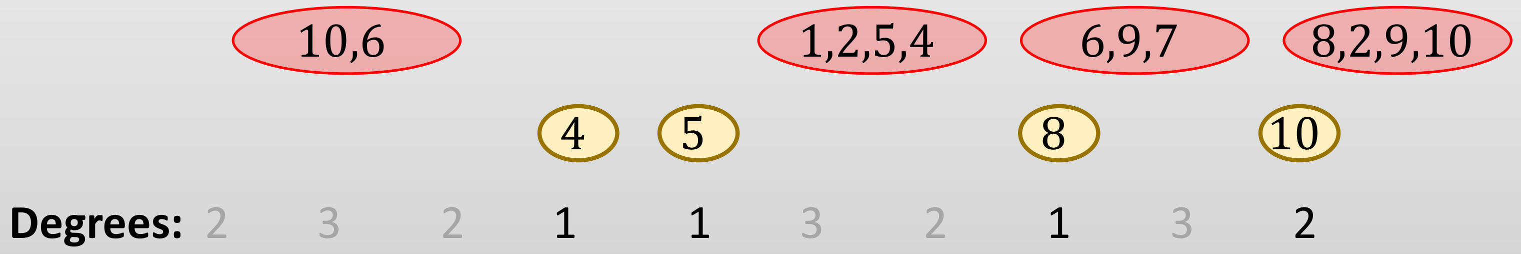


Degrees: 2 3 2 1 1 3 2 1 3 2

Component I: set sampling

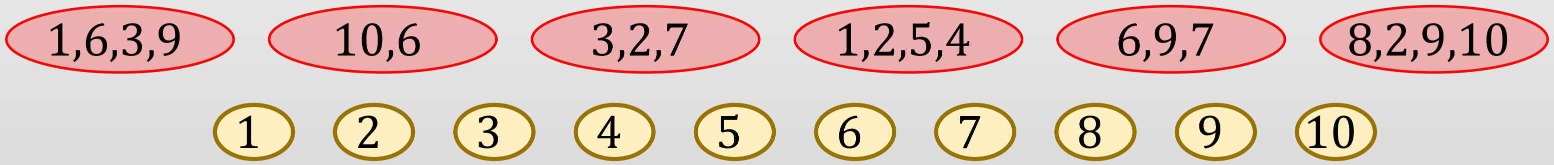
Set Sampling: After picking ℓ sets uniformly at random, all elements with degree at least $\frac{\mathrm{m}\log n}{\ell}$ are covered w.h.p.

- We only need to worry about low degree elements.



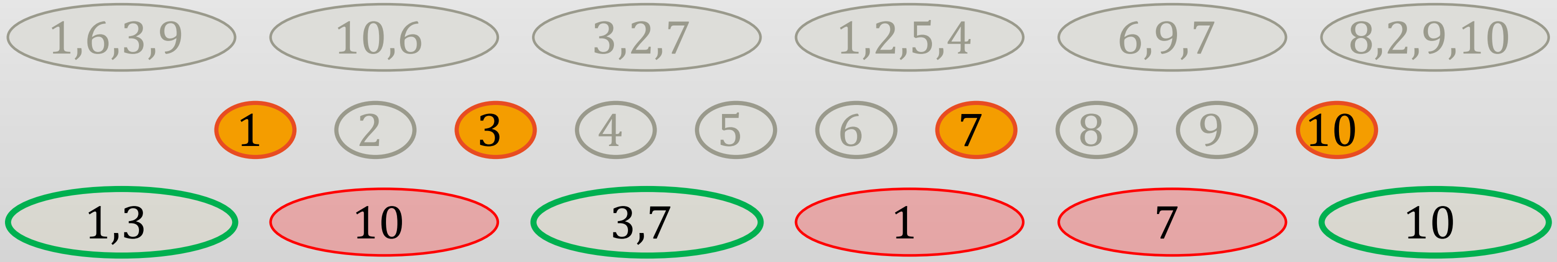
Component II: element sampling

Element Sampling: Sample a few elements and solve the set cover for the sampled elements.



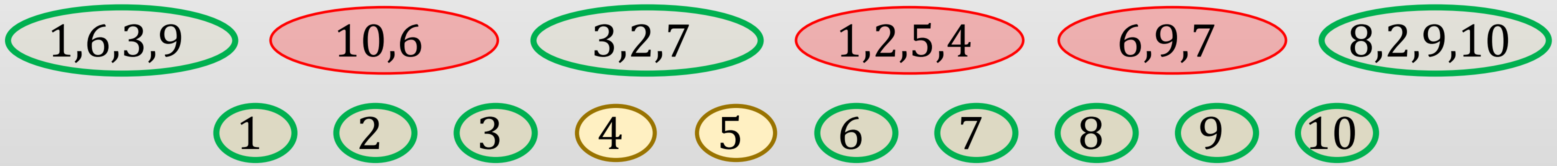
Component II: element sampling

Element Sampling: Sample a few elements and solve the set cover for the sampled elements.



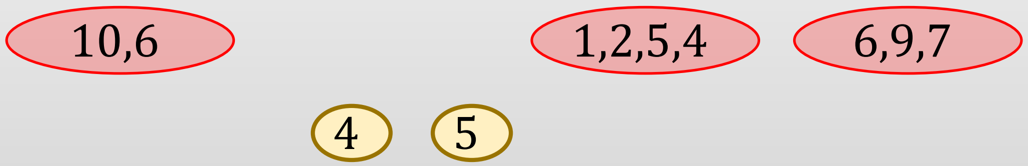
Component II: element sampling

Element Sampling: Sample a few elements and solve the set cover for the sampled elements.



Component II: element sampling

Element Sampling: Sampling $\Theta(\frac{\rho k \log m}{\delta})$ elements uniformly at random and finding a ρ-approximate cover for the sampled elements, will cover $(1-\delta)$ fraction of the original elements w.h.p.



Algorithm

$\log n$ different guesses
$\ell \in \{1,2,4,\ldots,n\}$

Make a guess ℓ of the value of the optimal solution k

Algorithm

Make a guess ℓ of the value of the optimal solution k

- ❑ Preprocessing: perform set sampling
- ❑ Sol ← sampled sets

$\log n$ different guesses $\ell \in \{1,2,4,\ldots,n\}$

sample ℓ sets,
number of queries: $n\ell$

Set Sampling: After picking ℓ sets uniformly at random, all elements with degree at least $\frac{m \log n}{\ell}$ are covered w.h.p.

Algorithm

Make a guess ℓ of the value of the optimal solution k

- ❑ Preprocessing: perform set sampling
- ❑ Sol ← sampled sets
- ❑ For α iterations
  - Use element sampling to cover $(1 - \frac{1}{n^{1/\alpha}})$ fraction of the uncovered elements.
  - Add the sets to Sol

$\log n$ different guesses $\ell \in \{1,2,4, \ldots, n\}$

sample ℓ sets,
number of queries: $n\ell$

sample $(\rho \ell n^{1/\alpha} \log m)$ elements,
number of queries:
$O\left(\rho \ell n^{1/\alpha} \log m \frac{m \log n}{\ell}\right)$
$= O(\rho m n^{1/\alpha} \log m \log n)$

$\boldsymbol{\delta = 1/n^{1/\alpha}}$

Element Sampling: Sampling $\Theta(\frac{\rho k \log m}{\delta})$ elements uniformly at random and finding a ρ-approximate cover for the sampled elements, will cover $(1 - \delta)$ fraction of the original elements w.h.p.

Algorithm

Make a guess ℓ of the value of the optimal solution k

- ❑ Preprocessing: perform set sampling
- ❑ Sol ← sampled sets
- ❑ For α iterations
  - Use element sampling to cover $(1-\frac{1}{n^{1/\alpha}})$-fraction of the uncovered elements.
  - Add the sets to Sol
  - Update uncovered elements.

$\log n$ different guesses $\ell \in \{1,2,4,\ldots,n\}$

sample ℓ sets,
number of queries: $n\ell$

sample $(\rho \ell n^{1/\alpha} \log m)$ elements,
number of queries:
$O\left(\rho \ell n^{1/\alpha} \log m \frac{m \log n}{\ell}\right)$
$= O(\rho m n^{1/\alpha} \log m \log n)$

number of queries: $\rho n \ell$

Algorithm

Make a guess ℓ of the value of the optimal solution k

- ❑ Preprocessing: perform set sampling
- ❑ Sol $\leftarrow$ sampled sets
- ❑ For α iterations
  - Use element sampling to cover $(1 - \frac{1}{n^{1/\alpha}})$-fraction of the uncovered elements.
  - Add the sets to Sol
  - Update uncovered elements.
- ❑ If all elements are covered, report Sol

$\log n$ different guesses $\ell \in \{1,2,4, \dots, n\}$

sample ℓ sets,
number of queries: $n\ell$

sample $(\rho\ell n^{1/\alpha} \log m)$ elements,
number of queries:
$O\left(\rho\ell n^{1/\alpha} \log m \frac{m \log n}{\ell}\right)$
$= O(\rho m n^{1/\alpha} \log m \log n)$

number of queries: $\rho n\ell$

Algorithm

Make a guess ℓ of the value of the optimal solution k

- ❑ Preprocessing: perform set sampling
- ❑ Sol ← sampled sets
- ❑ For α iterations
  - Use element sampling to cover $(1 - \frac{1}{n^{1/\alpha}})$-fraction of the uncovered elements.
  - Add the sets to Sol
  - Update uncovered elements.
- ❑ If all elements are covered, report Sol

$\log n$ different guesses $\ell \in \{1,2,4,\dots,n\}$

sample ℓ sets,
number of queries: $n\ell$

sample $(\rho\ell n^{1/\alpha}\log m)$ elements,
number of queries:
$O\left(\rho\ell n^{1/\alpha}\log m \frac{m\log n}{\ell}\right)$
$=O(\rho m n^{1/\alpha}\log m\log n)$

number of queries: $\rho n\ell$

Theorem: start querying with the smaller guesses of ℓ

Algorithm

Make a guess ℓ of the value of the optimal solution k

- ❑ Preprocessing: perform set sampling
- ❑ Sol ← sampled sets
- ❑ For α iterations
  - Use element sampling to cover $(1 - \frac{1}{n^{1/\alpha}})$-fraction of the uncovered elements.
  - Add the sets to Sol
  - Update uncovered elements.
- ❑ If all elements are covered, report Sol

$\log n$ different guesses $\ell \in \{1,2,4, \ldots, n\}$

sample ℓ sets,
number of queries: $n\ell$

sample $(\rho\ell n^{1/\alpha} \log m)$ elements,
number of queries:
$O\left(\rho\ell n^{1/\alpha} \log m \frac{m \log n}{\ell}\right)$
$= O(\rho m n^{1/\alpha} \log m \log n)$

number of queries: $\rho n\ell$

Theorem: There exists an algorithm that with high probability finds an $O(\rho\alpha)$-approximate cover which uses $\tilde{O}(\boldsymbol{mn^{1/\alpha} + nk})$ number of queries.

