Greedy Set Cover

Outline

- The Set Cover problem
  - Quite abstract
  - Models many other problems
- The greedy algorithm
  - An $\Theta(\log n)$-approximation

Outline

- The Set Cover problem
  - Quite abstract
  - Models many other problems
- The greedy algorithm
  - An $\Theta(\log n)$-approximation

Learning objectives:
You are able to
- formally describe the set cover problem
- analyze the approximation ratio of the greedy set cover algorithm

The Set Cover Problem

Input:
A universe $U = \{1, \dots, n\}$ of n elements.
A collection $S \subseteq 2^U$ of sets.

Output:
A set $C \subseteq S$ of sets is called a *cover* if each element belongs to at least one set.
The goal is to find a cover with the smallest number of sets.

The Set Cover Problem

Input:
A universe $U = \{1, \dots, n\}$ of n elements.
A collection $S \subseteq 2^U$ of sets.

Denotes all the possible subsets of U

Output:
A set $C \subseteq S$ of sets is called a *cover* if each element belongs to at least one set.
The goal is to find a cover with the smallest number of sets.

The Set Cover Problem

Input:
A universe $U = \{1, \dots, n\}$
of n elements.
A collection $S \subseteq 2^U$ of sets.

Task:
Find the
smallest cover.

The Set Cover Problem

First, it may sound quite abstract.

Input:
A universe $U = \{1, \ldots, n\}$ of n elements.
A collection $S \subseteq 2^U$ of sets.

Task:
Find the smallest cover.

The Set Cover Problem

Input:
A universe $U = \{1, \dots, n\}$ of n elements.
A collection $S \subseteq 2^U$ of sets.

Task:
Find the smallest cover.

First, it may sound quite abstract.

Find the smallest set of movies that everyone likes?

The Set Cover Problem

First, it may sound quite abstract.

Input:
A universe $U = \{1, \dots, n\}$ of n elements.
A collection $S \subseteq 2^U$ of sets.

Find the smallest set of movies that everyone likes?

Task:
Find the smallest cover.

Many "natural" covering and graph problems like minimum vertex cover and minimum dominating set are special cases of minimum set cover.

The Set Cover Problem is Hard (?)

The set cover problem is NP-hard

We believe that it cannot be solved exactly in polynomial time.

Outline

- The Set Cover problem
  - Quite abstract
  - Models many more specific problems
- The greedy algorithm
  - An $\Theta(\log n)$-approximation

Greedy Algorithm for Set Cover

The greedy algorithm is an $O(\log n)$-approximation algorithm.

The analysis is tight!

Greedy Algorithm for Set Cover

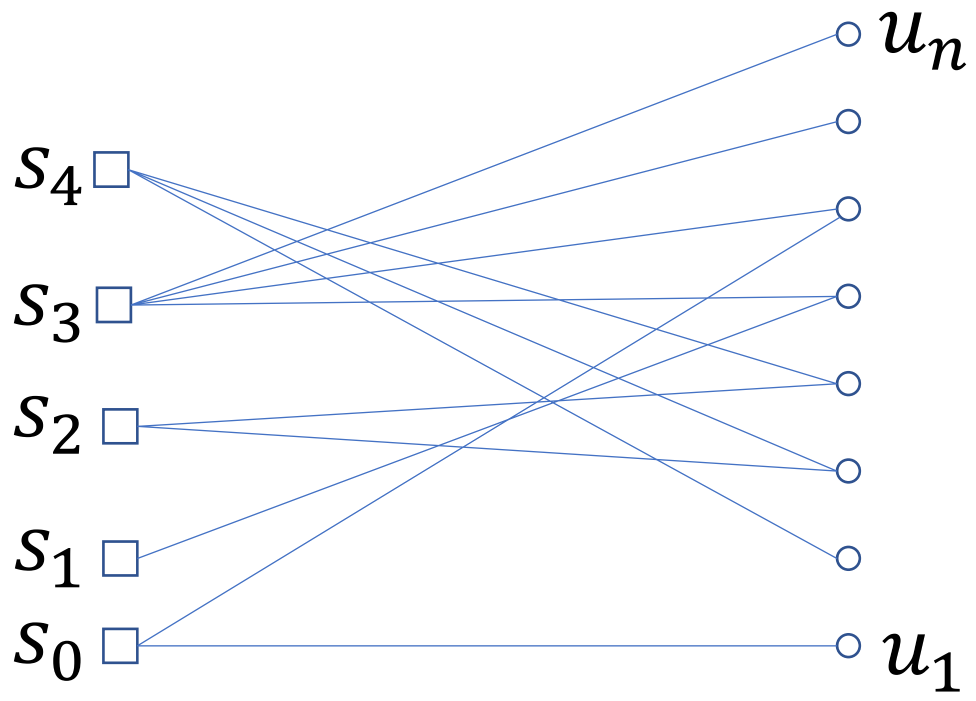
Anything better is APX-hard

The greedy algorithm is an $O(\log n)$-approximation algorithm.

The analysis is tight!

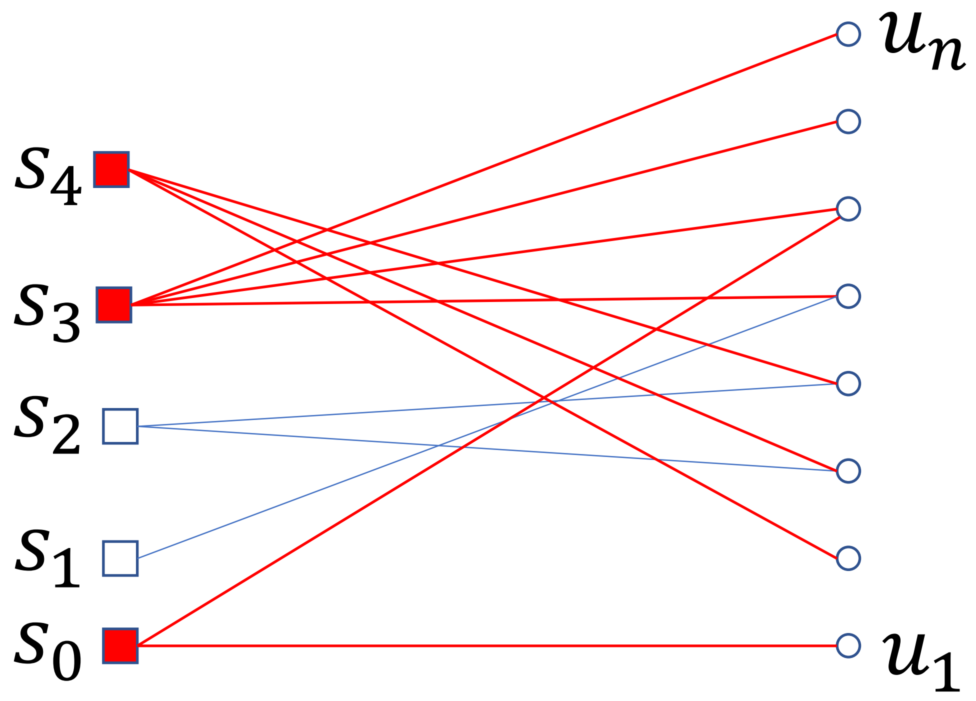
Greedy Algorithm for Set Cover

Consider a bipartite graph where sets on the left and elements on the right.



Greedy Algorithm for Set Cover

Consider a bipartite graph where sets on the left and elements on the right.



Sets s_0, s_3 and s_4 form a cover

Greedy Algorithm for Set Cover

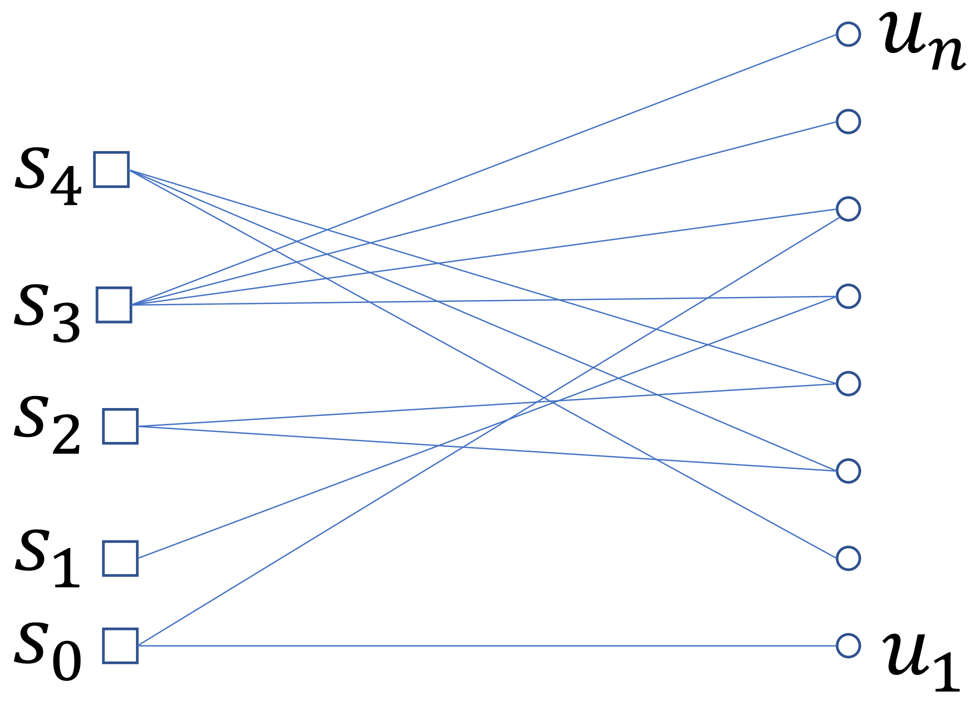
Remove covered nodes from U

Update the nodes in S

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
  Find highest degree node $s \in S$
  $U := U \setminus N^1(s)$
  For each($v \in S$**)**
    $N^1(v) := N^1(v) \cap U$

