

CS4234

Optimiz(s)ation Algorithms

L3a – Min-Set-Cover

No VisuAlgo page yet

Any taker?

Admins

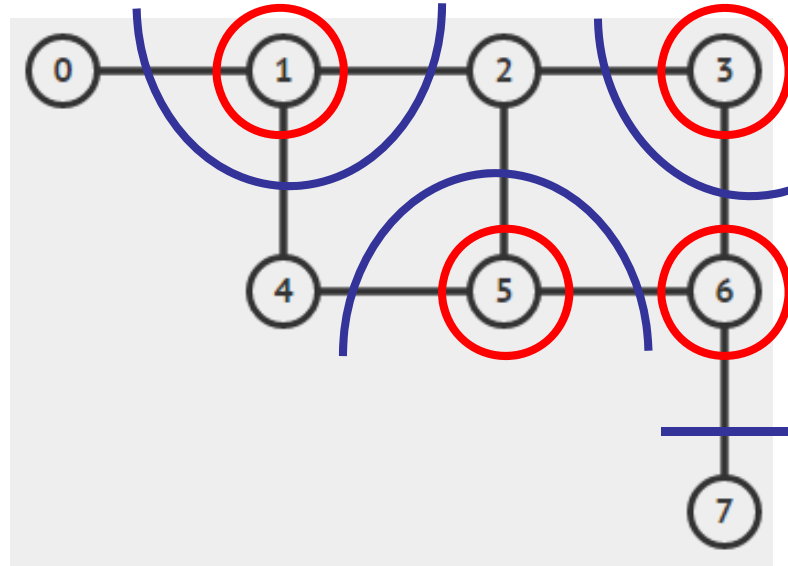
- Official class size for S1 AY2021/22 is 31
 - Running average of class size thus decreases a bit
 - From 35 pax in the previous 5 AYs (before this AY)
 - To 34 pax in the last 6 AYs (including this AY)
- “Detailed” T01-ans.pdf are in LumiNUS files
 - password-protected, not for public consumption!!
 - We repeat this cycle 10 more times until T11
- 31/31 PS1 + top 10/31 of PS2 have been fully graded
 - PS1 C++ ‘modal answer’ are in our Discord channel
 - **AGAIN: DO NOT SHARE/SUBMIT VERBATIM!!**
 - I am faster than IOI 2020-affected last S1 AY2020/21 😊
 - We repeat this cycle 4 more times (PS2+3+4+Mini Project)

MIN-SET-COVER (one more COP)

Combinatorial Optimization Problem

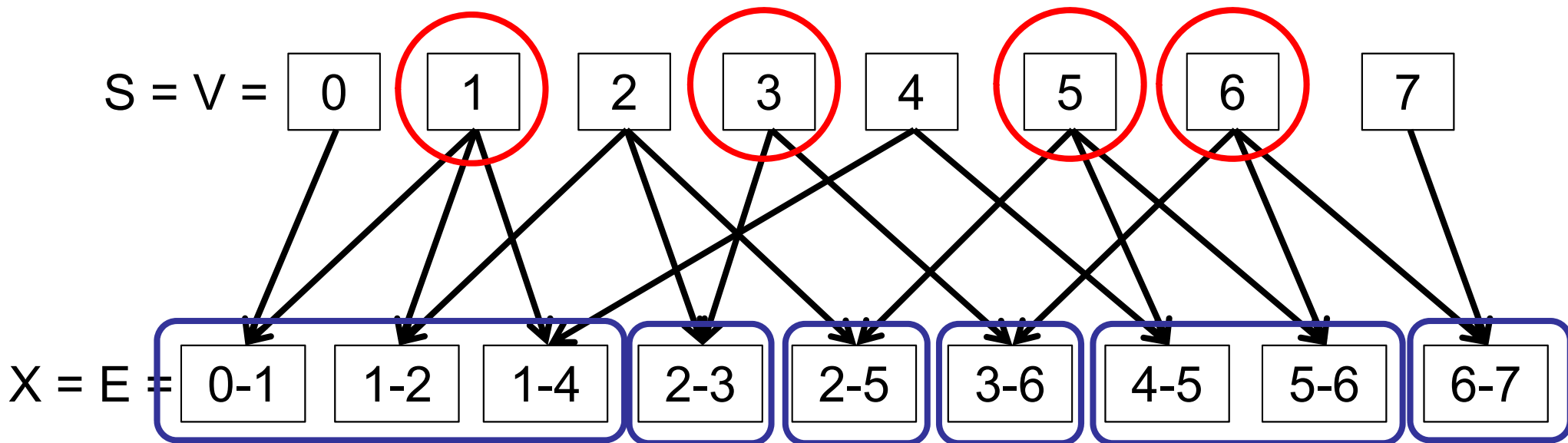
- Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of n elements
- Let S_1, S_2, \dots, S_m be subsets of X , i.e., each $S_j \subseteq X$
 - Assume that every item in X appears in some set, i.e., $\cup_j S_j = X$
- A **set cover** of X with S is a set $I \subseteq \{1, 2, \dots, m\}$ such that $\cup_{j \in I} S_j = X$ Notice... $\exists 2^m$ possible such subsets
- The solution for **MIN-SET-COVER** problem is a **set cover I of minimum size**