Part two: lower bound

Theorem: Any randomized algorithm that with probability at least 2/3 distinguishes whether the minimum Set Cover size is 2 or at least 3 requires $\widetilde{\boldsymbol{\Omega}}(\boldsymbol{mn})$ number of queries.

High Level Approach

1. Construct a median instance I^*
   - Minimum Set Cover Size is 3
2. Randomized Procedure on I^* to get a modified instance I
   - Minimum Set Cover Size is 2
   - I^* and I only differ in a few positions
   - The differences are distributed almost uniformly at random
3. Any algorithm that can detect these two cases requires to query at least $\widetilde{\Omega}(mn)$ queries.

The Median Instance

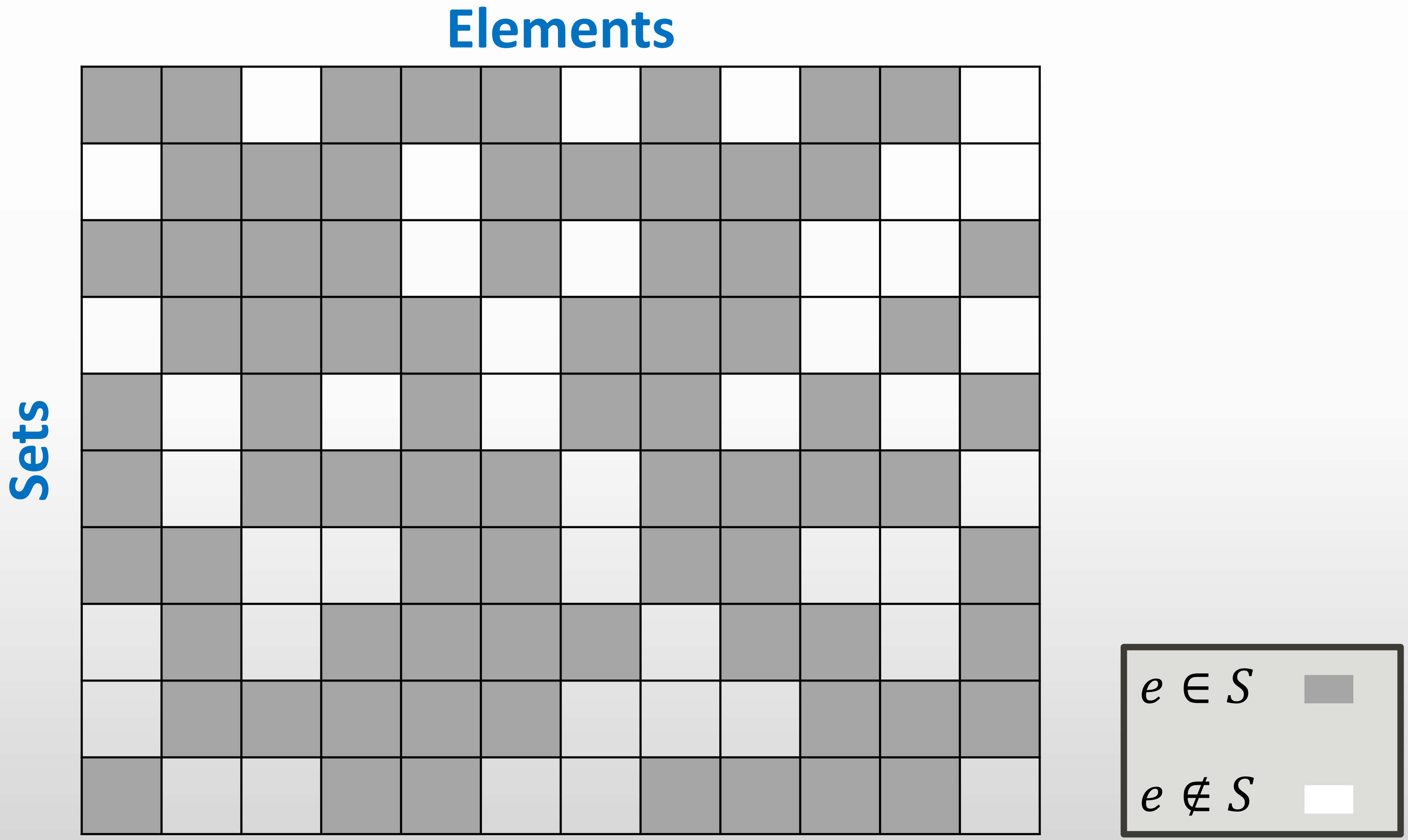
Construction: is randomized. For every S, e the set S contains e with probability $1 - p_0$ where $p_0 = \sqrt{\frac{9 \log m}{n}}$

Properties: by Chernoff, most of such instances have the following properties:

1. No 2 sets cover all the elements
2. For any two sets the number of uncovered elements is $O(\log m)$
3. The intersection is at least $\Omega(n)$
4. For each element, the number of sets not covering it is at most $6m\sqrt{\frac{\log m}{n}}$
5. For any pair of elements the number of sets containing only the first element is at least $\frac{m\sqrt{9 \log m}}{4\sqrt{n}}$
6. For any three sets, the number of elements in the first two but not in the third one is at least $6\sqrt{n \log m}$

Take one such instance I^* with the above properties

The Median Instance



Generating a Modified Instance

Pick two random sets S_1 and S_2 and turn them into a set cover.
How?

$$\boldsymbol{U = \{e_1, e_2, e_3, e_4\}}$$

$\boldsymbol{S_1} = \{\boldsymbol{e_2}, \boldsymbol{e_3}\}$
$\boldsymbol{S_2} = \{\boldsymbol{e_2}, \boldsymbol{e_4}\}$

Generating a Modified Instance

Pick two random sets S_1 and S_2 and turn them into a set cover.
How?

- For each uncovered element $e_1 \in U \setminus (S_1 \cup S_2)$,
  - Add e_1 to S_2

$$\boldsymbol{U = \{e_1, e_2, e_3, e_4\}}$$

$$\boldsymbol{S_1 = \{e_2, e_3\}}$$
$$\boldsymbol{S_2 = \{e_2, e_4\} \leftarrow e_1}$$

Generating a Modified Instance

Pick two random sets S_1 and S_2 and turn them into a set cover.
How?

- For each uncovered element $e_1 \in U \setminus (S_1 \cup S_2)$,
  - Add e_1 to S_2
  - Remove an element $e_2 \in S_2 \cap S_1$ from S_2

$$\boldsymbol{U = \{e_1, e_2, e_3, e_4\}}$$

$$\boldsymbol{S_1 = \{e_2, e_3\}}$$

$$\boldsymbol{S_2 = \{e_2, e_4\}} \leftarrow \boldsymbol{e_1}$$

$$\rightarrow \boldsymbol{e_2}$$

Generating a Modified Instance

Pick two random sets S_1 and S_2 and turn them into a set cover.
How?

- For each uncovered element $e_1 \in U \setminus (S_1 \cup S_2)$,
  - Add e_1 to S_2
  - Remove an element $e_2 \in S_2 \cap S_1$ from S_2
  - Pick a random set S_3 that contains e_1 but not e_2

$$\boldsymbol{U = \{e_1, e_2, e_3, e_4\}}$$

$$\boldsymbol{S_1 = \{e_2, e_3\}}$$
$$\boldsymbol{S_2 = \{e_2, e_4\}}$$
$$\boldsymbol{S_3 = \{e_4, e_1\}}$$

Generating a Modified Instance

Pick two random sets S_1 and S_2 and turn them into a set cover.
How?

- For each uncovered element $e_1 \in U \setminus (S_1 \cup S_2)$,
  - Add e_1 to S_2
  - Remove an element $e_2 \in S_2 \cap S_1$ from S_2
  - Pick a random set S_3 that contains e_1 but not e_2
  - S_2 and S_3 swap e_1 and e_2

$$\boldsymbol{U = \{e_1, e_2, e_3, e_4\}}$$

$$\boldsymbol{S_1 = \{e_2, e_3\}}$$
$$\boldsymbol{S_2 = \{e_2, e_4\}}$$
$$\boldsymbol{S_3 = \{e_4, e_1\}}$$

Modified instance

$$\boldsymbol{S_1 = \{e_2, e_3\}}$$
$$\boldsymbol{S_2 = \{e_1, e_4\}}$$
$$\boldsymbol{S_3 = \{e_4, e_2\}}$$

Only four positions changes in the query access model.

Generating a Modified Instance

Pick two random sets S_1 and S_2 and turn them into a set cover.
How?