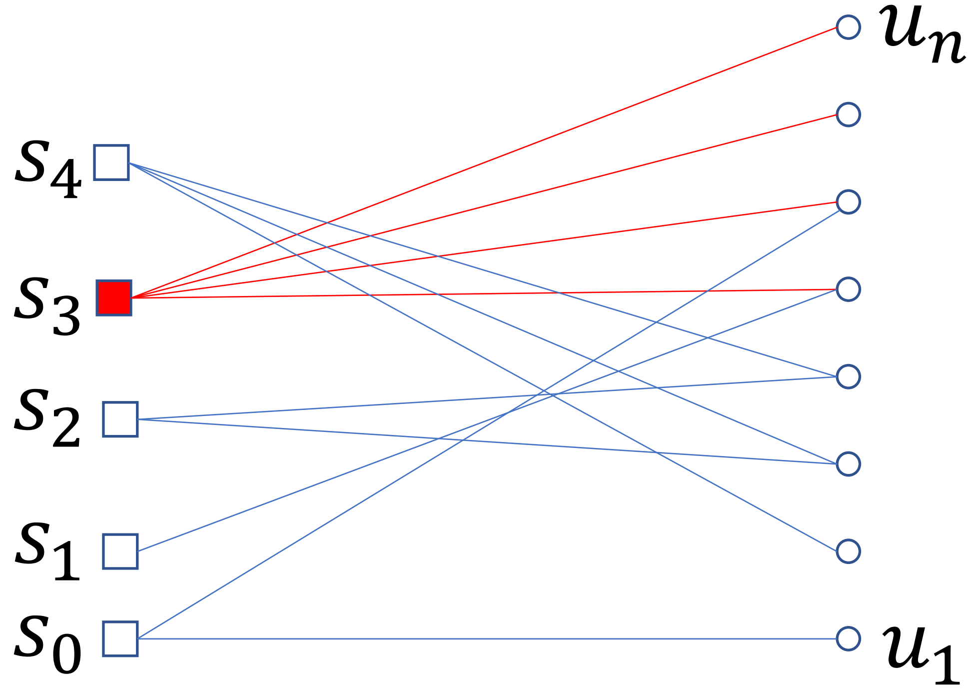
Greedy Algorithm for Set Cover

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



Greedy Algorithm for Set Cover

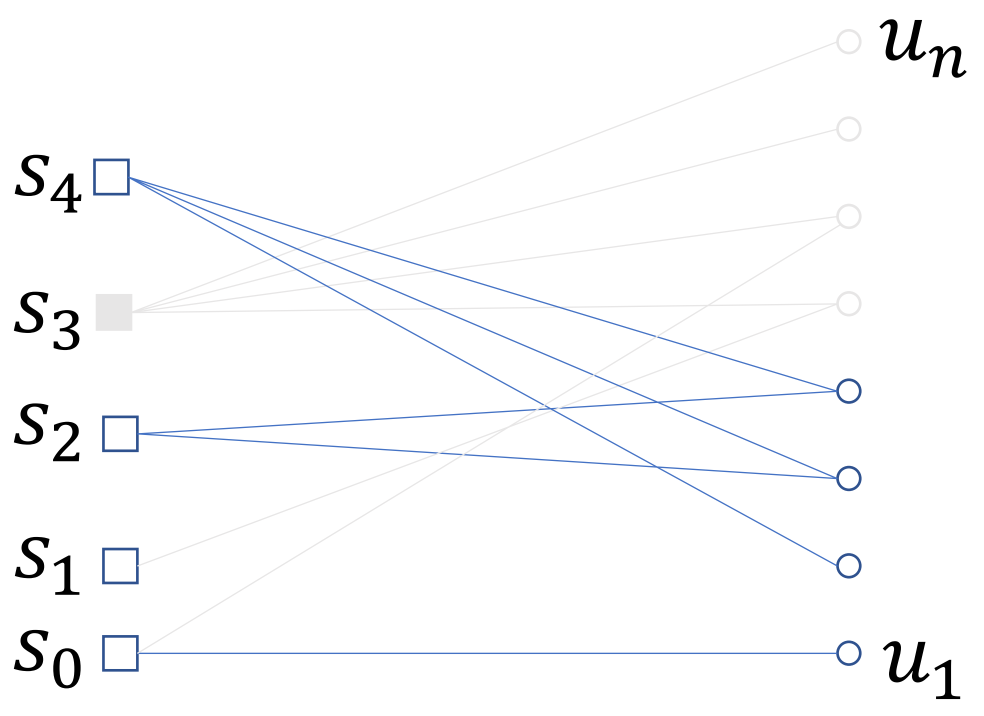
Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



Set s_3 has the most neighbors.

Greedy Algorithm for Set Cover

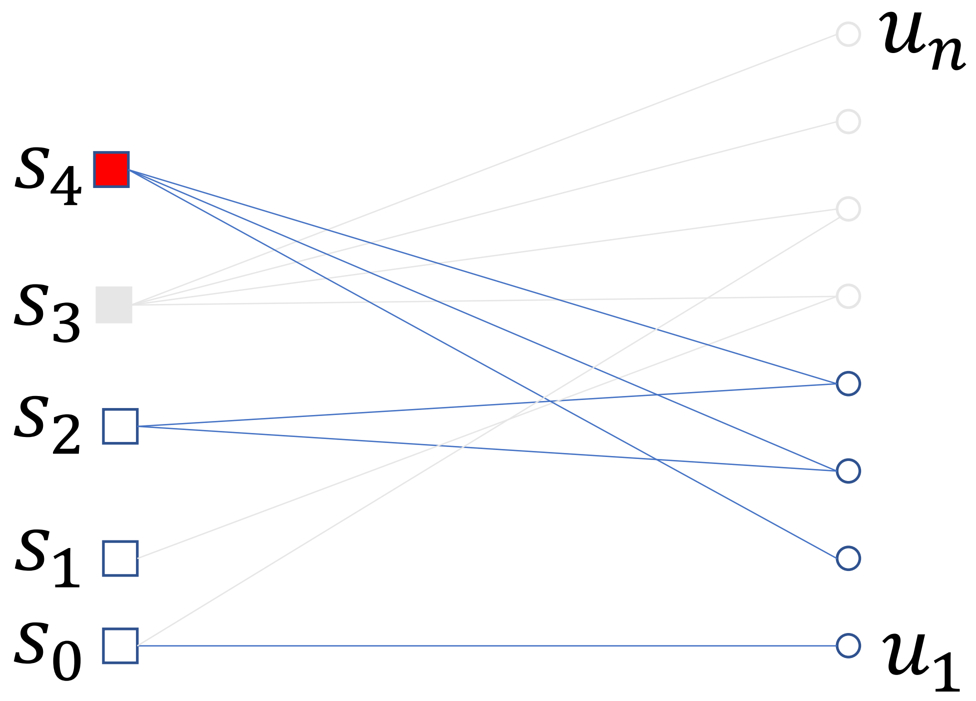
Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