MIN-SET-COVER Example 1



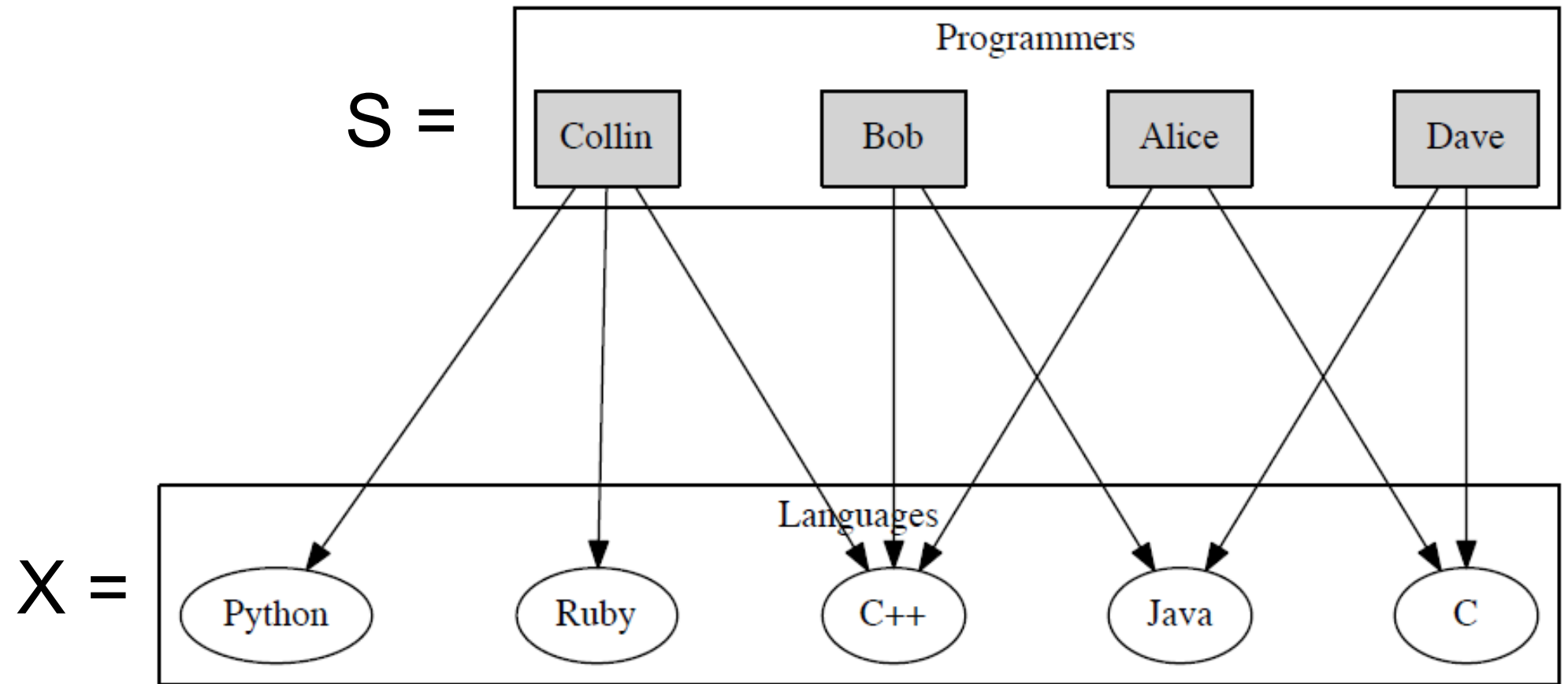
$VC \leq_p SC \dots$, so
both are NP-hard

$|VC| = |SC| = 4$ in
this example



MIN-SET-COVER Example 2

Cost Saving in Software Company
What is the optimal solution?



MIN-SET-COVER Example 3

Can you state Steven's problem for this sem's CS4234 tutorial group issues into an MSC problem?

3 groups

Group	Students	Timetable
T1 (CS4234)	12	Session 1 - Monday, Time: 11:00 - 12:00, Venue: SR_LT19, Recurrence: 13 (Steven Halim)
T2 (CS4234)	9	Session 1 - Monday, Time: 14:00 - 15:00, Venue: SR_LT19, Recurrence: 13 (Steven Halim)
T3 (CS4234)	10	Session 1 - Monday, Time: 17:00 - 18:00, Venue: SR_LT19, Recurrence: 13 (Steven Halim)

Review the recording for NUS students 😊

Now a “Doable” Task?

- <https://nus.kattis.com/problems/socialadvertising>

Review the recording for NUS students 😊

Semi-live coding (C++)

Steven will attempt to live code from here

Without compiling :O..., as he is not sure what devtools are available in LT19 desktop PC

Hopefully AC

GreedySetCover – A Greedy Algorithm

```
/* This algorithm adds sets greedily, one at a time, until everything is covered. At each step, the algorithm chooses the next set that will cover the most uncovered elements. */
```

```
1 Algorithm: GreedySetCover( $X, S_1, S_2, \dots, S_m$ )
```

```
2 Procedure:
```

```
3  $I \leftarrow \emptyset$ 
```

```
/* Repeat until every element in  $X$  is covered: */
```

```
4 while  $X \neq \emptyset$  do
```