- For each uncovered element $e_1 \in U \setminus (S_1 \cup S_2)$,
  - Add e_1 to S_2
  - Remove an element $e_2 \in S_2 \cap S_1$ from S_2
  - Pick a random set S_3 that contains e_1 but not e_2
  - S_2 and S_3 swap e_1 and e_2

$$\boldsymbol{U = \{e_1, e_2, e_3, e_4\}}$$

Two in **ElemOf** oracles
+
Two in **SetOf** oracles

Swap

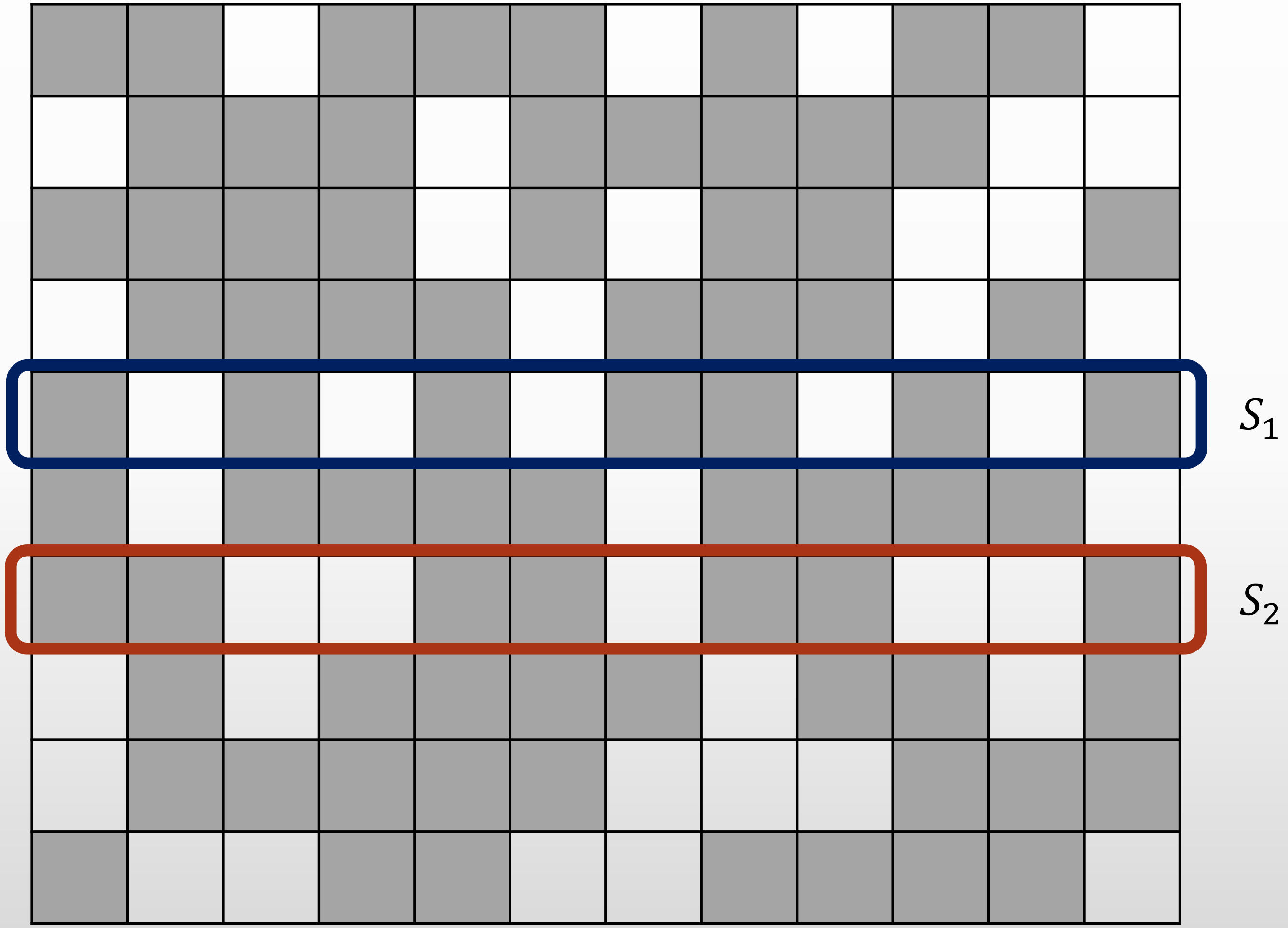
Modified instance

$$\boldsymbol{S_1 = \{e_2, e_3\}}$$
$$\boldsymbol{S_2 = \{e_1, e_4\}}$$
$$\boldsymbol{S_3 = \{e_4, e_2\}}$$

Only four positions changes in the query access model.