Remove s_3, the covered elements and the related edges

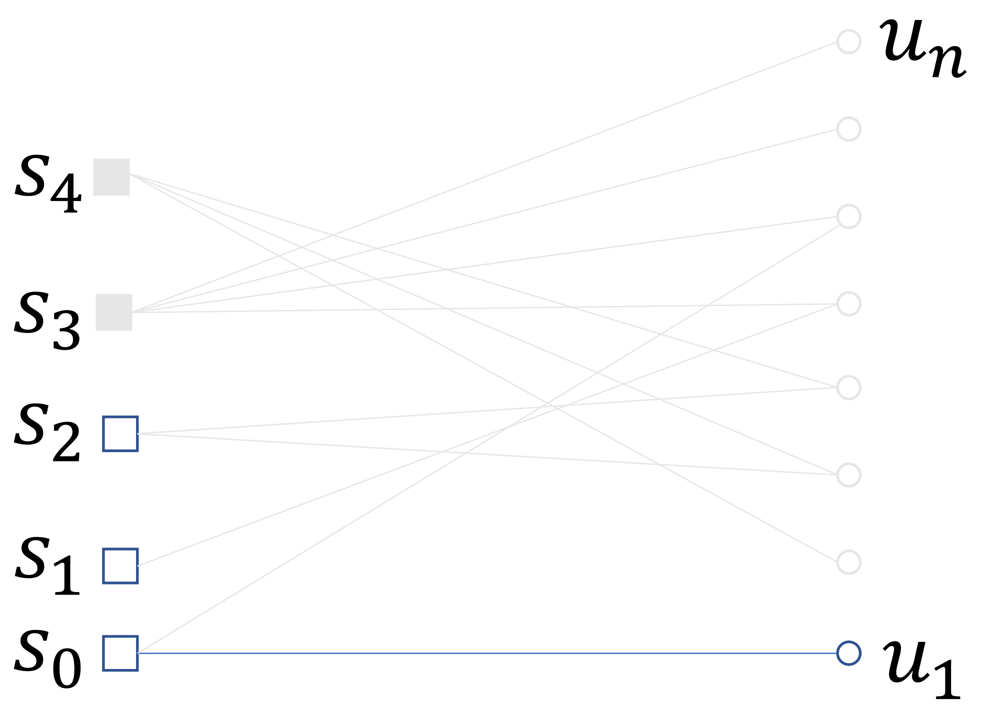
Greedy Algorithm for Set Cover

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



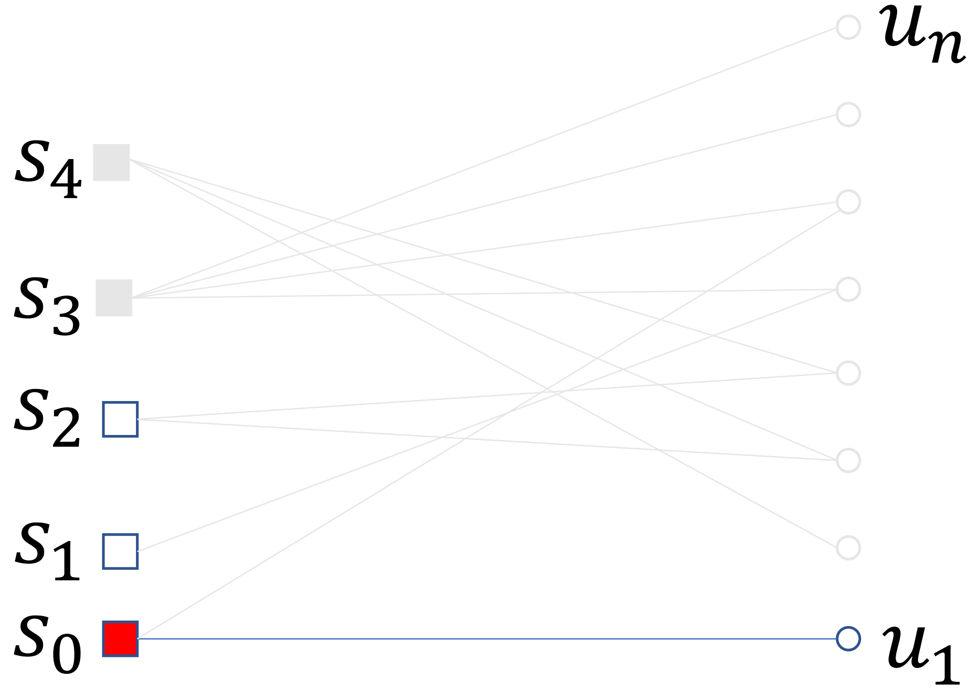
Greedy Algorithm for Set Cover

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



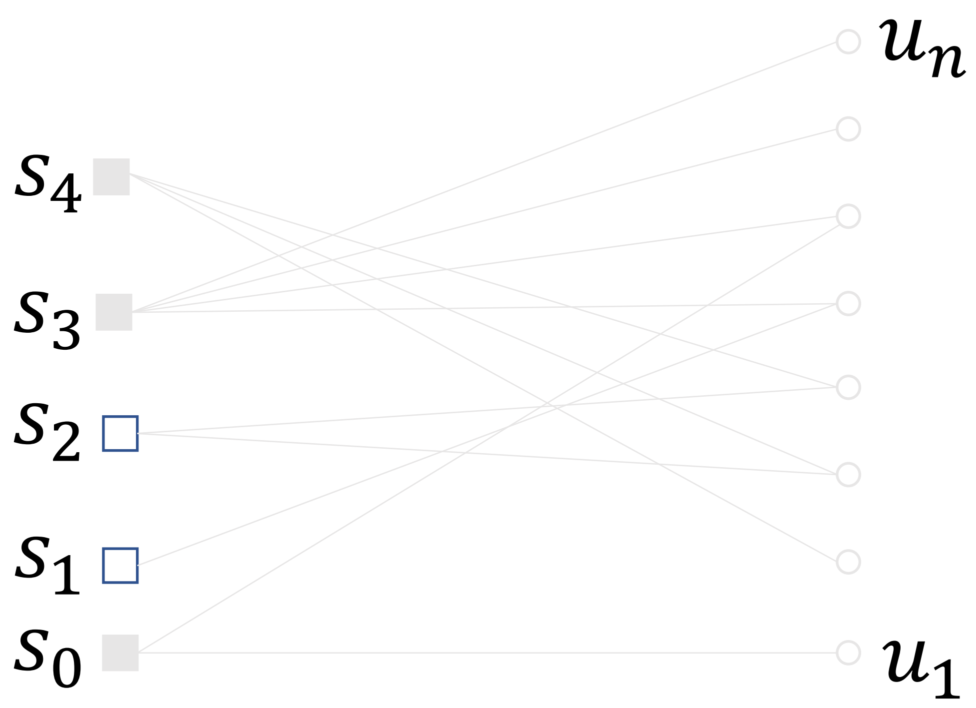
Greedy Algorithm for Set Cover

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



Greedy Algorithm for Set Cover

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$



The resulting set cover is $\{s_0, s_3, s_4\}$

Approximation Analysis

How to relate cost(OPT) with cost(greedy)?

Approximation Analysis

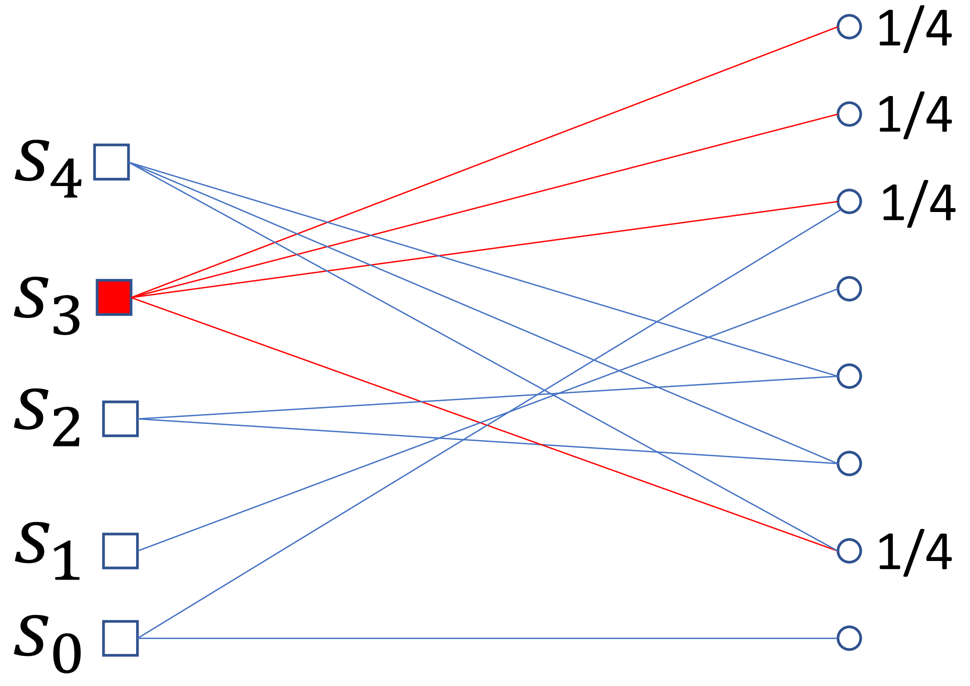
How to relate cost(OPT) with cost(greedy)?

Idea:
Consider some set-node $s \in OPT$ and some $u \in N(s)$. If greedy covers u with some other set-node s', then deg(s') was at least as large as deg(s) during that iteration.

Approximation Analysis

Give one euro to each set-node selected by the greedy algorithm.

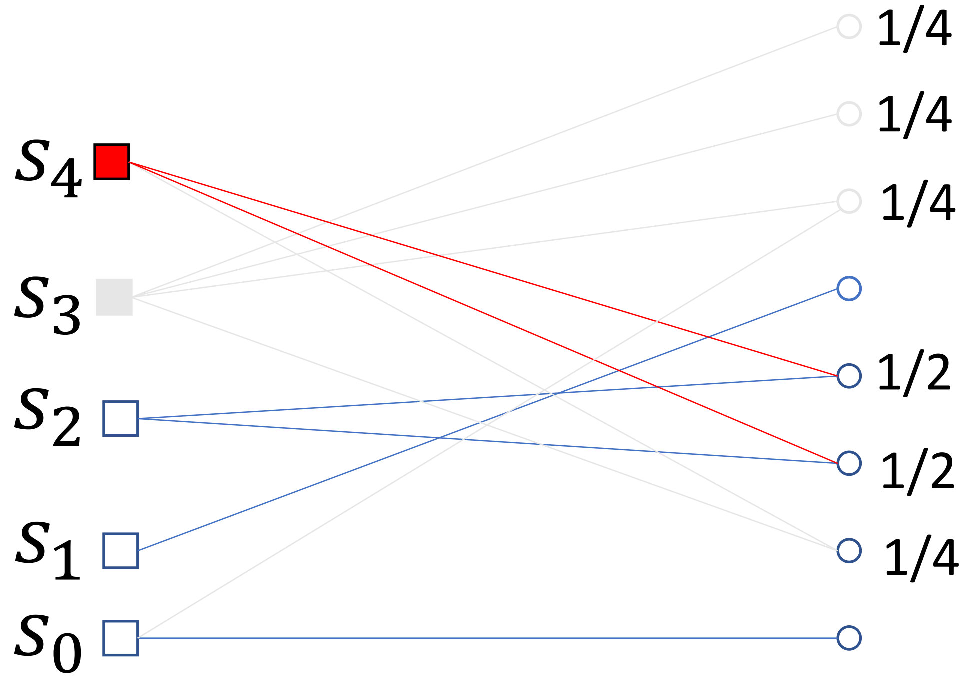
This euro is divided equally among the element-nodes covered in that iteration



Approximation Analysis

Give one euro to each set-node selected by the greedy algorithm.