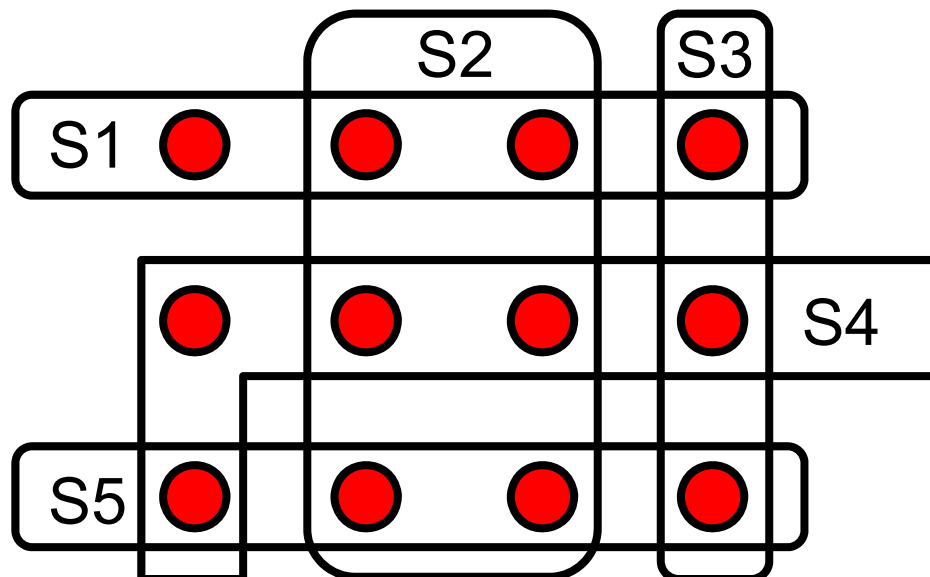
```
5   Let  $d(j) = |S_j \cap X|$  // This is the number of uncovered elements in  $S_j$ 
```

```
6   Let  $j = \operatorname{argmax}_{i \in \{1, 2, \dots, m\}} d(i)$  // Break ties by taking lower  $i$ 
```

```
7    $I \leftarrow I \cup \{j\}$  // Include set  $S_j$  into the set cover
```

```
8    $X \leftarrow X \setminus S_j$  // Remove elements in  $S_j$  from  $X$ .
```

```
9 return  $I$ 
```



GreedySetCover Execution (1)

```
/* This algorithm adds sets greedily, one at a time, until everything is covered. At each step, the algorithm chooses the next set that will cover the most uncovered elements. */
```

```
1 Algorithm: GreedySetCover( $X, S_1, S_2, \dots, S_m$ )
```

```
2 Procedure:
```

```
3  $I \leftarrow \emptyset$ 
```

```
/* Repeat until every element in  $X$  is covered: */
```

```
4 while  $X \neq \emptyset$  do
```

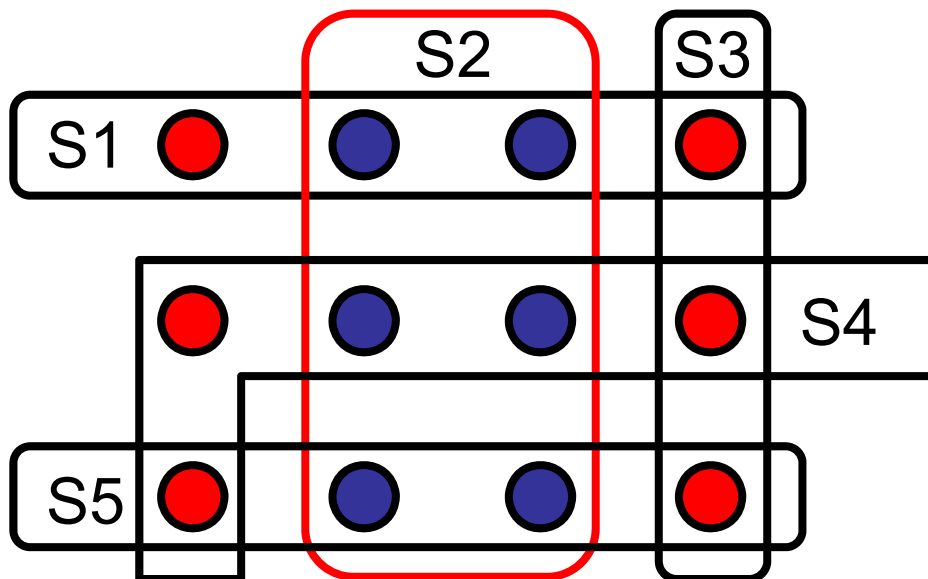
```
5 Let  $d(j) = |S_j \cap X|$  // This is the number of uncovered elements in  $S_j$ 
```

```
6 Let  $j = \operatorname{argmax}_{i \in \{1, 2, \dots, m\}} d(i)$  // Break ties by taking lower  $i$ 
```

```
7  $I \leftarrow I \cup \{j\}$  // Include set  $S_j$  into the set cover
```

```
8  $X \leftarrow X \setminus S_j$  // Remove elements in  $S_j$  from  $X$ .
```

```
9 return  $I$ 
```



Set	$d(j) - 1$				
S1	4				
S2	6				
S3	3				
S4	5				
S5	4				

GreedySetCover Execution (2)

```
/* This algorithm adds sets greedily, one at a time, until everything is covered. At each step, the algorithm chooses the next set that will cover the most uncovered elements. */
```

```
1 Algorithm: GreedySetCover( $X, S_1, S_2, \dots, S_m$ )
```

```
2 Procedure:
```

```
3  $I \leftarrow \emptyset$ 
```

```
/* Repeat until every element in  $X$  is covered: */
```

```
4 while  $X \neq \emptyset$  do
```

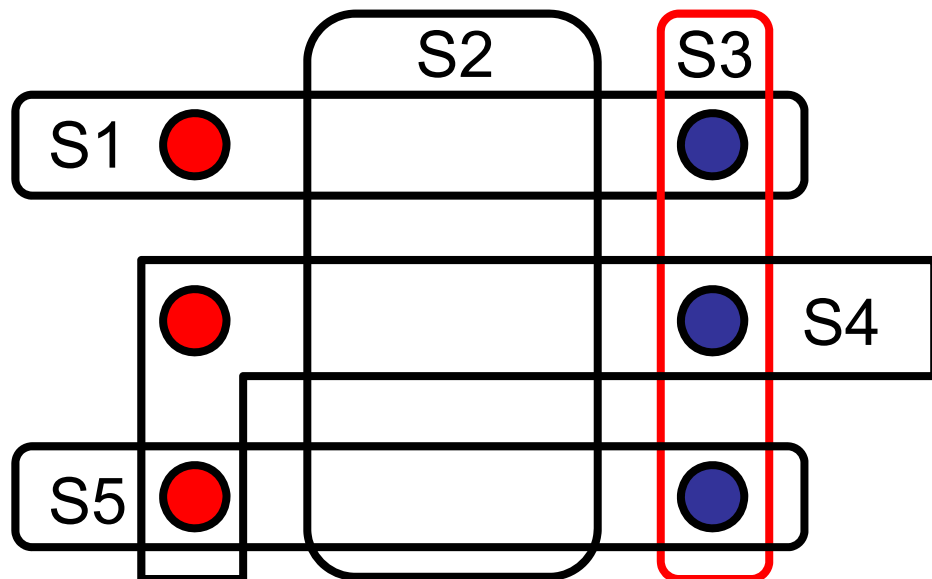
```
5 Let  $d(j) = |S_j \cap X|$  // This is the number of uncovered elements in  $S_j$ 
```

```
6 Let  $j = \operatorname{argmax}_{i \in \{1, 2, \dots, m\}} d(i)$  // Break ties by taking lower  $i$ 
```

```
7  $I \leftarrow I \cup \{j\}$  // Include set  $S_j$  into the set cover
```

```
8  $X \leftarrow X \setminus S_j$  // Remove elements in  $S_j$  from  $X$ .
```

```
9 return  $I$ 
```