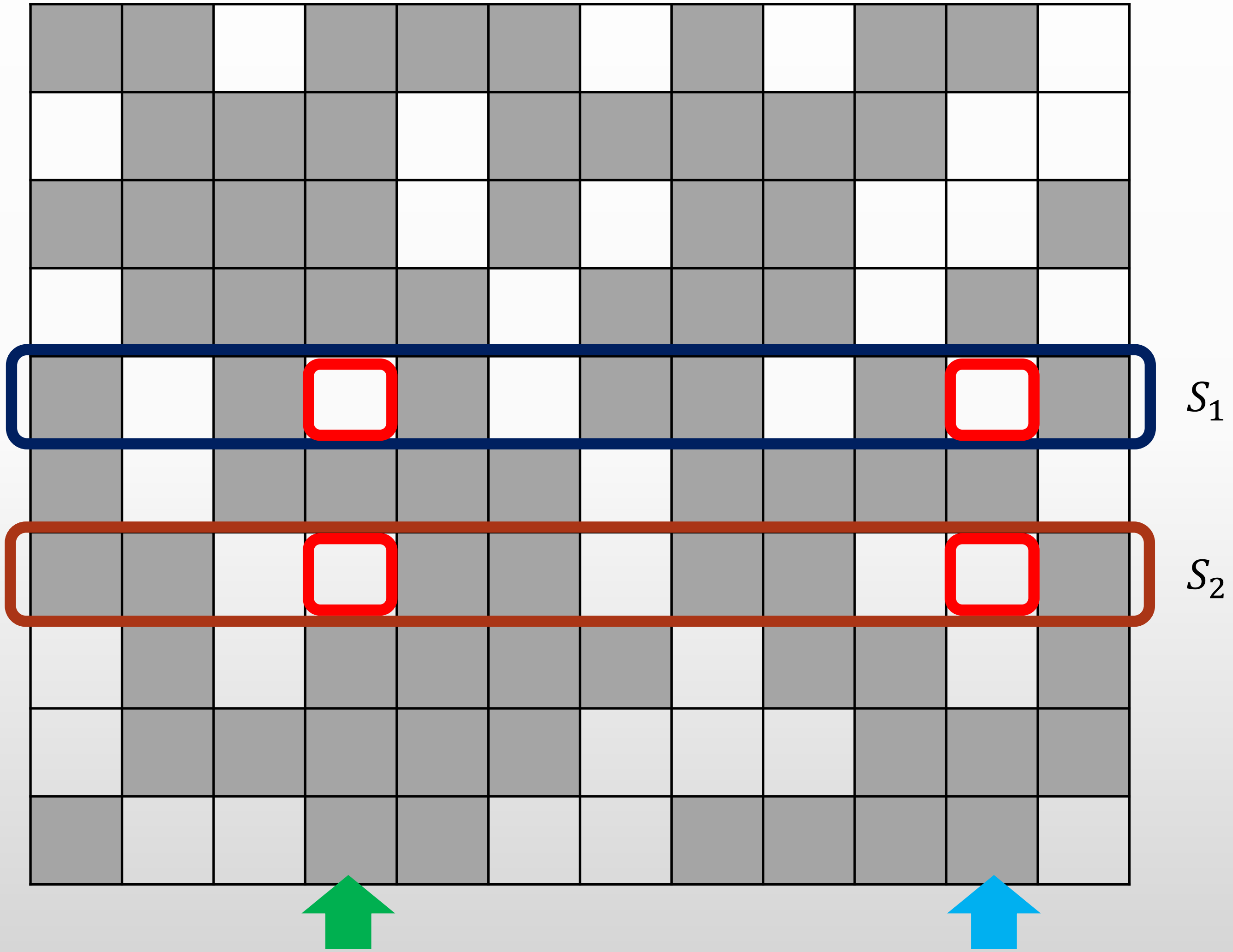
The Randomized Procedure

- Median Instance
- **Pick two Sets Uniformly at Random**



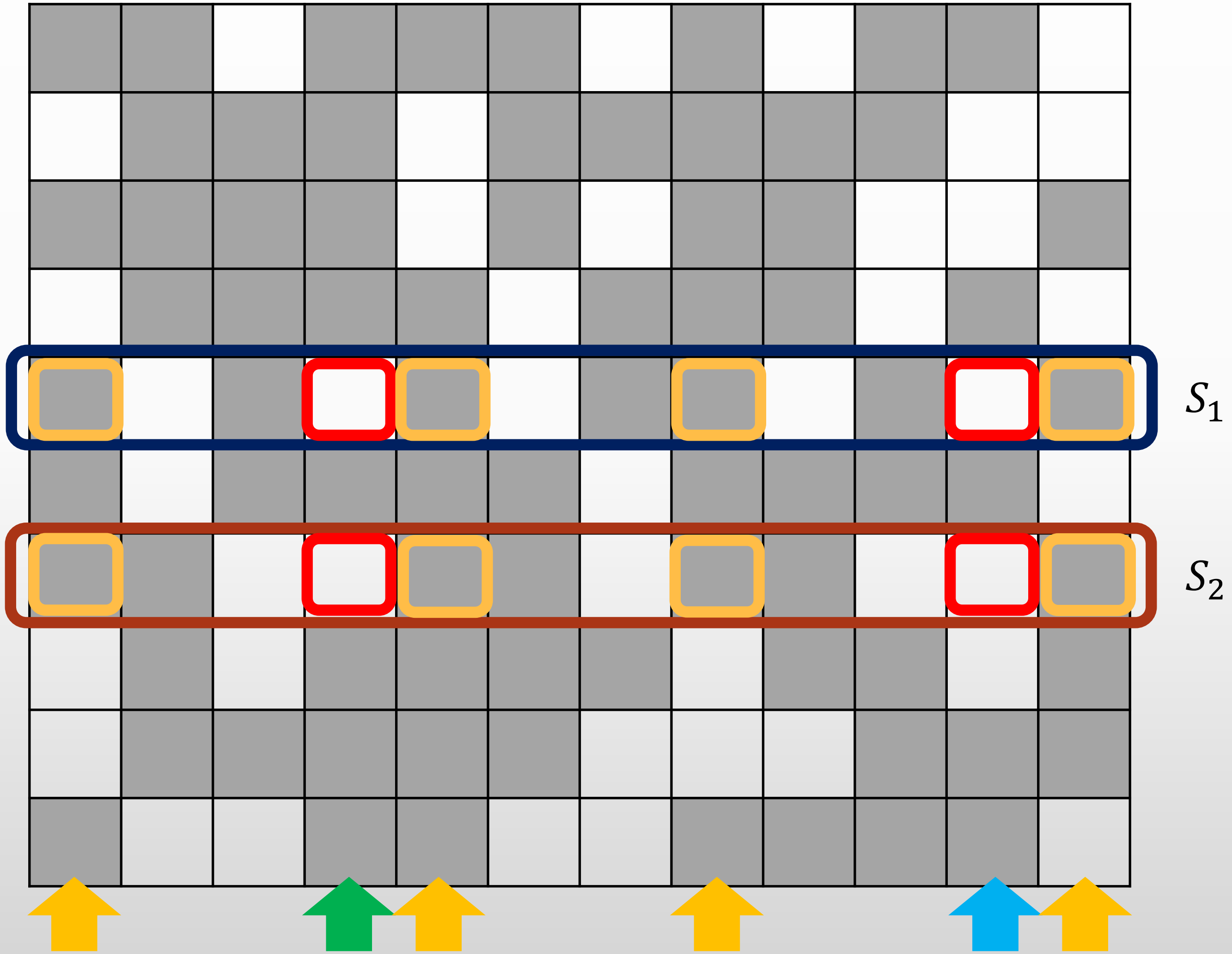
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- **Find the elements that are not covered**



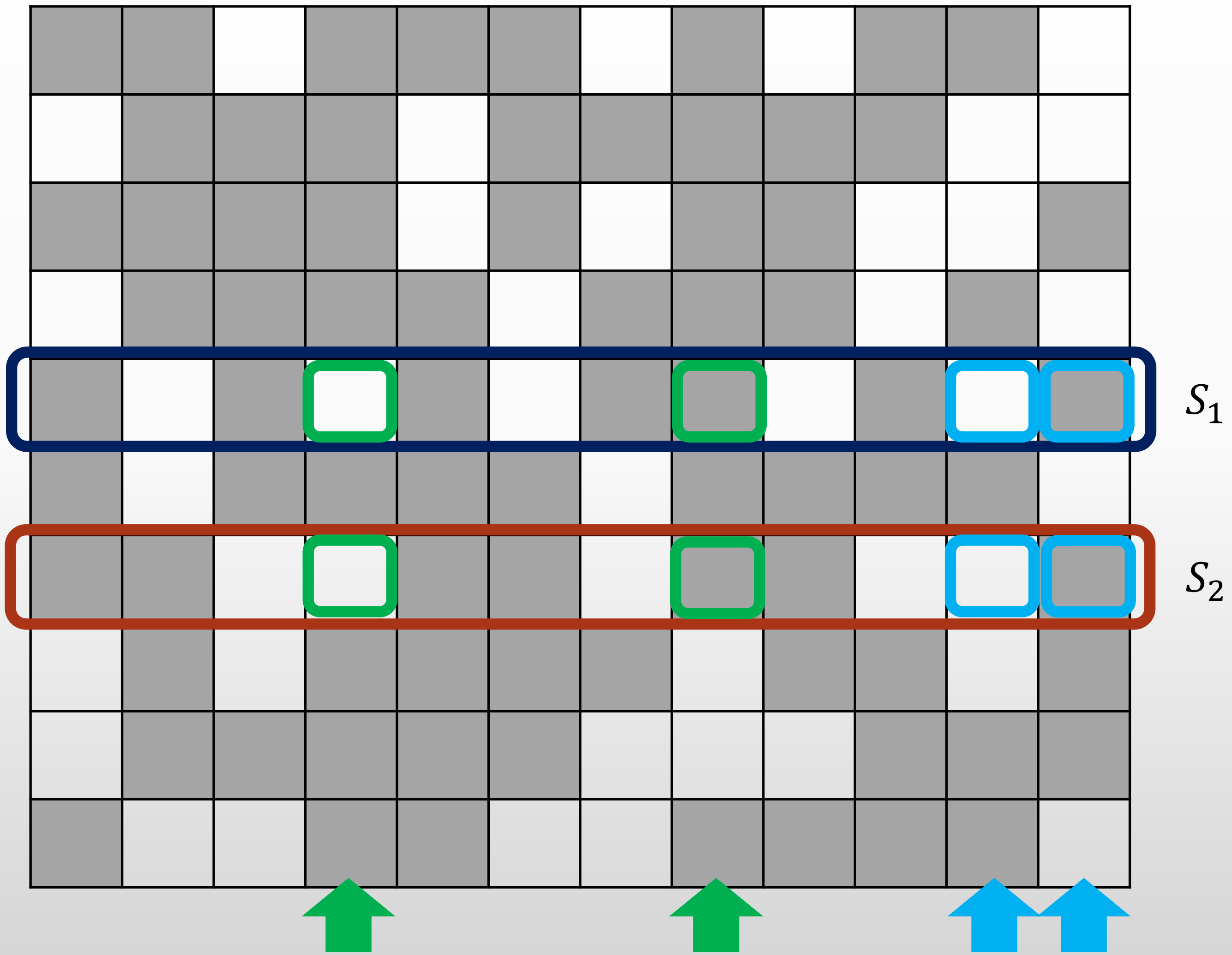
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- **Also find the elements that are covered by both**



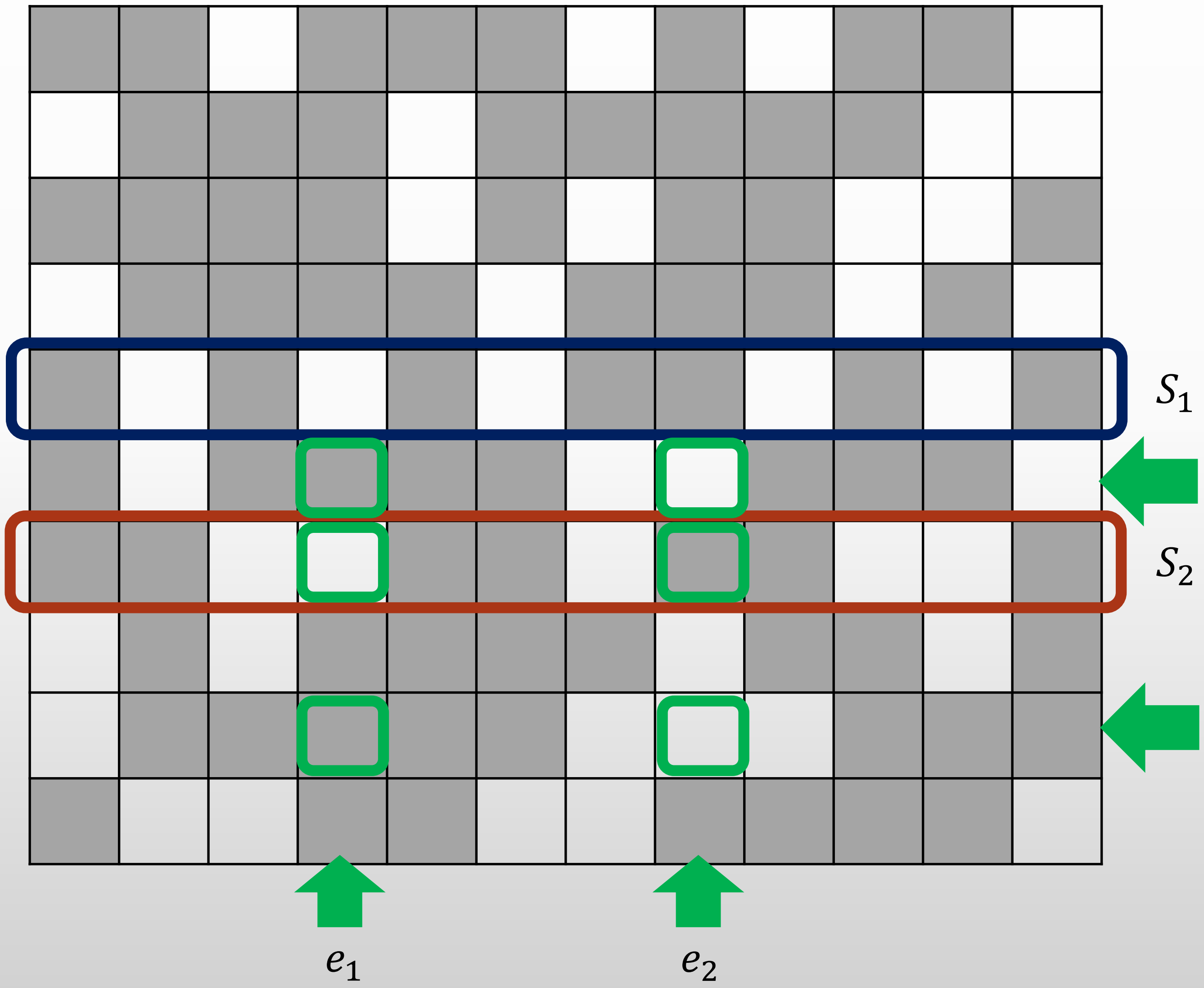
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- **Assign one element in the intersection to each uncovered element**