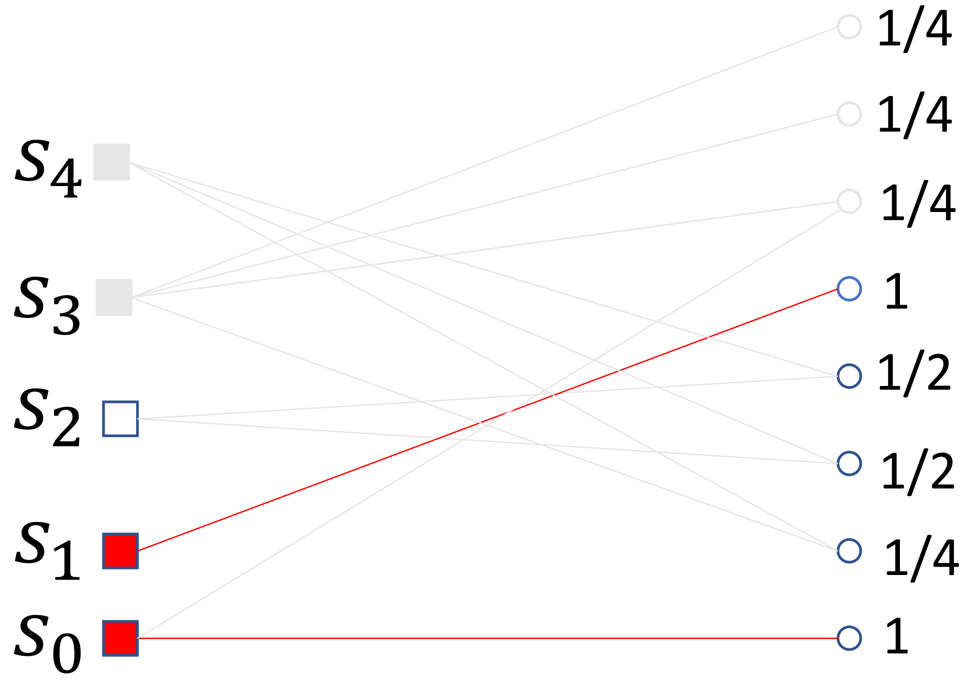
This euro is divided equally among the element-nodes covered in that iteration



Approximation Analysis

Give one euro to each set-node selected by the greedy algorithm.

This euro is divided equally among the element-nodes covered in that iteration



Approximation Analysis

Observation:
Cost(greedy) $= \Sigma_{v \in U}\, f(v)$

Observation:
OPT must cover all nodes in U

Denote $f(v) =$ #euros of v

Approximation Analysis

Observation:
Cost(greedy) $= \Sigma_{v \in U} f(v)$

Denote $f(v)$ = #euros of v

Observation:
OPT must cover all nodes in U

Cost(greedy) $=$
$\Sigma_{s \in OPT} \Sigma_{v \in N(s)} f(v)$

Approximation Analysis

Observation:
Cost(greedy) $= \Sigma_{v \in U} f(v)$

Task:
Bound max $\Sigma_{v \in N(s)}$ #euros

Observation:
OPT must cover all nodes in U

↓

Cost(greedy) $=$
$\Sigma_{s \in OPT} \Sigma_{v \in N(s)} f(v)$

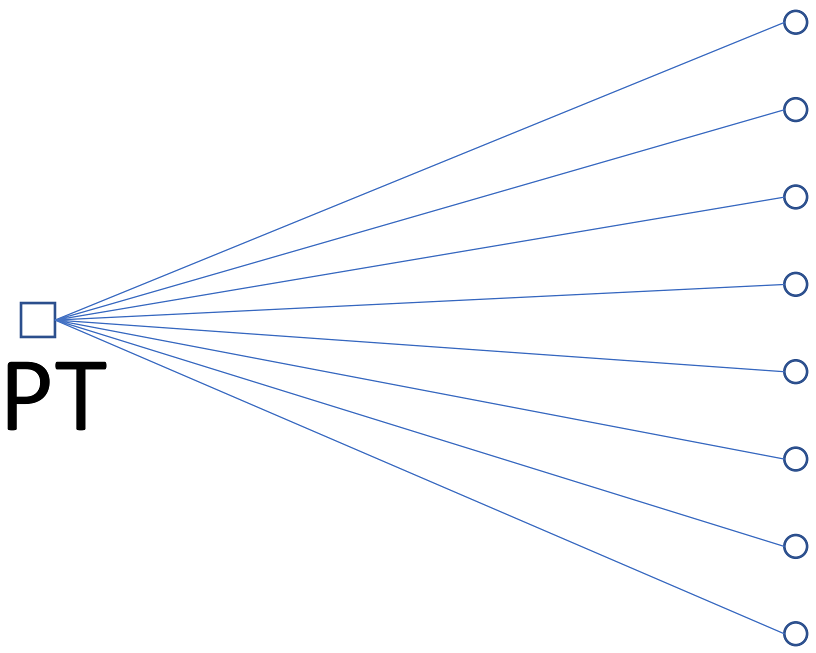
↓

Cost(greedy) $\leq$
cost(OPT) $\cdot$ max $\Sigma_{v \in N(s)} f(v)$

Approximation Analysis

Task:
Bound max $\Sigma_{v \in N(s)}$ #euros of v

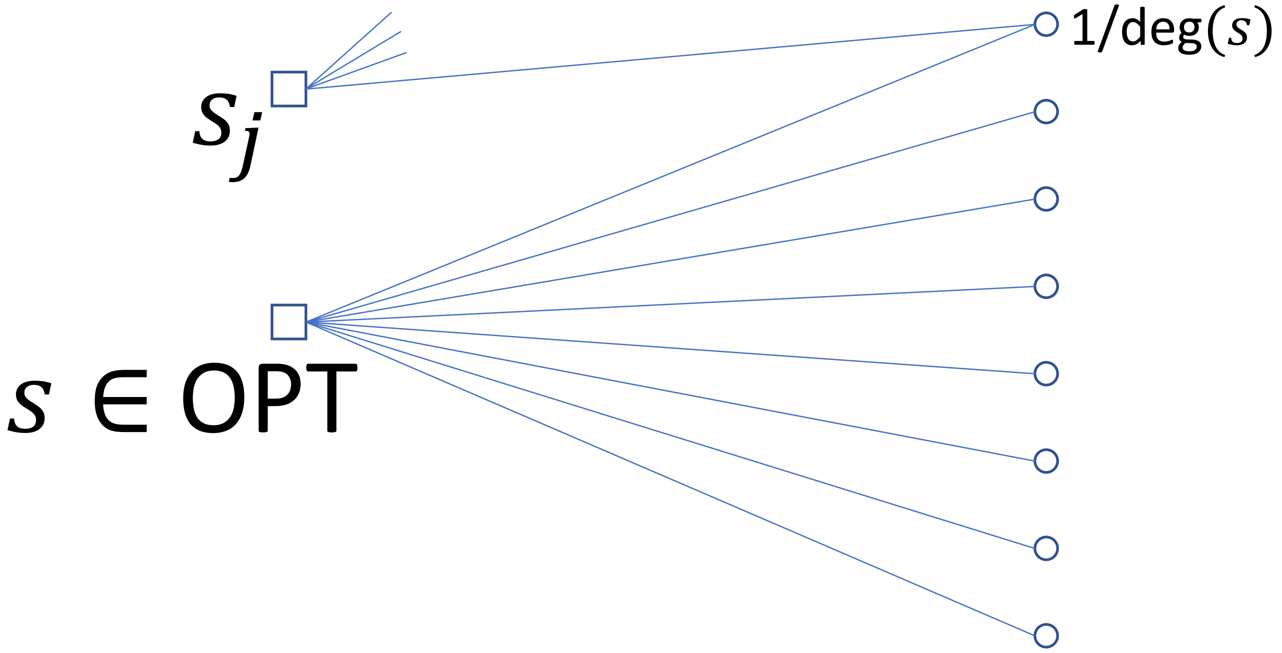
$s \in$ OPT



If $\deg(s) = d$ in iteration i, then a neighbor gets at most $1/d$ euros

Approximation Analysis

Task:
Bound max $\Sigma_{v \in N(s)}$ #euros of v

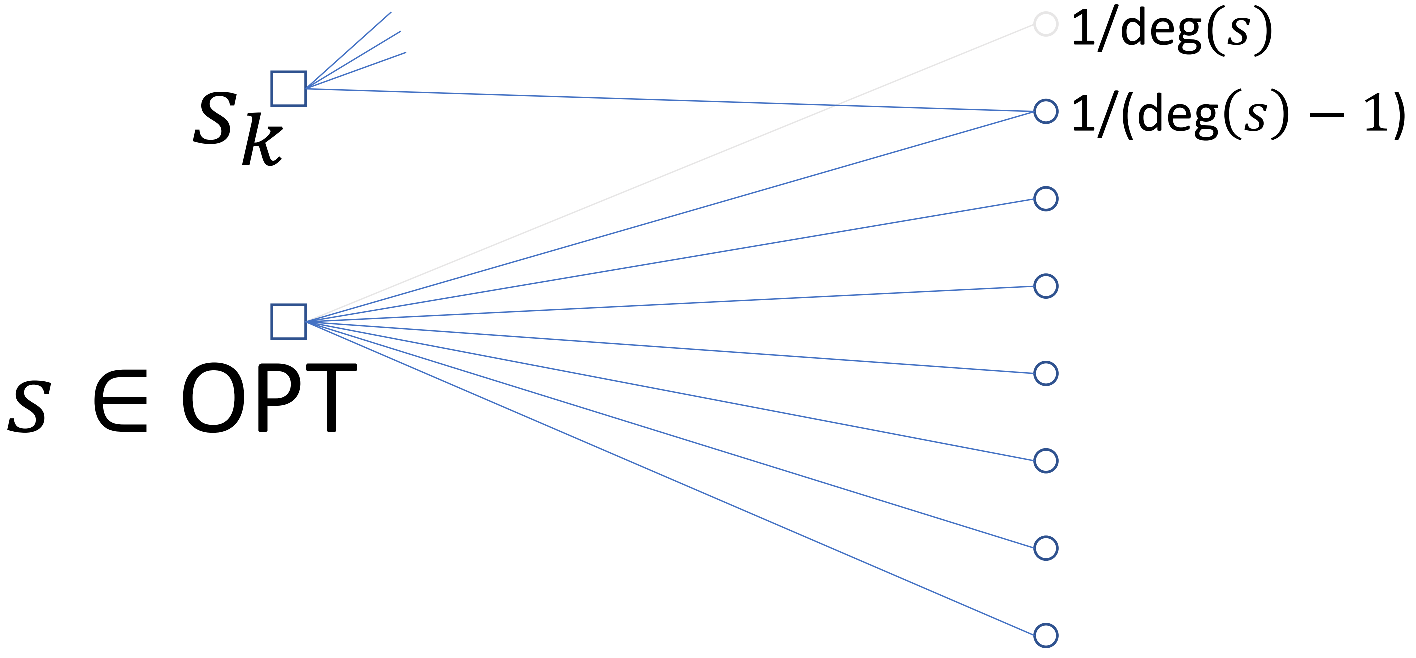


Greedy chooses the highest degree in each iteration i

If $\deg(s) = d$ in iteration i, then a neighbor gets at most $1/d$ euros

Approximation Analysis

Task:
Bound max $\Sigma_{v \in N(s)}$ #euros of v



If $\deg(s) = d$ in iteration $\geq i$, then a neighbor gets at most $1/d$ euros

Approximation Analysis

Lemma:
Suppose the degree of s is d in some iteration. Any neighbor covered in this iteration gets at most $1/d$ euros.