Set	$d(j) - 1$	$d(j) - 2$			
S1	4	2			
S2	6				
S3	3	3			
S4	5	3			
S5	4	2			

GreedySetCover Execution (3)

```
/* This algorithm adds sets greedily, one at a time, until everything is covered. At each step, the algorithm chooses the next set that will cover the most uncovered elements. */
```

```
1 Algorithm: GreedySetCover( $X, S_1, S_2, \dots, S_m$ )
```

```
2 Procedure:
```

```
3  $I \leftarrow \emptyset$ 
```

```
/* Repeat until every element in  $X$  is covered: */
```

```
4 while  $X \neq \emptyset$  do
```

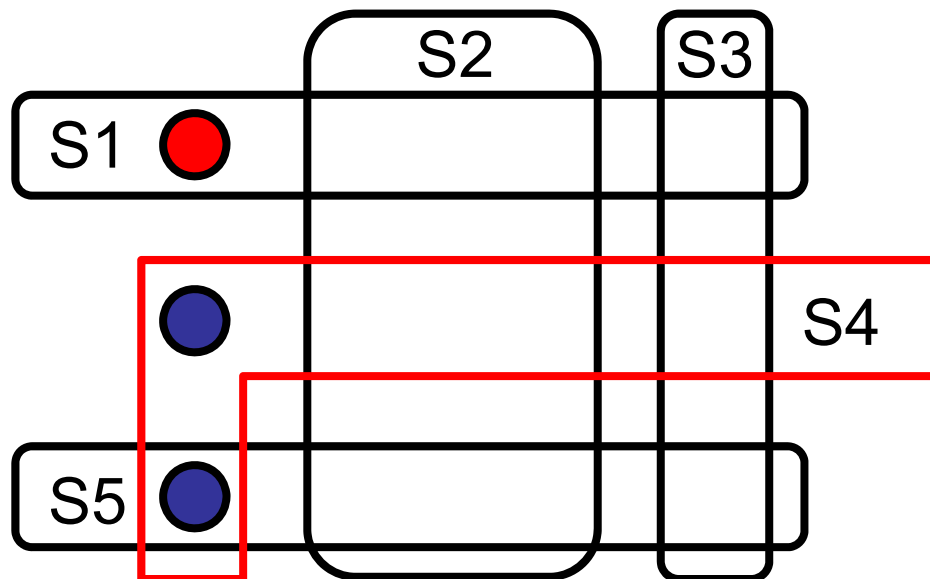
```
5 Let  $d(j) = |S_j \cap X|$  // This is the number of uncovered elements in  $S_j$ 
```

```
6 Let  $j = \operatorname{argmax}_{i \in \{1, 2, \dots, m\}} d(i)$  // Break ties by taking lower  $i$ 
```

```
7  $I \leftarrow I \cup \{j\}$  // Include set  $S_j$  into the set cover
```

```
8  $X \leftarrow X \setminus S_j$  // Remove elements in  $S_j$  from  $X$ .
```

```
9 return  $I$ 
```



Set	$d(j) - 1$	$d(j) - 2$	$d(j) - 3$		
S1	4	2	1		
S2	6				
S3	3	3			
S4	5	3	2		
S5	4	2	1		

GreedySetCover Execution (4)

/* This algorithm adds sets greedily, one at a time, until everything is covered. At each step, the algorithm chooses the next set that will cover the most uncovered elements. */

1 **Algorithm:** GreedySetCover(X, S_1, S_2, \dots, S_m)

2 **Procedure:**

3 $I \leftarrow \emptyset$

/* Repeat until every element in X is covered: */

4 **while** $X \neq \emptyset$ **do**

5 Let $d(j) = |S_j \cap X|$ // This is the number of uncovered elements in S_j

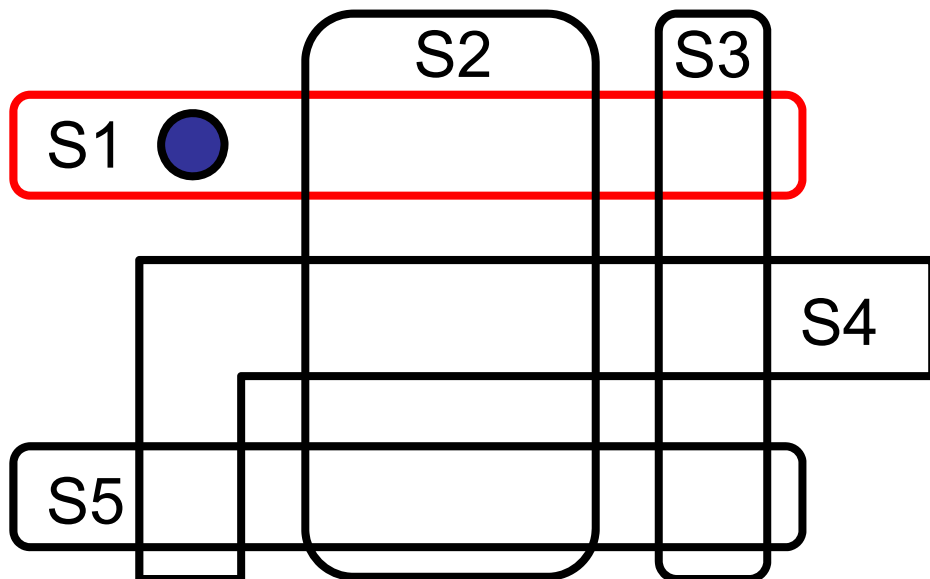
6 Let $j = \operatorname{argmax}_{i \in \{1, 2, \dots, m\}} d(i)$ // Break ties by taking lower i