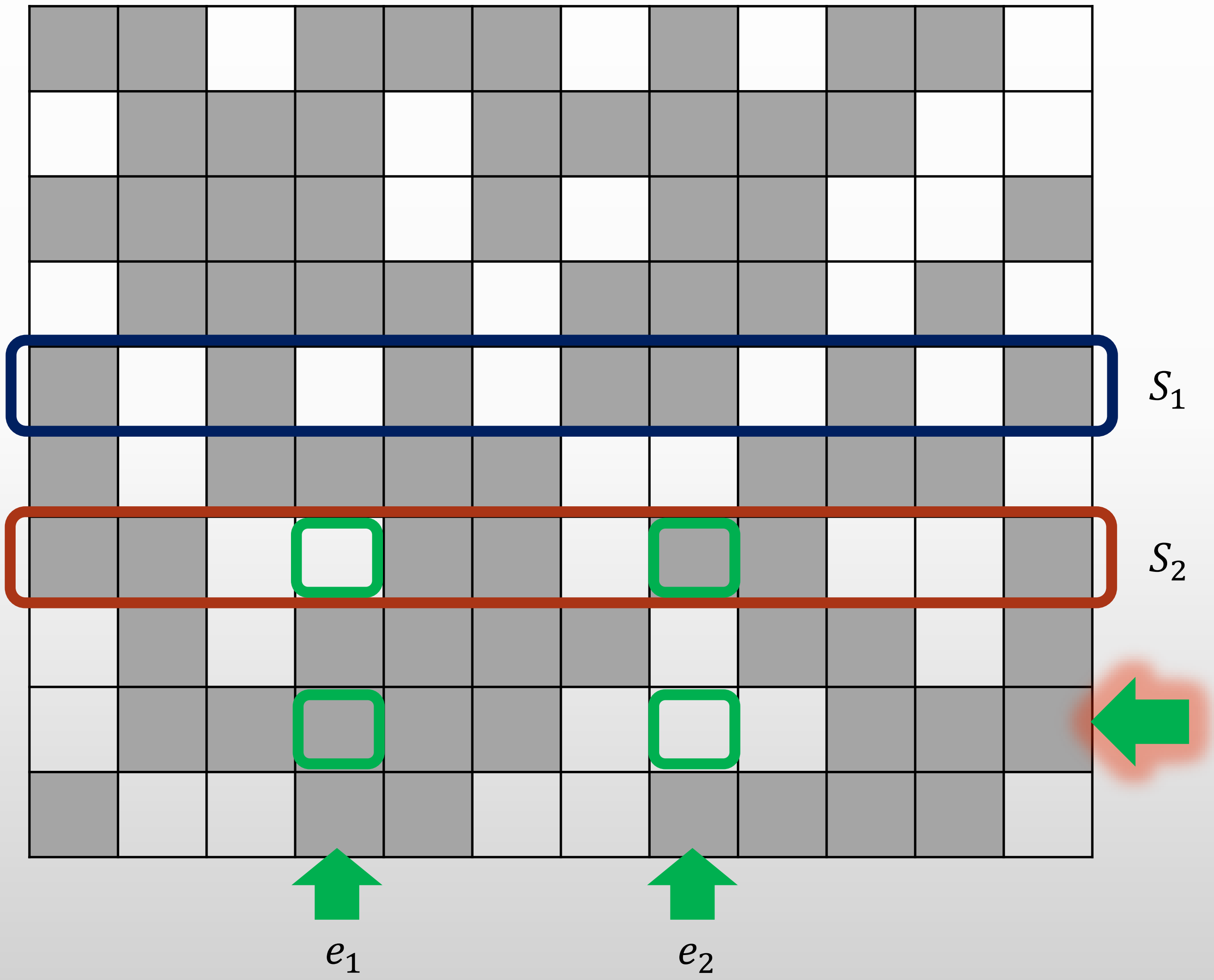
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- **Assign one element in the intersection to each uncovered element**



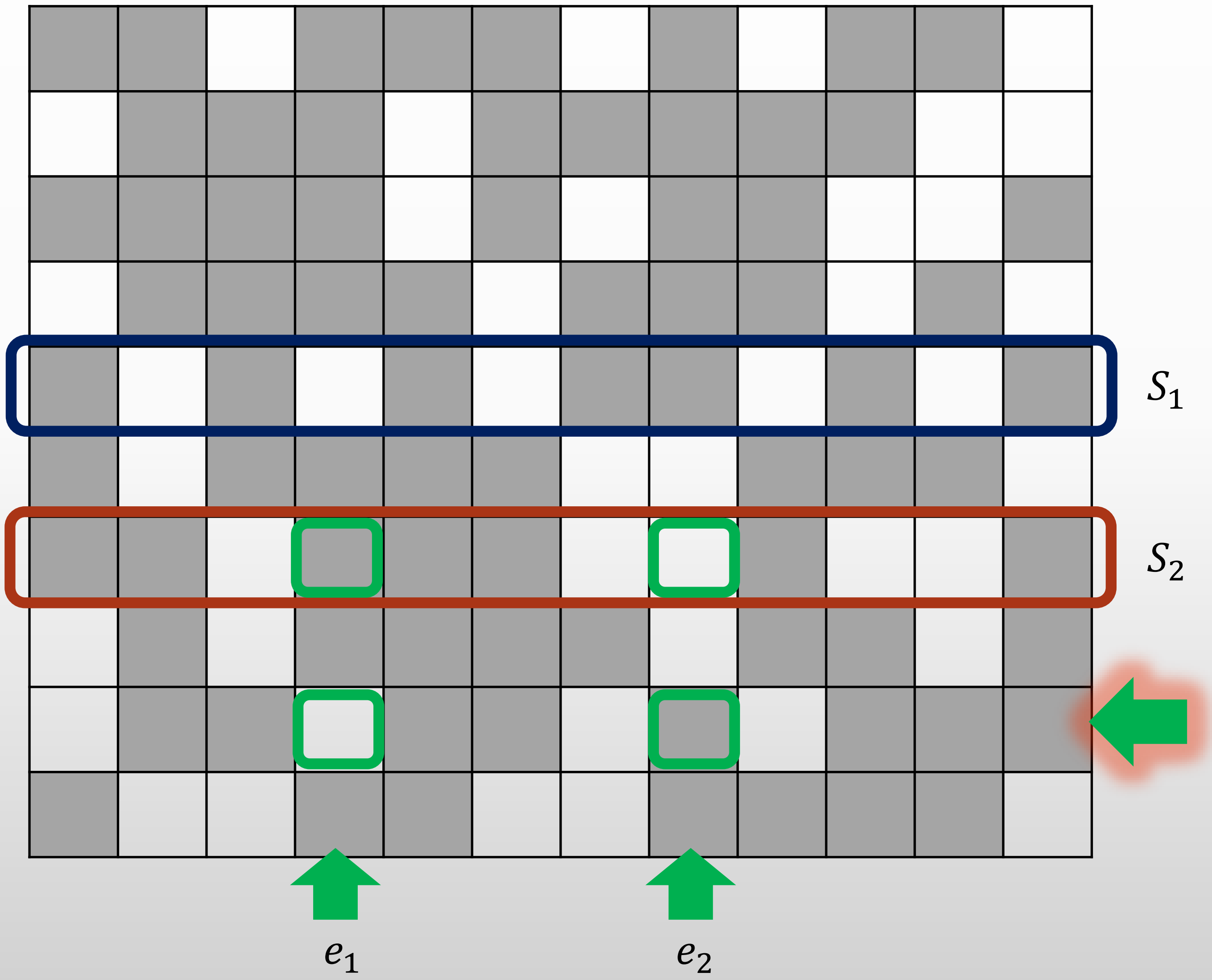
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - **Find a candidate set**



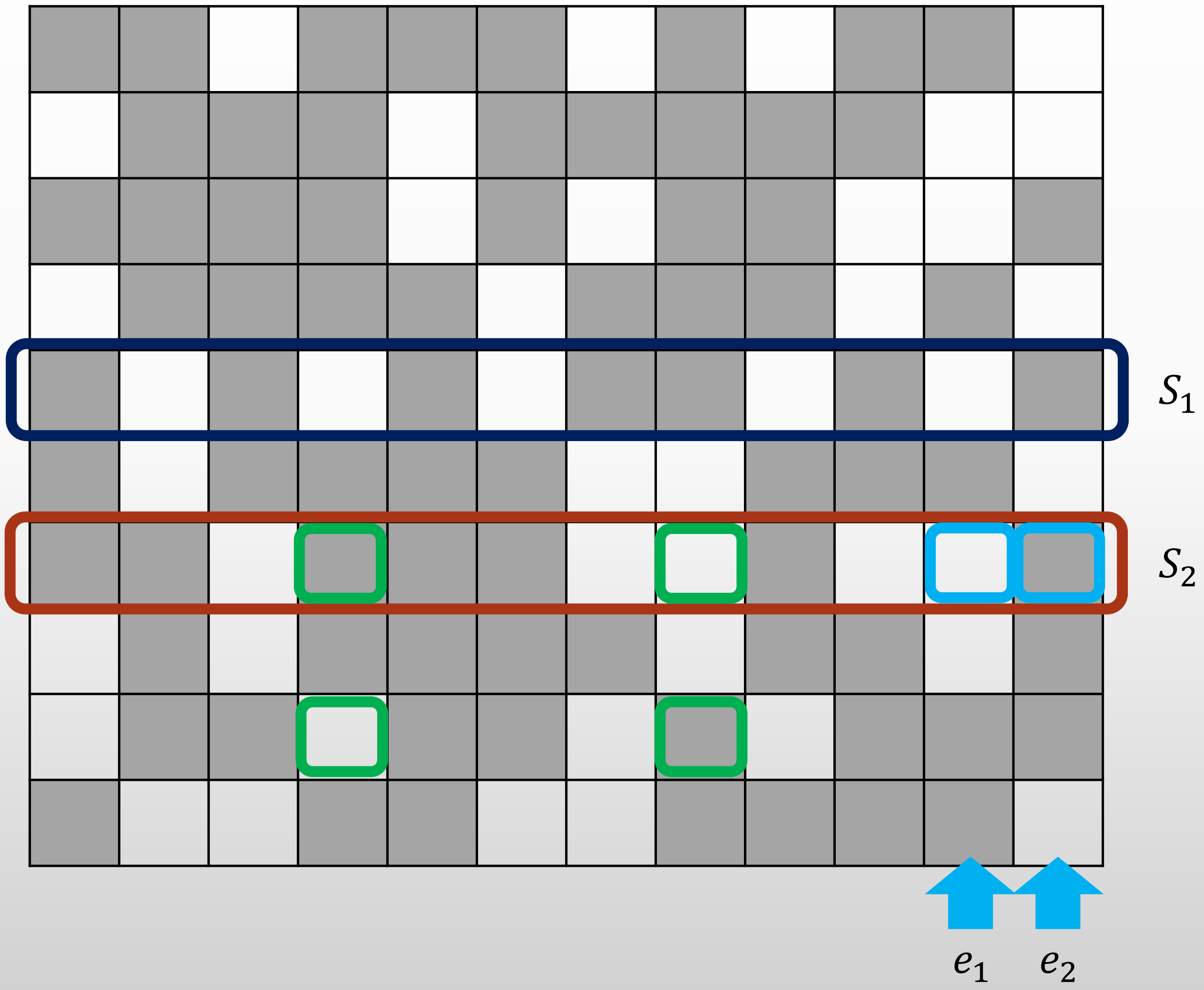
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - Find a candidate set
  - **swap**