At most i neighbors get $\geq 1/i$ euros

Approximation Analysis

Lemma:
Suppose the degree of s is d in some iteration. Any neighbor covered in this iteration gets at most $1/d$ euros.

At most i neighbors get $\geq 1/i$ euros

Denote $f(v) = $ #euros of v

Lemma:
$\Sigma_{u \in N(s)} f(u) \leq \Sigma_{i=1}^{|N(s)|} \frac{1}{i} = O(\log N(s))$

Harmonic sum

Approximation Analysis

Observation:
Cost(greedy) $= \Sigma_{v \in U} f(v)$

Denote $f(v) =$ #euros of v

Observation:
OPT must cover all nodes in U

Cost(greedy) $=$
$\Sigma_{S \in OPT} \Sigma_{v \in N(s)}$#euros

Cost(greedy) $\leq$
cost(OPT) $\cdot$ max $\Sigma_{v \in N(s)}$ #euros

Approximation Analysis

Observation:
Cost(greedy) $= \Sigma_{v \in U} f(v)$

Denote $f(v) =$ #euros of v

Lemma:

$$\Sigma_{v \in N(s)} f(v) \leq \Sigma_{i=1}^{|N(s)|} \frac{1}{i} = O(\log N(s))$$

Observation:
OPT must cover all nodes in U

Cost(greedy) $=$
$\Sigma_{S \in OPT} \Sigma_{v \in N(s)} f(v)$

Cost(greedy) $\leq$
cost(OPT) $\cdot$ max $\Sigma_{v \in N(s)} f(v)$

Cost(greedy) $\leq$ cost(OPT) $\cdot\, O(\log n)$

Greedy Algorithm for Set Cover

The greedy algorithm is an $O(\log n)$-approximation algorithm.

The analysis is tight!

Greedy Algorithm for Set Cover

The analysis is tight!

Greedy Algorithm for Set Cover

The analysis is tight!

?

Disprove:
For every input, the solution by the greedy algorithm is within an $o(\log n)$ factor.

Greedy Algorithm for Set Cover

The analysis is tight!

?

It is enough to find an example where this is not true.

Disprove:
For every input, the solution by the greedy algorithm is within an $o(\log n)$ factor.

Greedy Algorithm for Set Cover

The analysis is tight!

?

Disprove:
For every input, the solution by the greedy algorithm is within an $o(\log n)$ factor.

It is enough to find an example where this is not true.

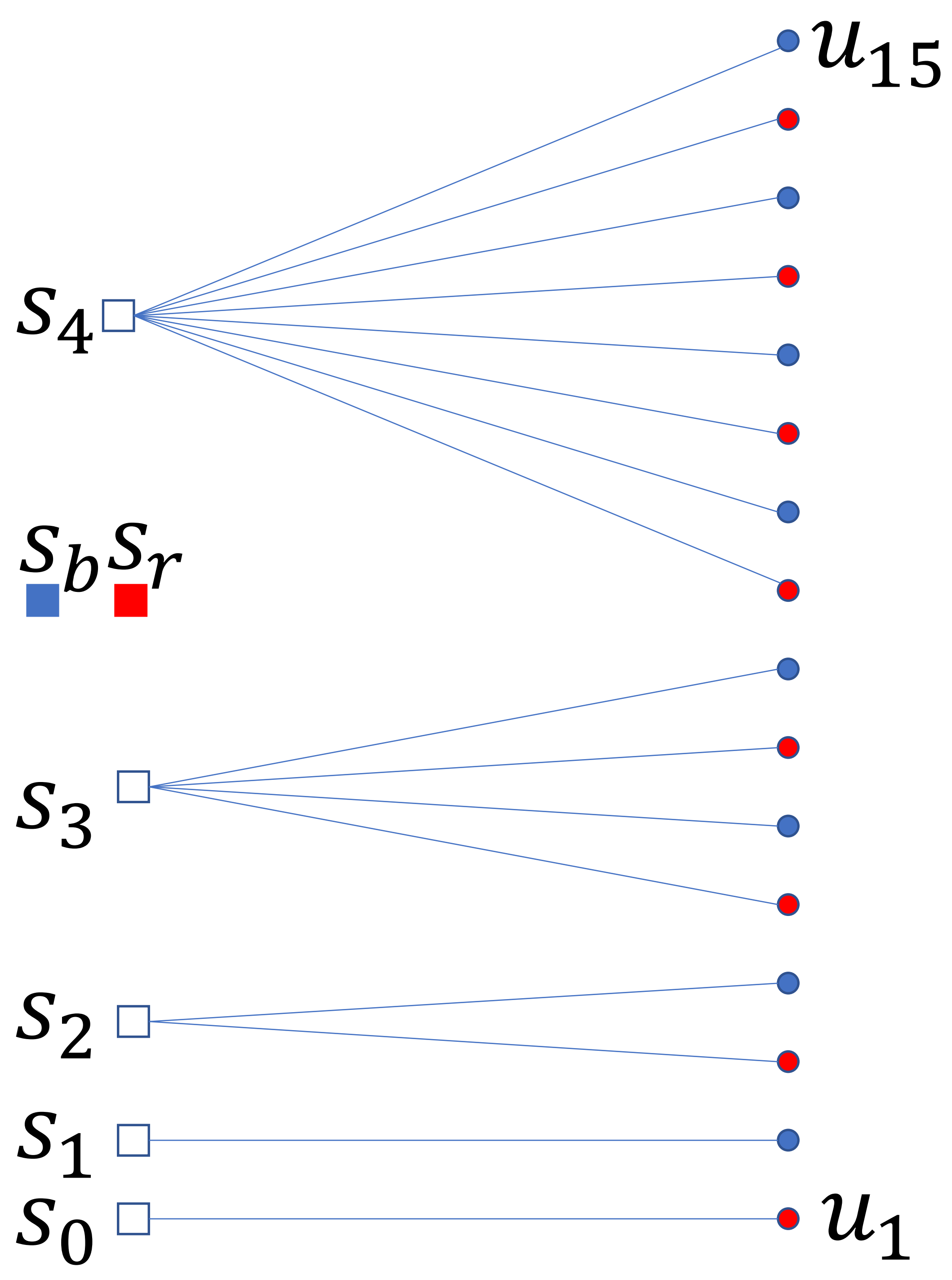
To be precise, we need an infinite family of inputs to deal with the O-notation.