7 $I \leftarrow I \cup \{j\}$ // Include set S_j into the set cover

8 $X \leftarrow X \setminus S_j$ // Remove elements in S_j from X .

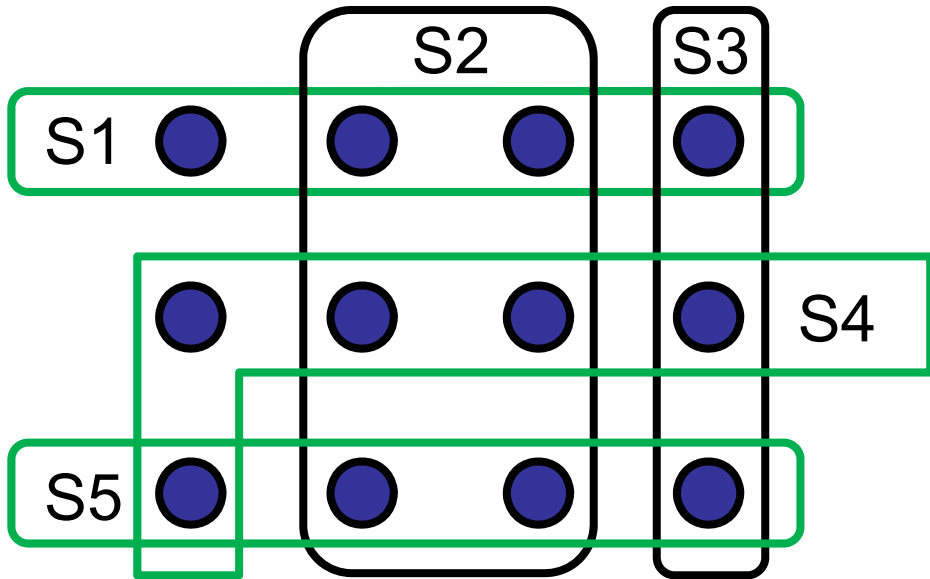
9 **return** I

Time complexity: $O(m(n + \log m))$,
greedy with max PQ with key = $(d(i), i)$,
the PQ must also supports update key

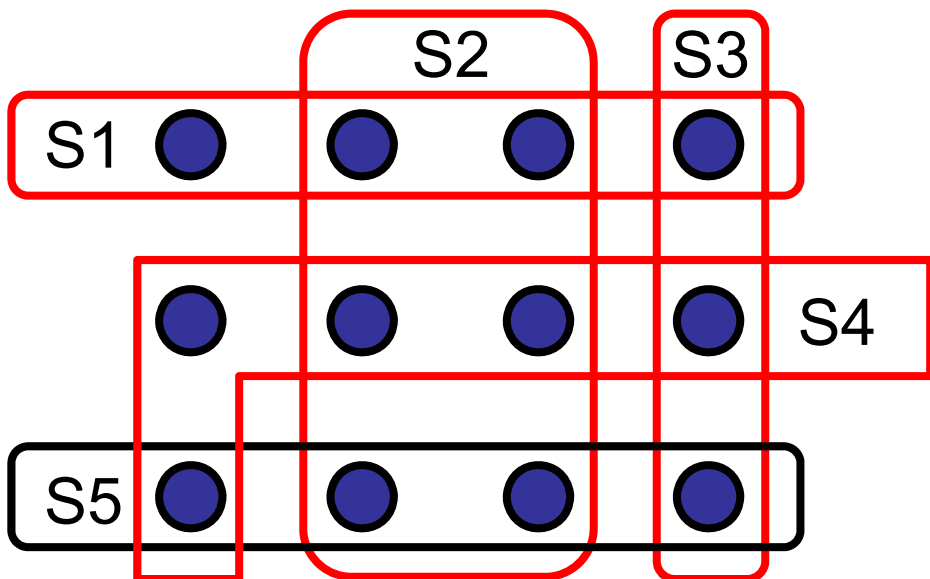


Set	$d(j) - 1$	$d(j) - 2$	$d(j) - 3$	$d(j) - 4$	
S1	4	2	1	1	
S2	6				
S3	3	3			
S4	5	3	2		
S5	4	2	1	0	