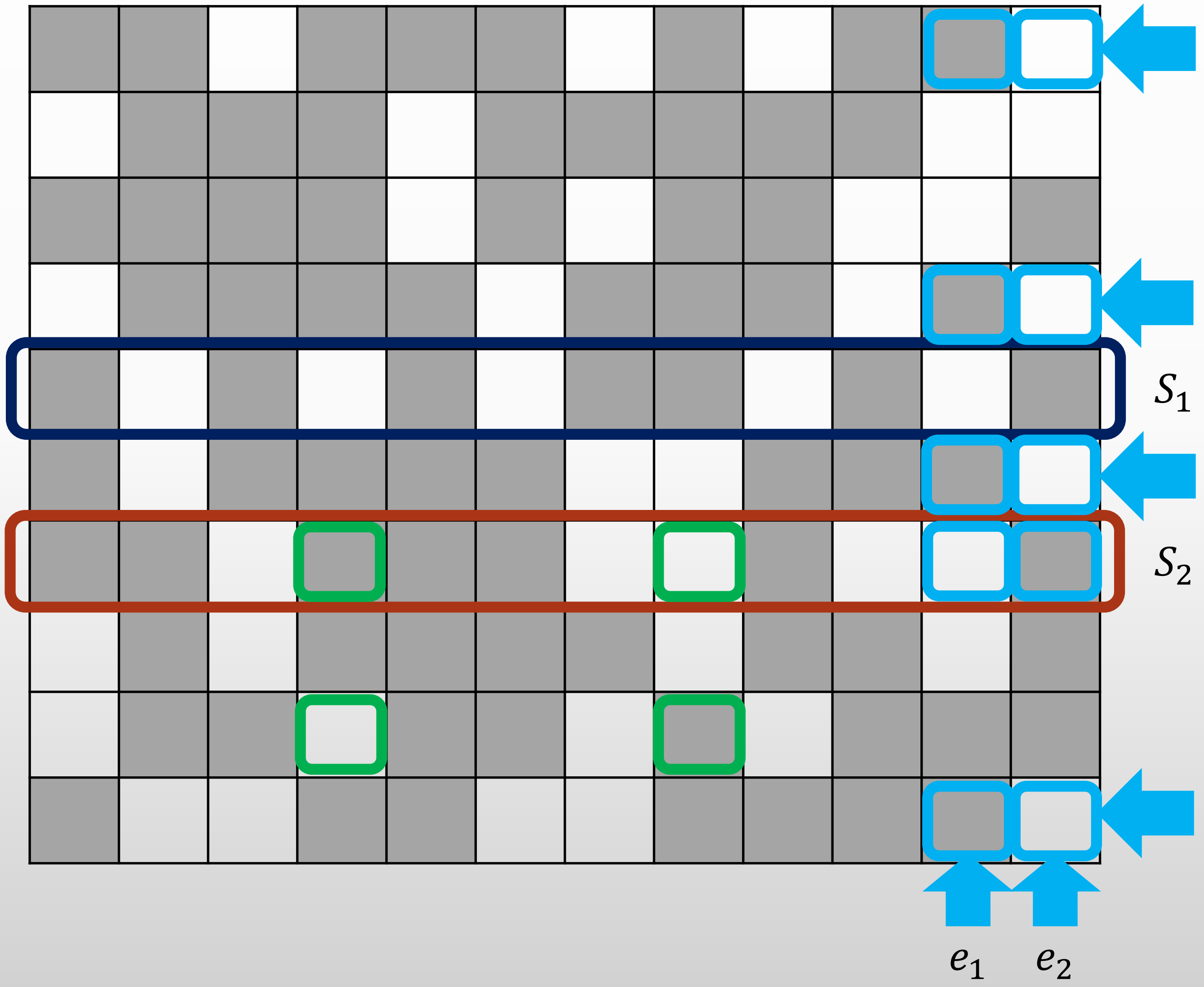
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - Find a candidate set
  - **swap**



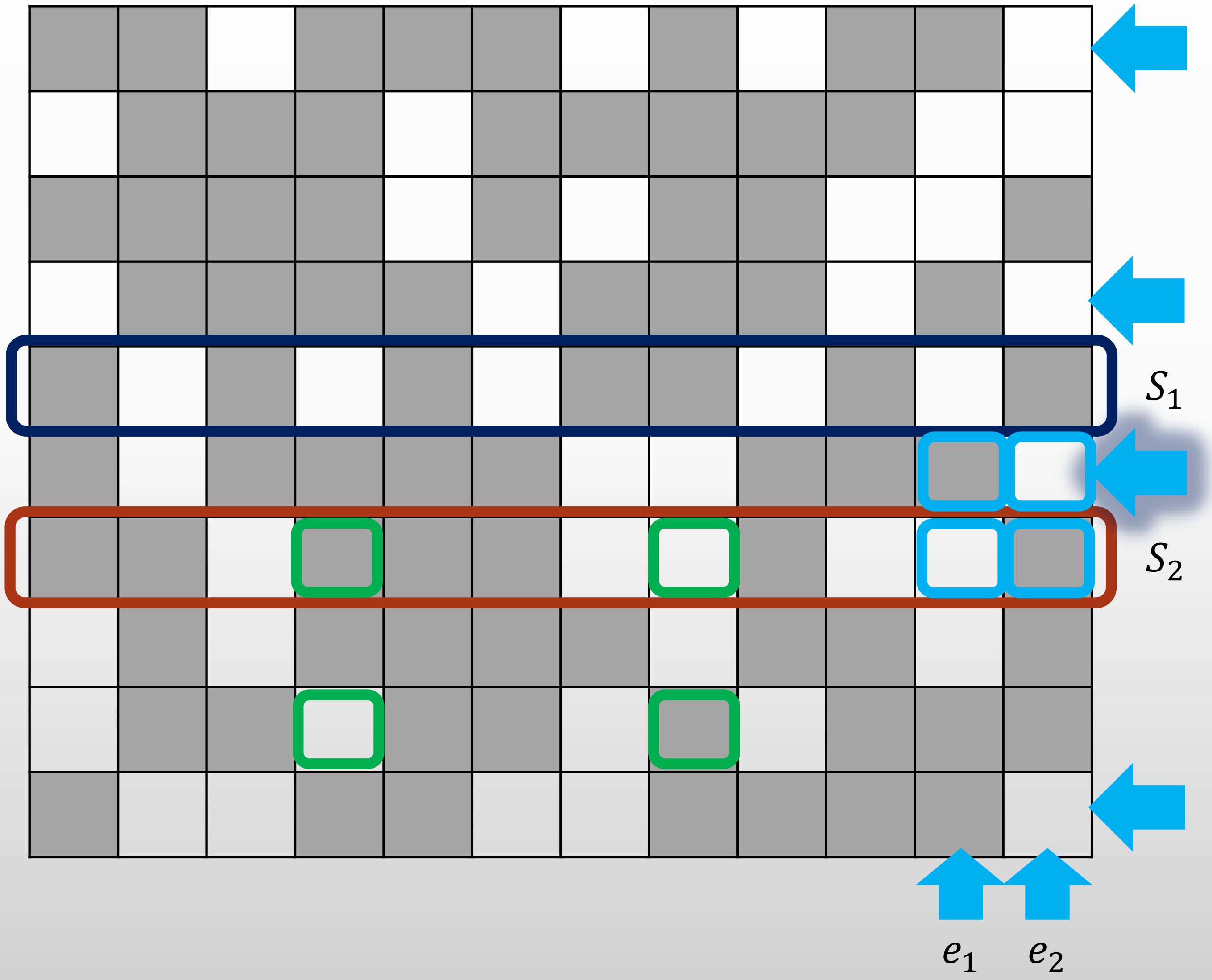
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - **Find a candidate set**
  - swap