Greedy Algorithm for Set Cover

Universe $u_1, \ldots u_n$

Set s_i covers elements $u_{2^{i-1}}, \ldots, u_{2^i-1}$

Special sets s_b and s_r cover odd and even elements, respectively

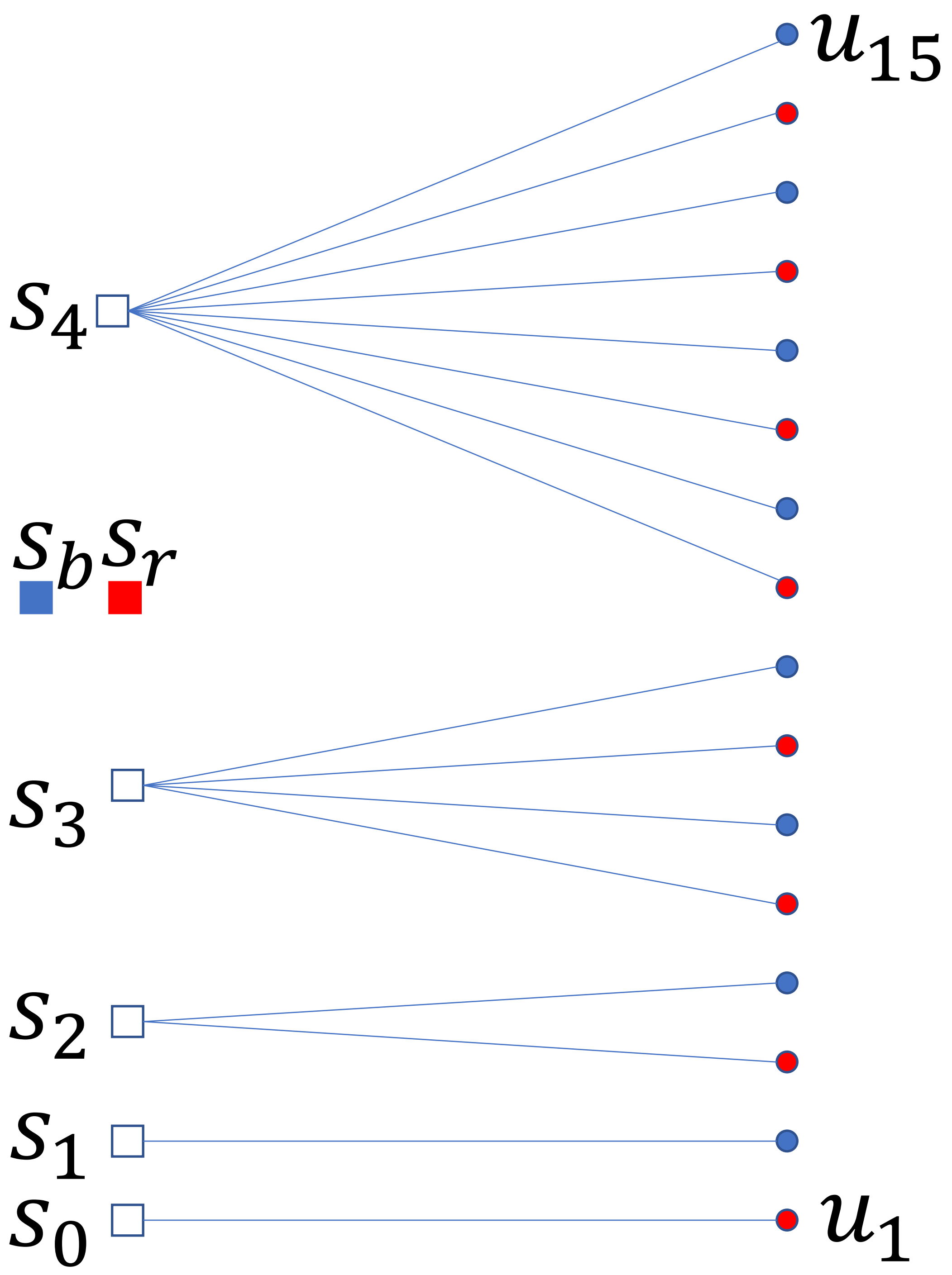


Greedy Algorithm for Set Cover

Universe $u_1, \ldots u_n$

Set s_i covers elements $u_{2^{i-1}}, \ldots, u_{2^i-1}$

Special sets s_b and s_r cover odd and even elements, respectively



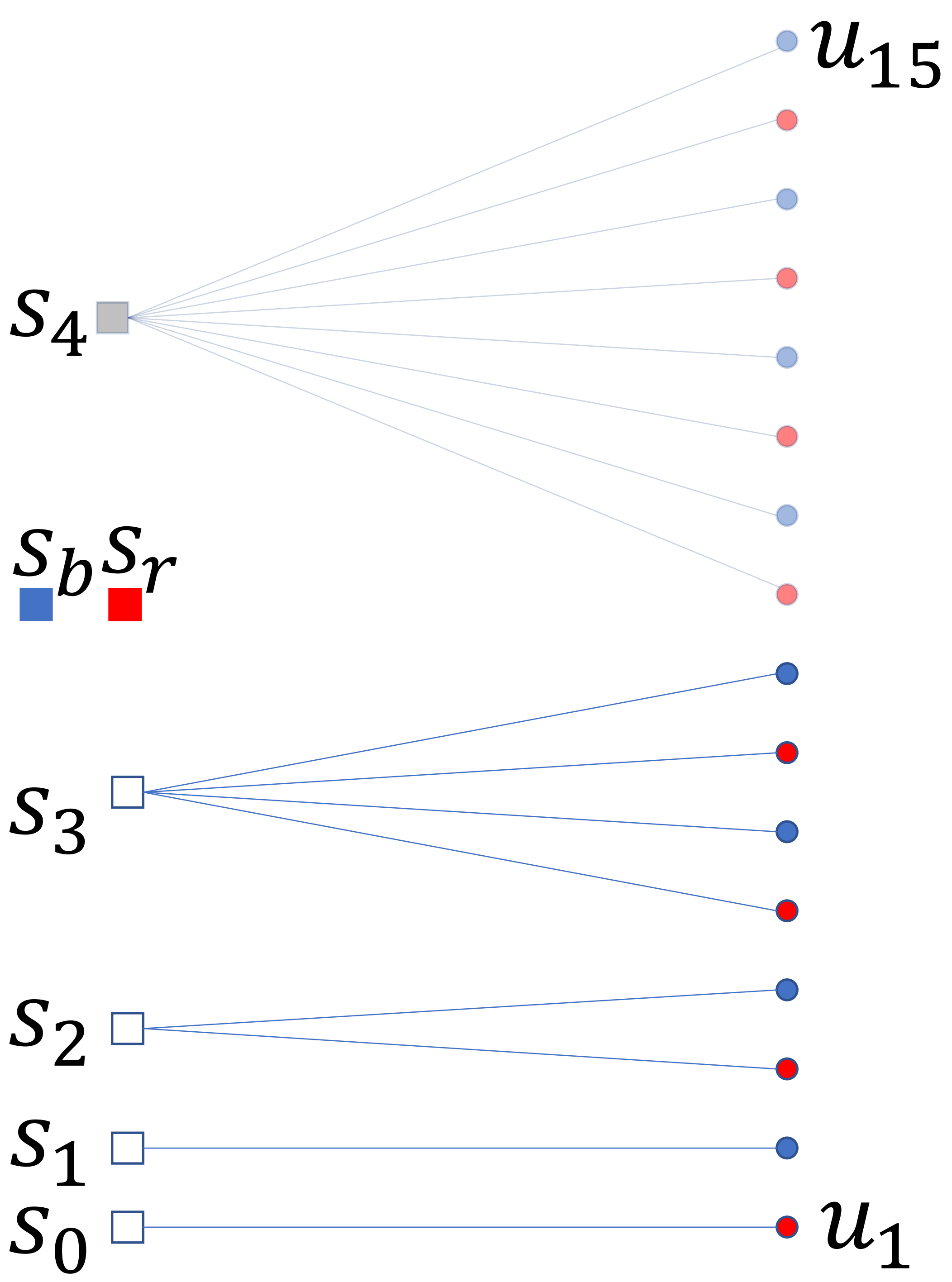
First greedy choice is s_4 (equally good as s_b and s_r)

Greedy Algorithm for Set Cover

Universe $u_1, \dots u_n$

Set s_i covers elements $u_{2^{i-1}}, \dots, u_{2^i-1}$

Special sets s_b and s_r cover odd and even elements, respectively



First greedy choice is s_4 (equally good as s_b and s_r)

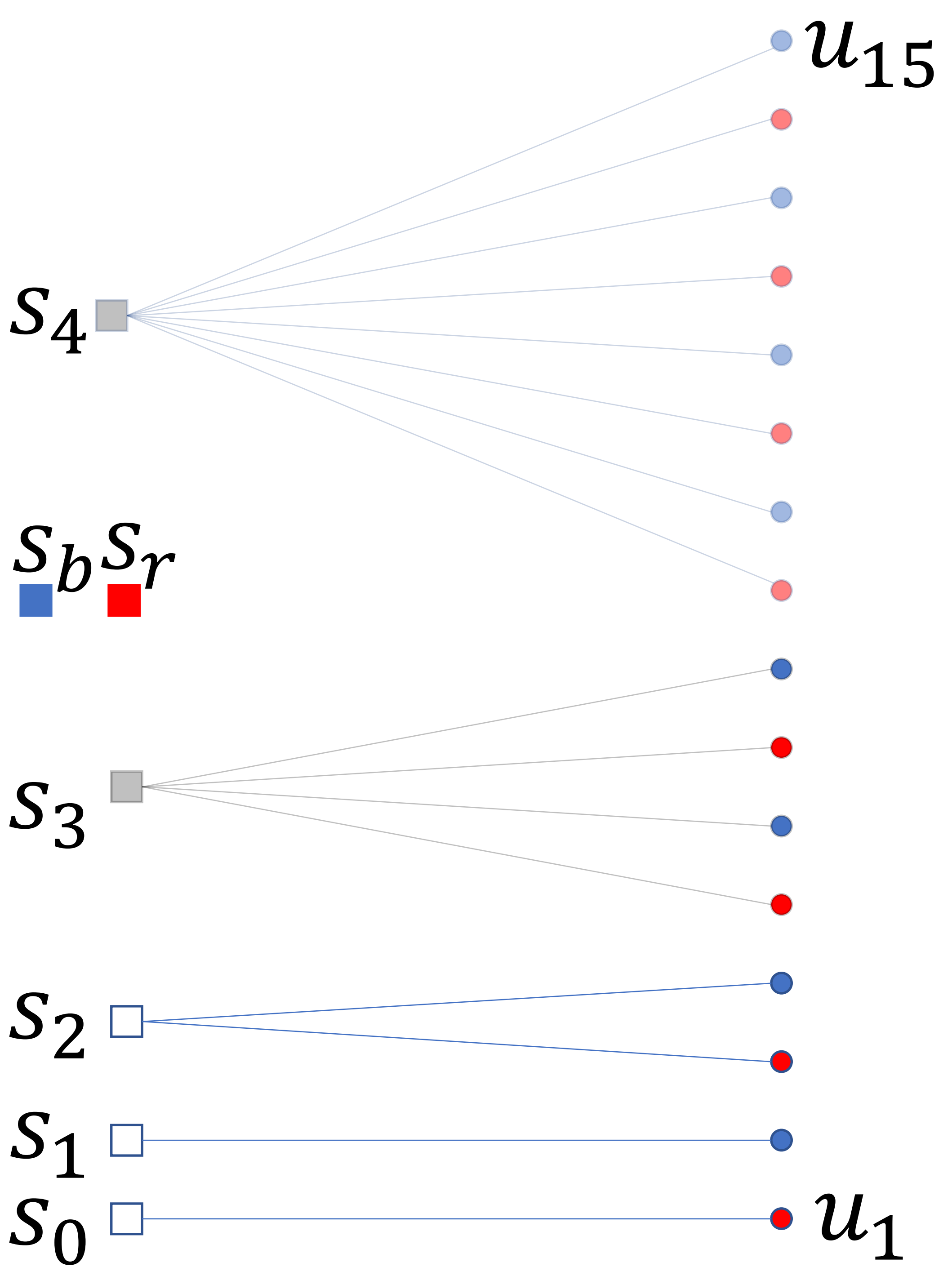
After updating s_b and s_r, greedy chooses s_3