GreedySetCover vs Optimal



{S1, S4, S5} is the optimal (minimum) Min-Set-Cover solution for this instance



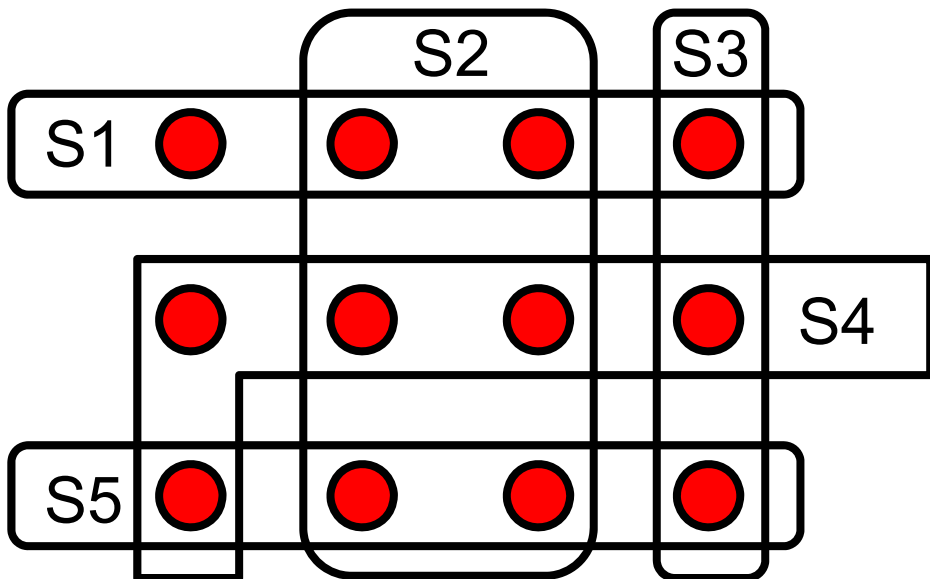
Set	$d(j) - 1$	$d(j) - 2$	$d(j) - 3$	$d(j) - 4$	
S1	4	2	1	1	
S2	6				
S3	3	3			
S4	5	3	2		
S5	4	2	1	0	

GreedySetCover – Analysis (1)

Initially, there are 12 elements

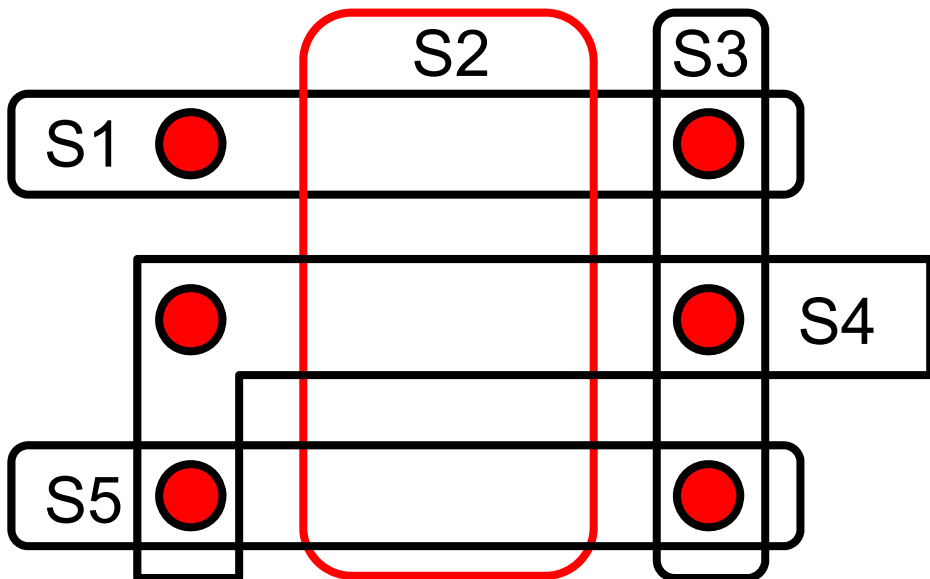
None of the sets cover more than 6 elements

Any solution for this instance of the `Min-Set-Cover` problem requires at least $12/6 = 2$ sets



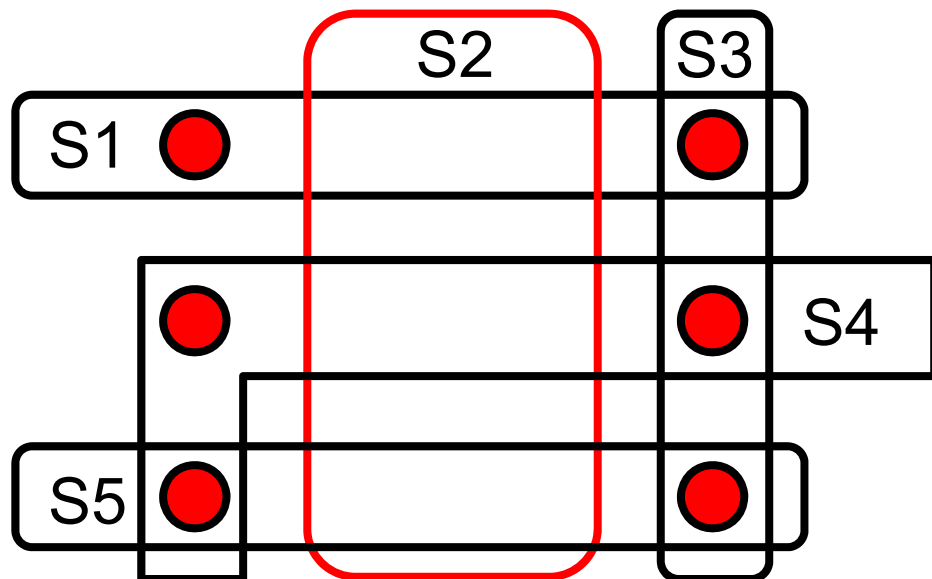
GreedySetCover – Analysis (2)

After S2 is selected greedily, there are 6 elements left
None of the available sets cover more than 3 elements
Any solution for this instance of the `Min-Set-Cover` problem now requires at least $6/3 = 2$ (more) sets



GreedySetCover – Analysis (3)

In general, if there are k elements left and
None of the available sets cover more than t elements
Any solution for this instance of the `Min-Set-Cover`
problem now requires at least k/t (more) sets



GreedySetCover – Analysis (4)

When we run the algorithm, let us label the elements in the order that they are covered.