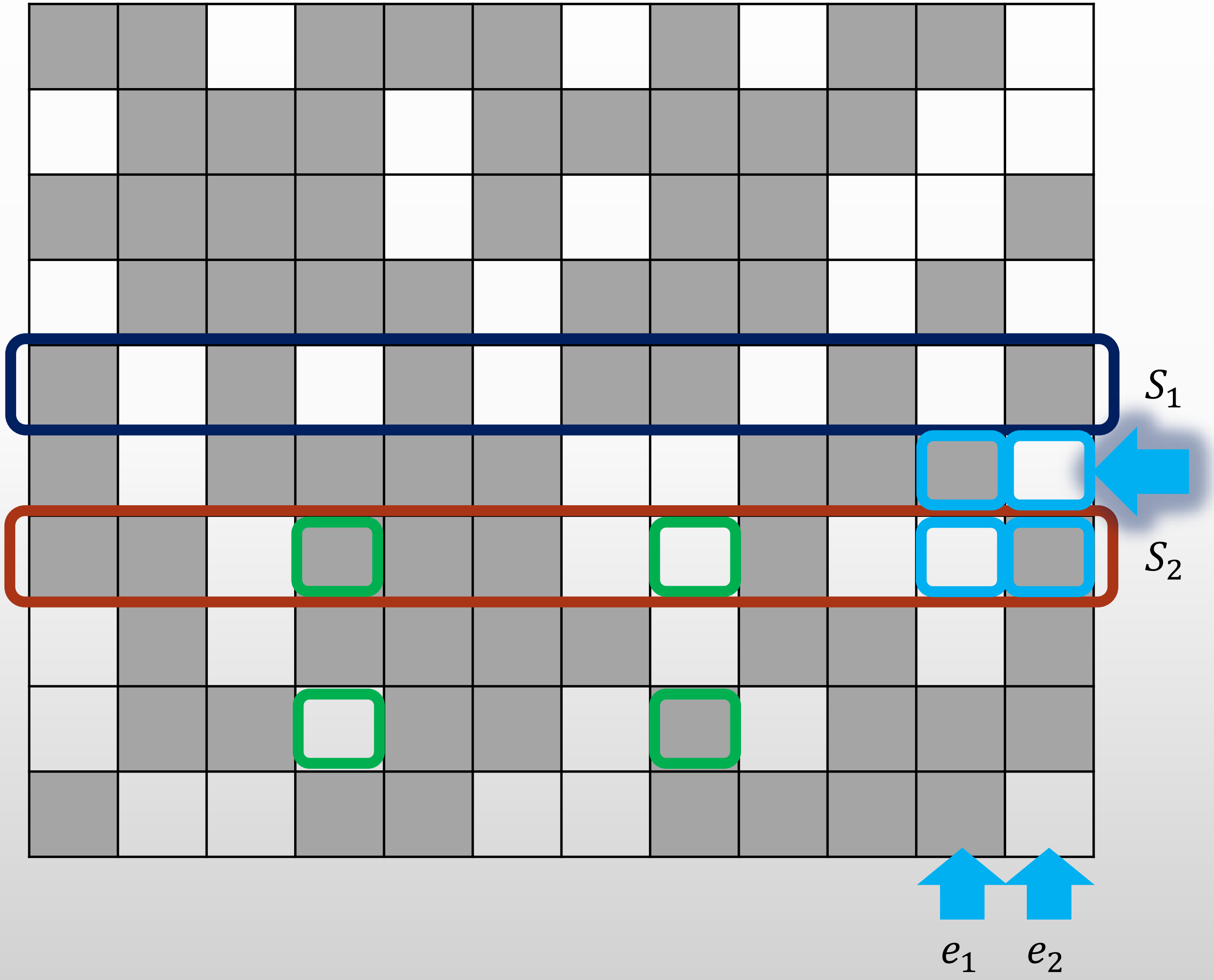
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - **Find a candidate set**
  - swap



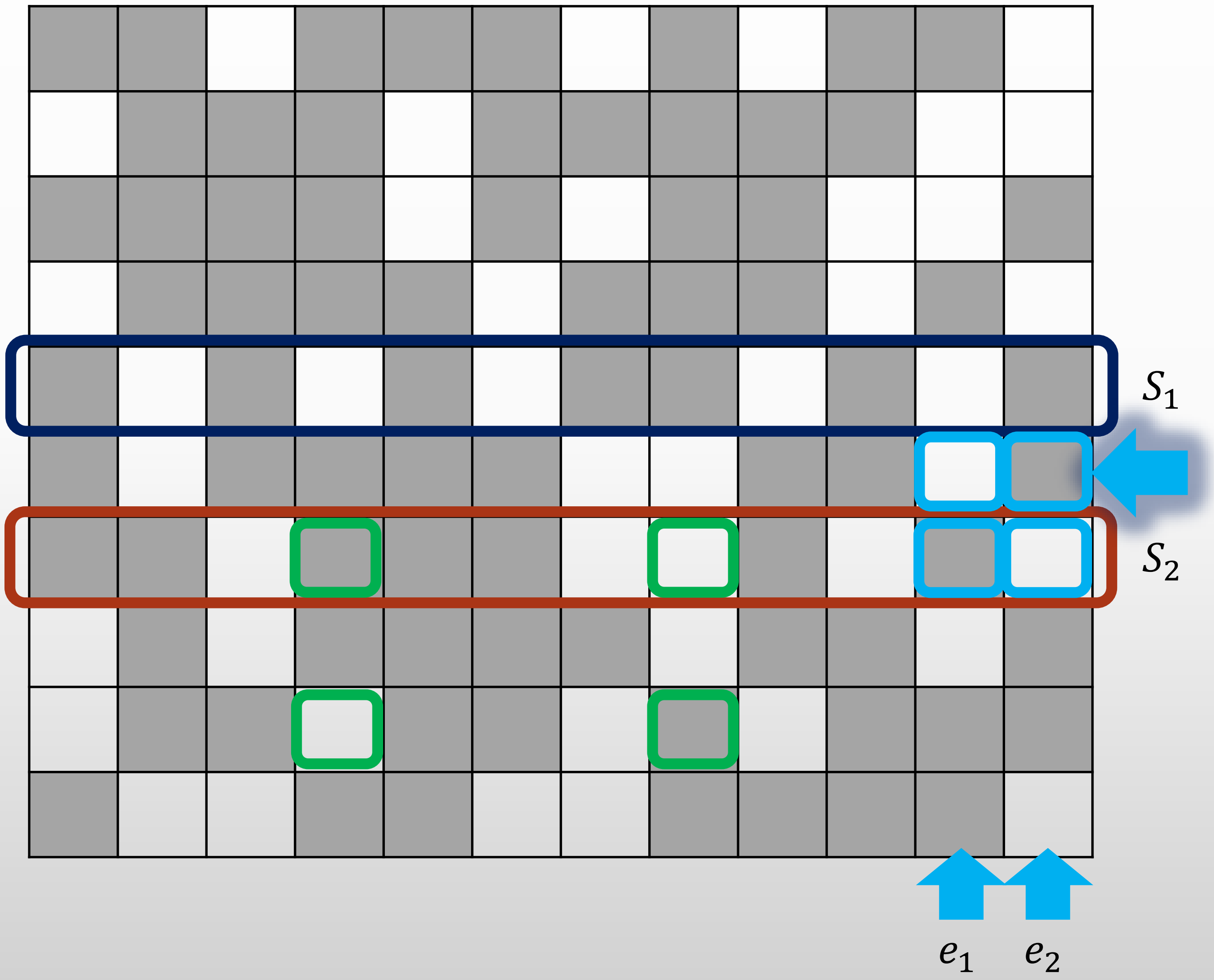
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - **Find a candidate set**
  - swap



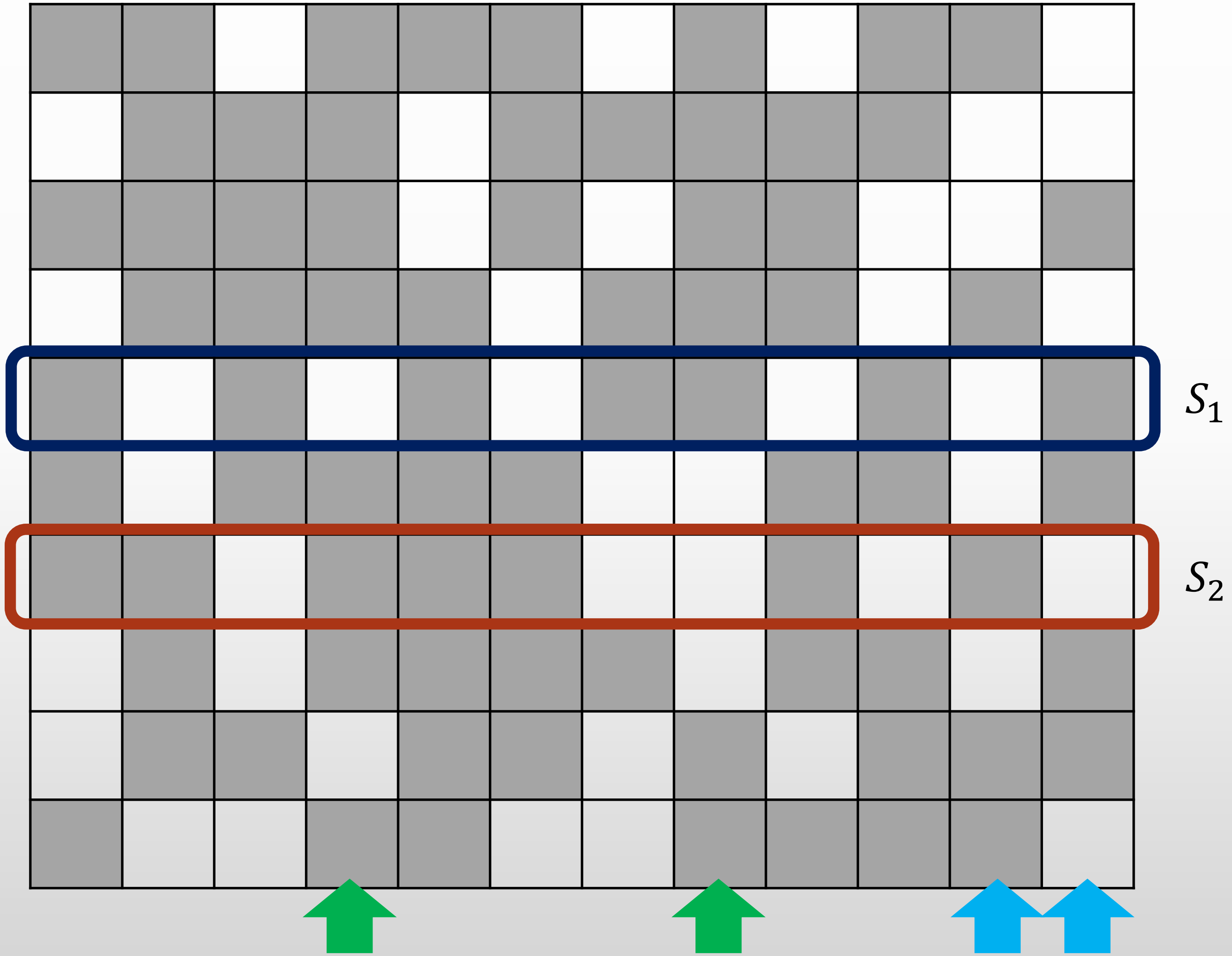
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- **In iteration:**
  - Find a candidate set
  - **swap**