Greedy Algorithm for Set Cover

Universe $u_1, \ldots u_n$

Set s_i covers elements $u_{2^{i-1}}, \ldots, u_{2^i-1}$

Special sets s_b and s_r cover odd and even elements, respectively



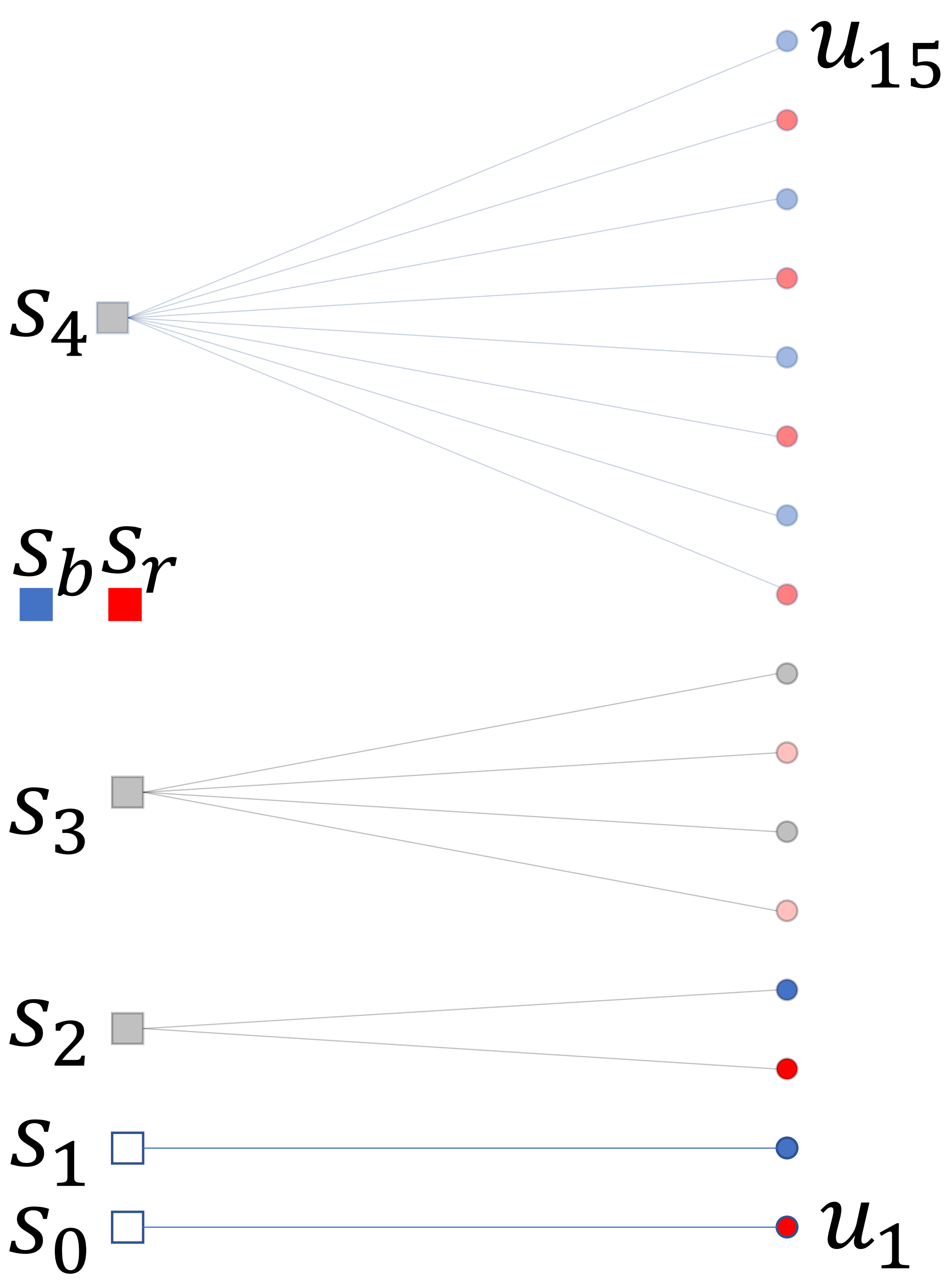
First greedy choice is s_4 (equally good as s_b and s_r)

Greedy Algorithm for Set Cover

Universe $u_1, \ldots u_n$

Set s_i covers elements $u_{2^{i-1}}, \ldots, u_{2^i - 1}$

Special sets s_b and s_r cover odd and even elements, respectively



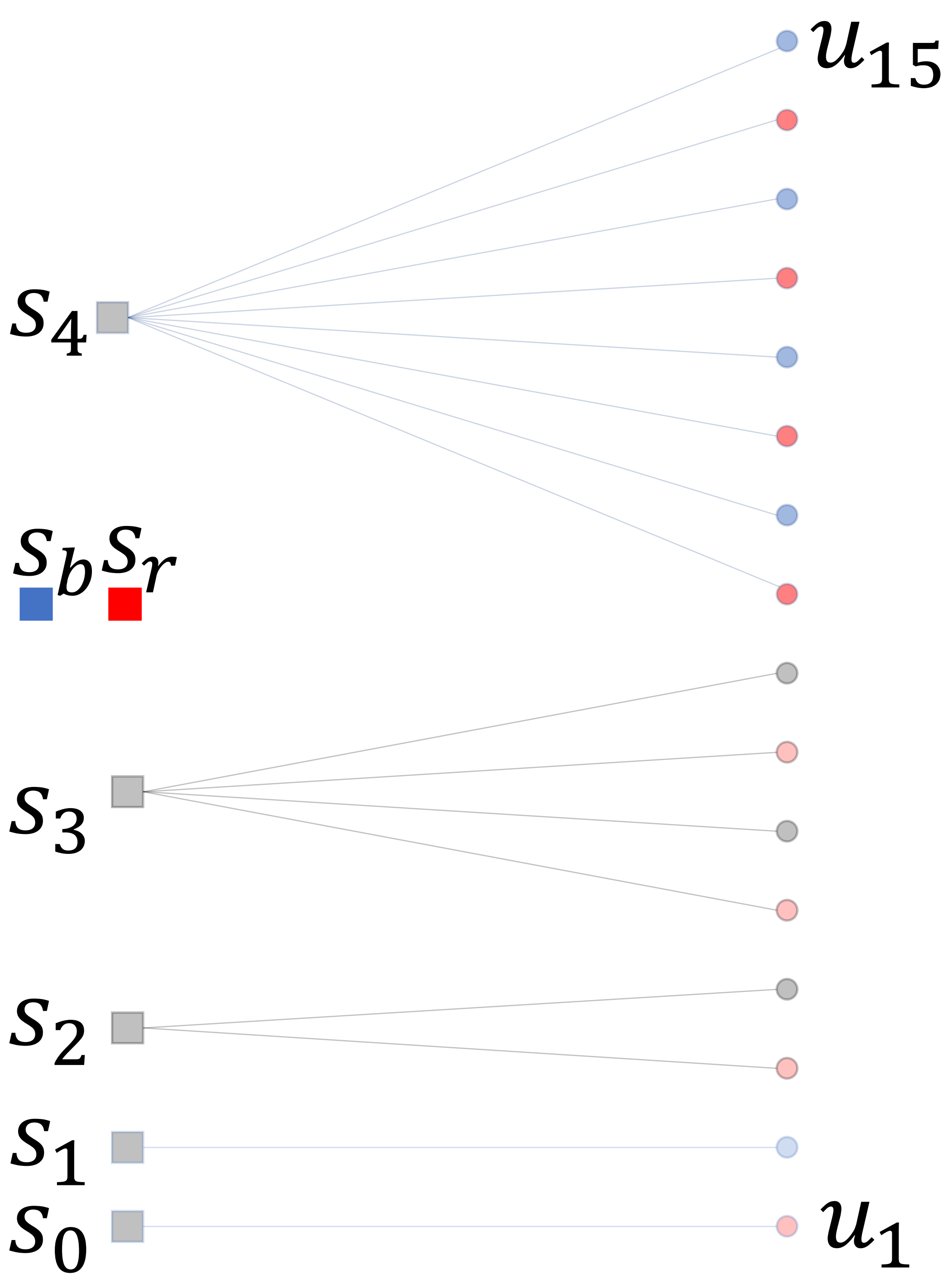
First greedy choice is s_4 (equally good as s_b and s_r)