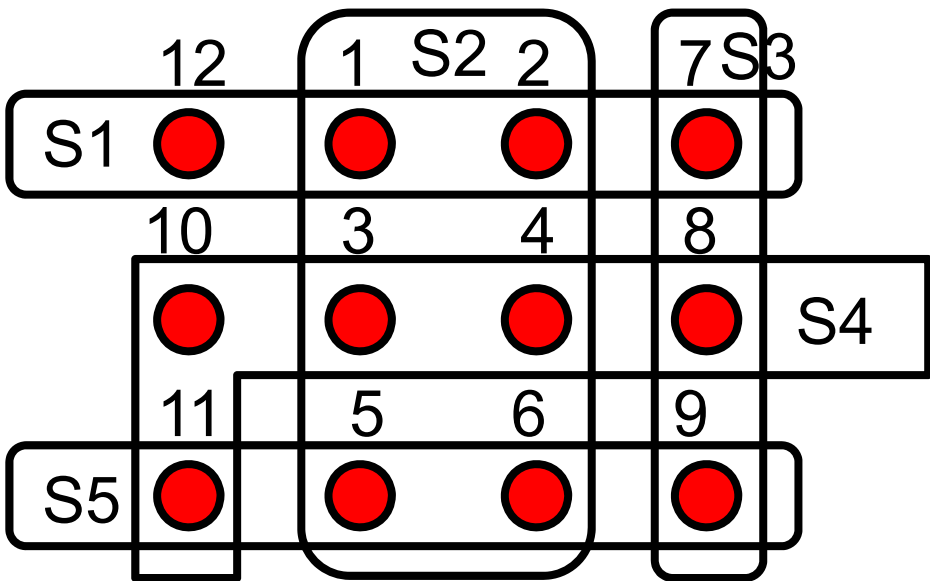
$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}$

S_2 S_3 S_4 S_1

← These are the first sets that cover these elements

For each element x_j , let c_j be the number of elements covered at the same time. In the example, this would yield:

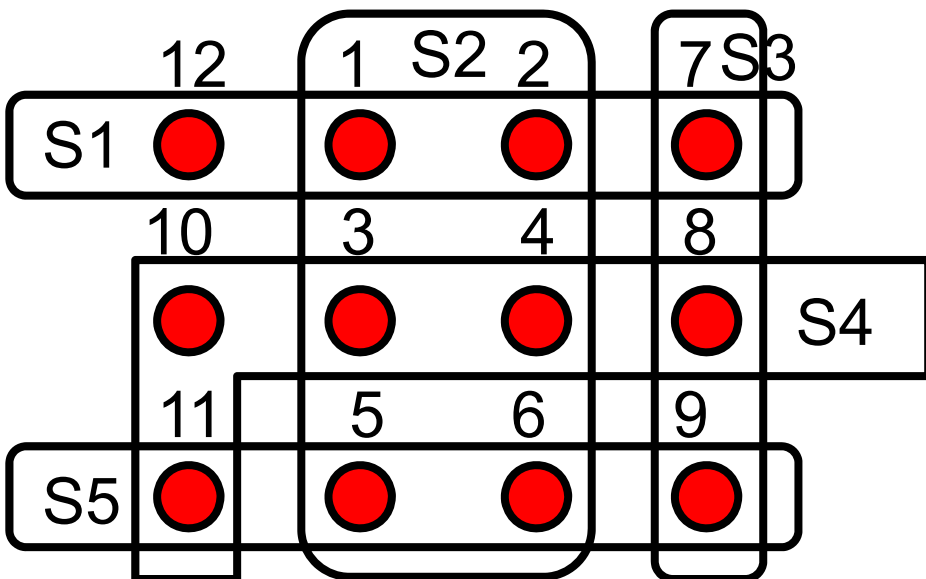
$$c_1 = 6, c_2 = 6, c_3 = 6, c_4 = 6, c_5 = 6, c_6 = 6, c_7 = 3, c_8 = 3, c_9 = 3, c_{10} = 2, c_{11} = 2, c_{12} = 1$$



GreedySetCover – Analysis (5)

We define $cost(x_j) = 1/c_j$. In this way, the cost of covering all the new elements for some set is exactly 1. In this example, the cost of covering $x_1, x_2, x_3, x_4, x_5, x_6$ is 1, the cost of covering x_7, x_8, x_9 is 1, etc. In general, if I is the set cover constructed by the Greedy Algorithm, then:

$$|I| = \sum_{j=1}^n cost(x_j) .$$



$$cost(x_1) = 1/6$$

...

$$cost(x_6) = 1/6$$

$$cost(x_7) = 1/3$$

...

$$cost(x_9) = 1/3$$

$$cost(x_{10}) = 1/2$$

$$cost(x_{11}) = 1/2$$

$$cost(x_{12}) = 1/1$$

$$||| = 4$$

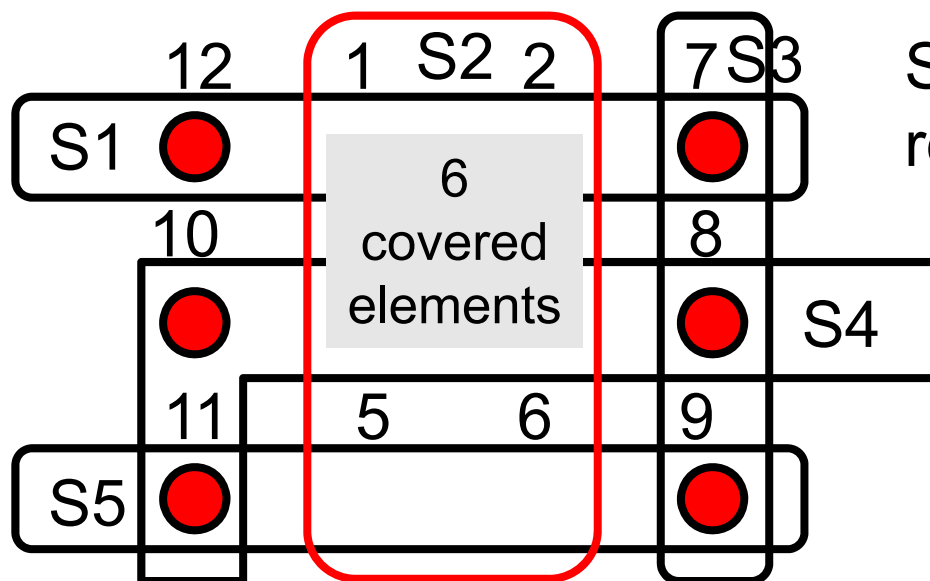
GreedySetCover – Analysis (6)