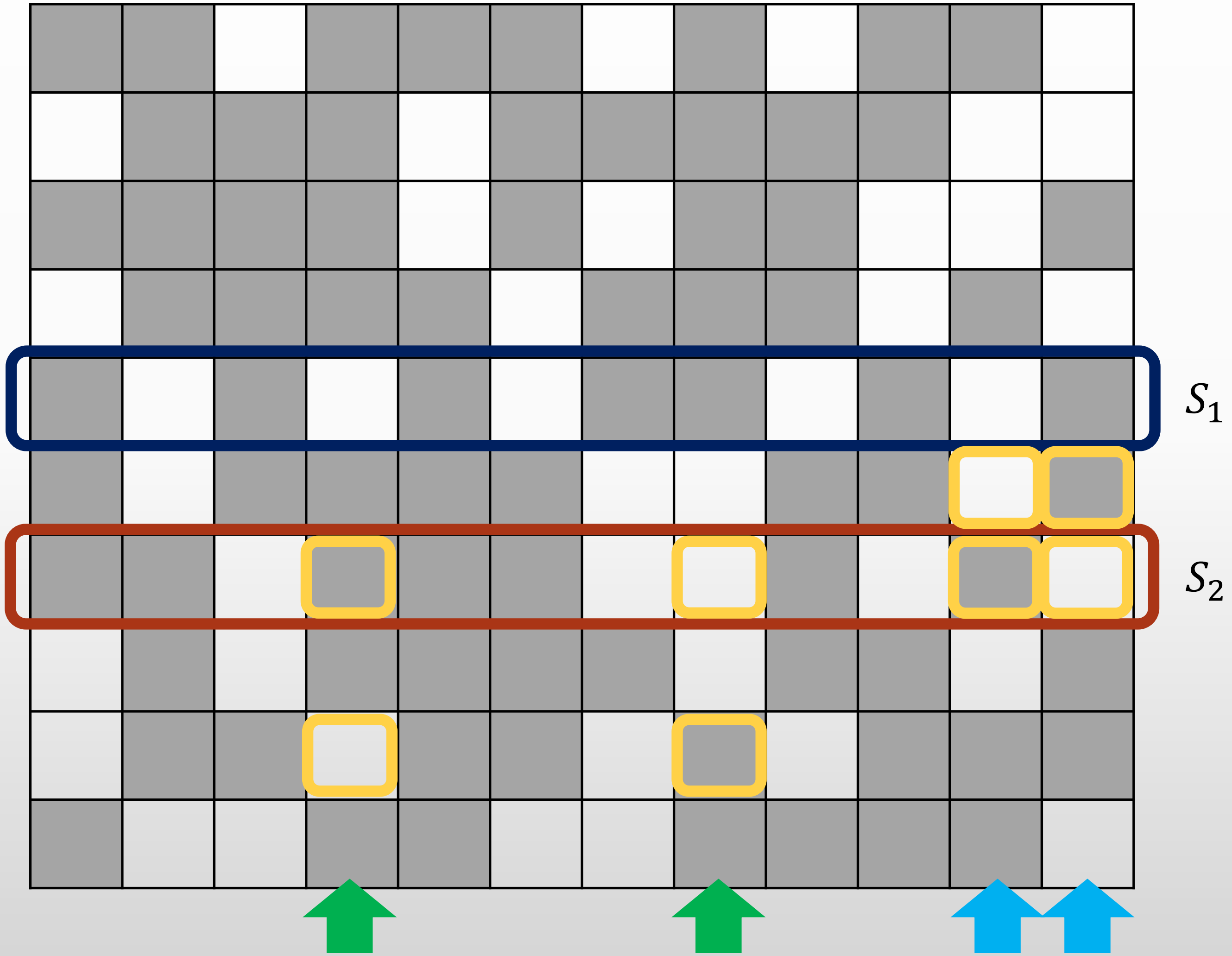
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- In iteration:
  - Find a candidate set
  - swap



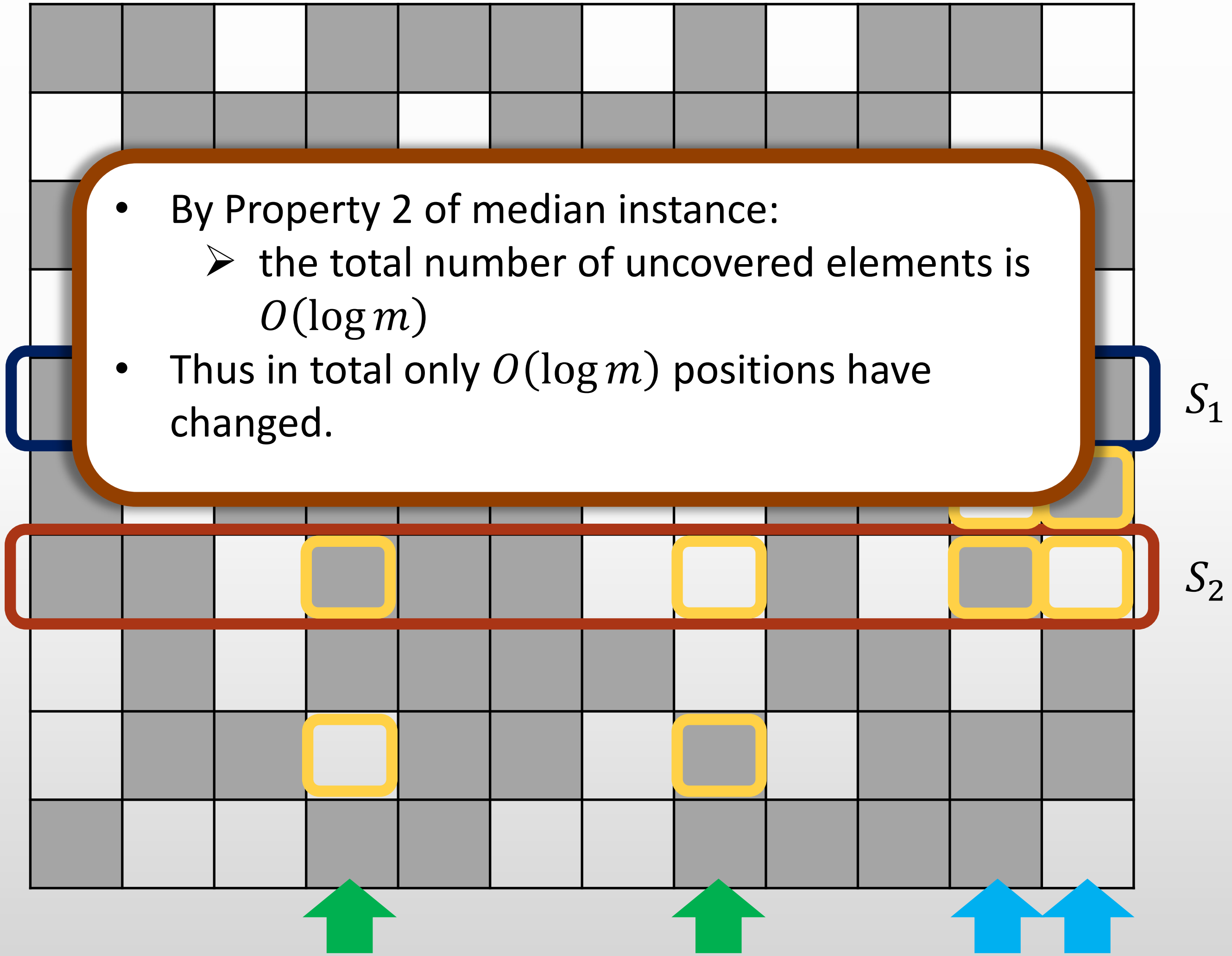
The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- In iteration:
    - Find a candidate set
    - swap



The Randomized Procedure

- Median Instance
- Pick two Sets Uniformly at Random
- Find the elements that are not covered
- Also find the elements that are covered by both
- Assign one element in the intersection to each uncovered element
- In iteration:
  - Find a candidate set
  - swap



- By Property 2 of median instance:
  - the total number of uncovered elements is $O(\log m)$
- Thus in total only $O(\log m)$ positions have changed.

Overall Argument

Lemma: For any element e and any set S, the probability that pair participate in a swap is almost uniform, i.e., $O(\frac{\log m}{mn})$.

- Using other properties of the median instances

Input:

- W.p. 1/c the input is the median instance I^*
- W.p. 1/c the input is a randomly generated modified instance I

Theorem: Any randomized algorithm that with probability at least 2/3 distinguishes whether the minimum Set Cover size is 2 or at least 3 requires $\widetilde{\boldsymbol{\Omega}}(\boldsymbol{mn})$ number of queries.

Next Lecture

More sublinear time algorithms

Sublinear Time Algorithms and Lower Bounds for Set Cover