Greedy Algorithm for Set Cover

Universe $u_1, \dots u_n$

Set s_i covers elements $u_{2^{i-1}}, \dots, u_{2^i-1}$

Special sets s_b and s_r cover odd and even elements, respectively



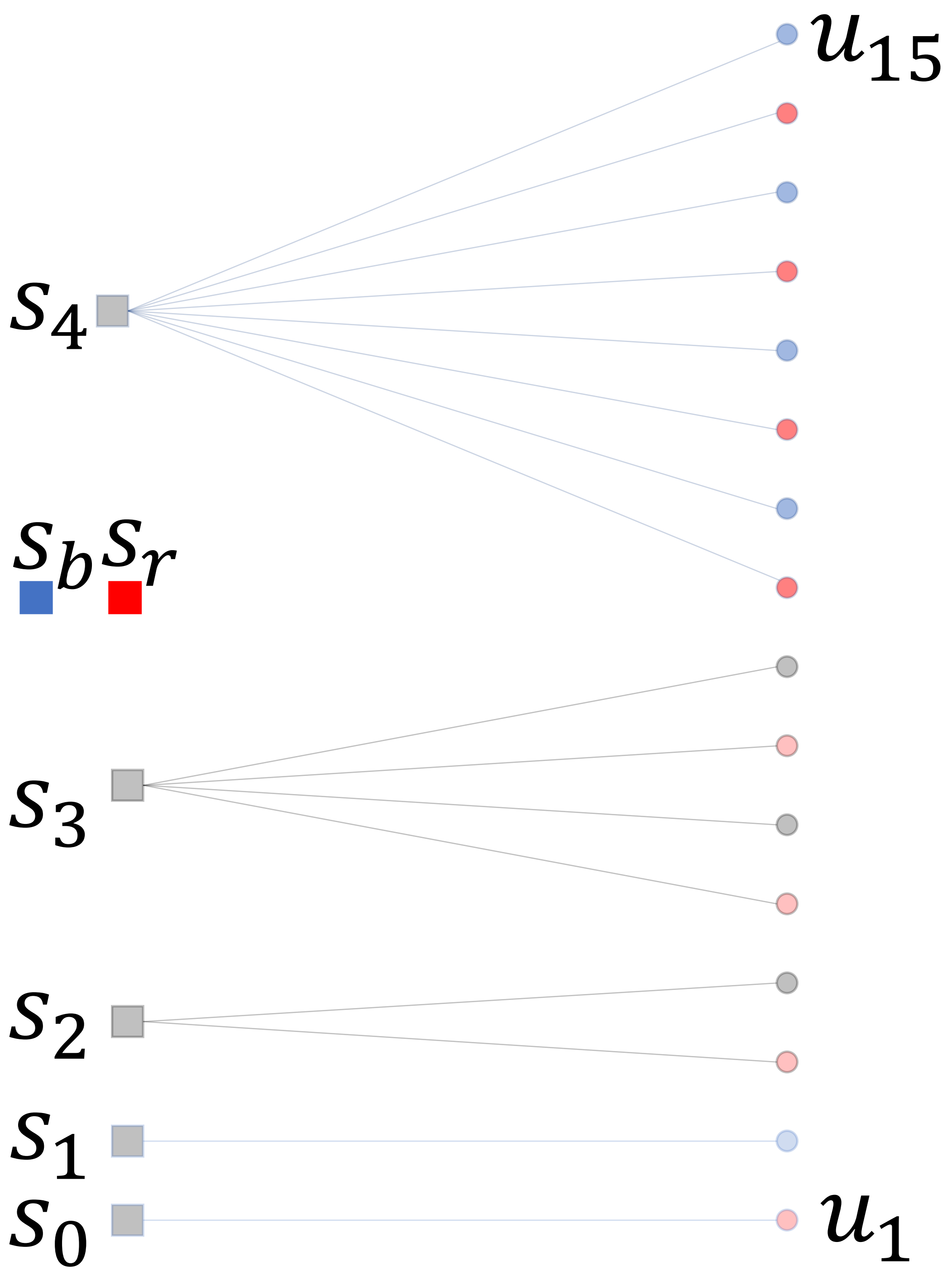
First greedy choice is s_4 (equally good as s_b and s_r)

Greedy Algorithm for Set Cover

Universe $u_1, \dots u_n$

Set s_i covers elements $u_{2^{i-1}}, \dots, u_{2^i-1}$

Special sets s_b and s_r cover odd and even elements, respectively



Optimal solution is two sets s_b and s_r

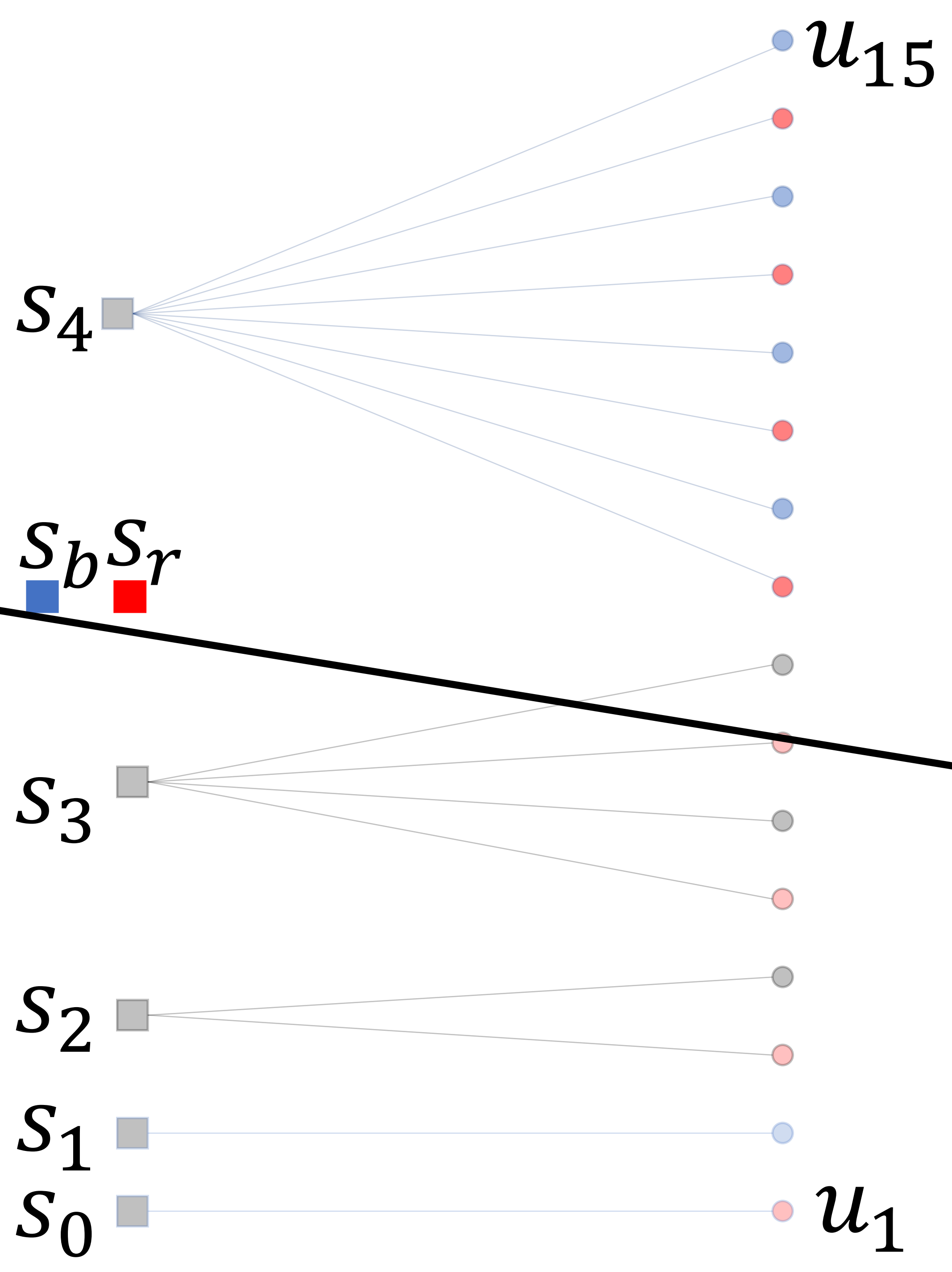
All elements covered with sets $s_1, \dots, s^{\log n - 1}$
Greedy picks these sets.

Greedy Algorithm for Set Cover

Universe $u_1, \ldots u_n$

Set s_i covers elements $u_{2^{i-1}}, \ldots, u_{2^i - 1}$

Special sets s_b and s_r cover odd and even elements, respectively



Optimal solution is two sets s_b and s_r

All elements covered with sets $s_1, \ldots, s^{\log n - 1}$
Greedy picks these sets.

Proof Recap

We created an input instance where:
1) **Optimal solution** is two sets

2) **Greedy algorithm** selects roughly $\log n$ sets

It is impossible to prove that the greedy algorithm always gives a better approximation that $O(\log n)$.

Proof Recap

We created an input instance where:
1) **Optimal solution** is two sets

2) **Greedy algorithm** selects roughly $\log n$ sets

It is impossible to prove that the greedy algorithm always gives a better approximation that $O(\log n)$.

Infinite family of graphs:
This construction works for arbitrarily large n and hence, we have an infinite family of ”bad” input graphs.

Proof Recap

We created an input instance where:
1) **Optimal solution** is two sets

2) **Greedy algorithm** selects roughly $\log n$ sets

It is impossible to prove that the greedy algorithm always gives a better approximation that $O(\log n)$.

Infinite family of graphs:
This construction works for arbitrarily large n and hence, we have an infinite family of ”bad” input graphs.

Greedy Algorithm - Runtime

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$

Crucial:
Naïve approach of browsing through the whole node set requires $O(n)$ time in every while-loop.

Heap:
By using a heap priority queue, you need one update operation per edge when the edge is removed.

Greedy Algorithm - Runtime

Input graph $G = (S \cup U, E)$
Cover $C = \emptyset$
While($U \neq \emptyset$**)**
    Find highest degree node $s \in S$
    $U := U \setminus N^1(s)$
    For each($v \in S$**)**
        $N^1(v) := N^1(v) \cap U$

Crucial:
Naïve approach of browsing through the whole node set requires $O(n)$ time in every while-loop.

Heap:
By using a heap priority queue, you need one update operation per edge when the edge is removed.

Runtime
Linear search: $O(n^2)$
Heap: $O(n \log n)$

Wrap-up

We introduced the set cover problem.

The greedy algorithm is an $O(\log n)$-approximation algorithm for set cover.

The analysis is tight!

With the APX-hardness result, this is the best we could hope for.