Key step. Let's consider the situation after elements x_1, x_2, \dots, x_{j-1} have already been covered, and the elements x_j, x_{j+1}, \dots, x_n remain to be covered. Let OPT be the optimal solution for covering all n elements.

What is the best that OPT can do to cover the element x_j, x_{j+1}, \dots, x_n ? How many sets does OPT need to cover these remaining elements?

Notice that there remain $n - j + 1$ uncovered elements. However, no set covers more than $c(j)$ of the remaining elements. In particular, all the sets already selected by the Greedy Algorithm cover *zero* of the remaining elements. Of the sets not yet chosen by the Greedy Algorithm, the one that covers the most remaining elements covers $c(j)$ of those elements: Otherwise, the Greedy Algorithm would have chosen a different set.

$j=7, 12-7+1 = 6$ uncovered elements



S3 covers $c(7) = 3$
remaining uncovered elements

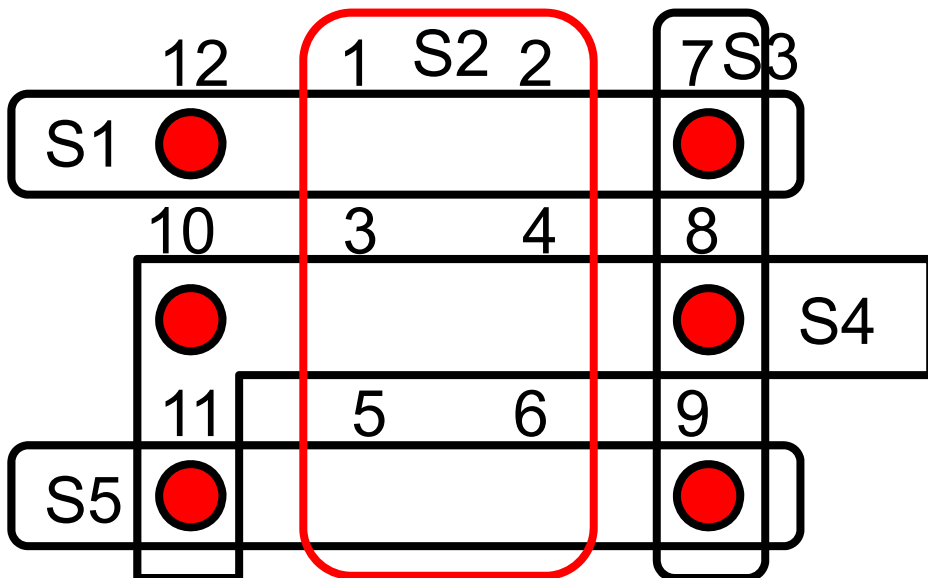
GreedySetCover – Analysis (7)

Therefore, OPT needs at least $(n - j + 1)/c(j)$ sets to cover the remaining $(n - j + 1)$ elements. We thus conclude that:

$$OPT \geq \frac{n - j + 1}{c(j)} \geq (n - j + 1) \text{cost}(x_j)$$

Or to put it differently:

$$\text{cost}(x_j) \leq \frac{OPT}{(n - j + 1)}$$



GreedySetCover – Analysis (8)

We can now show that the Greedy Algorithm provides a good approximation:

$$\begin{aligned} |I| &= \sum_{j=1}^n \text{cost}(x_j) \\ &\leq \sum_{j=1}^n \frac{OPT}{(n-j+1)} \\ &\leq OPT \sum_{i=1}^n \frac{1}{i} \\ &\leq OPT(\ln n + O(1)) \end{aligned}$$

$$\begin{aligned} &\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \dots + \frac{1}{2} + \frac{1}{1} \\ &= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n-1} + \frac{1}{n} \end{aligned}$$

(Notice that the third inequality is simply a change of variable where $i = (n - j + 1)$, and the fourth inequality is because the Harmonic series $1/1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$ can be bounded by $\ln n + O(1)$.)

We have therefore shown that the set cover constructed is at most $O(\log n)$ times optimal, i.e., the Greedy Algorithm is an $O(\log n)$ -approximation algorithm:

Anecdote: And yet we said the $O(\log n)$ Deterministic-3 approximation algorithm for MVC was ‘not good’... so let’s debate on ‘theory vs practice’



Shift the green line upwards

GreedySetCover – Challenge

Review the recording for NUS students 😊

Summary

- Yet another NP-hard COP: `Min-Set-Cover`
- Three :O Examples
- (Optimized) brute force for small instance
- `GreedySetCover` (an approximation algorithm)
- Analysis: **$O(\log n)$** -approximation algorithm
 - The proof is clever...
 - Can you do that kind of algebraic manipulation to arrive at the proof by yourself (for another analysis)?

Before I forget

Time to do attendance taking 😊

- <https://inetapps.nus.edu.sg/ctr/Home>

You need to:

- ~~Declare your temperature regularly~~ (no longer needed)
- Declare your (+family member(s)) health regularly
- Do your FET/ART test if you are scheduled for it
- Should *not* be on SHN/LOA at the moment