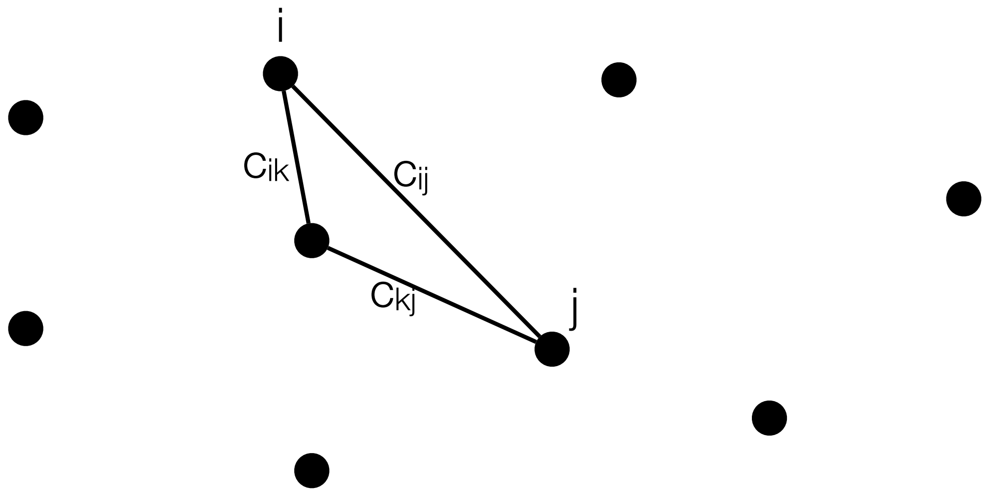
Traveling salesman problem

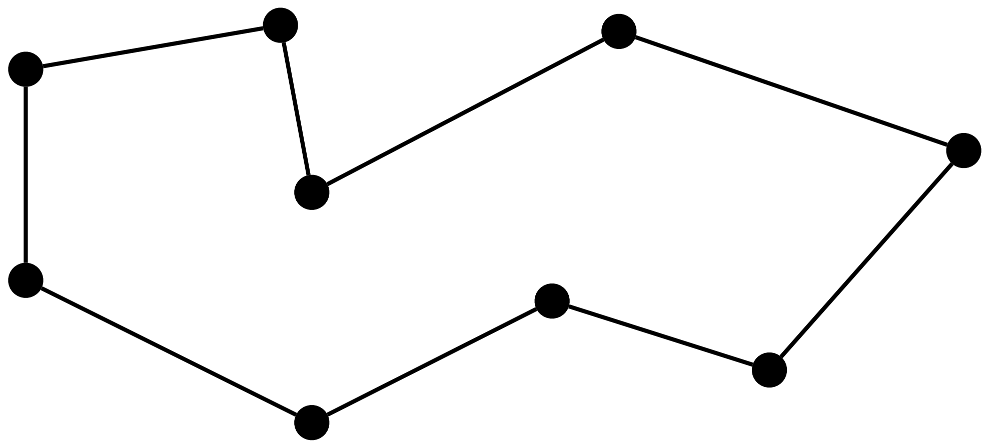
Inge Li Gørtz

Traveling Salesman Problem (TSP)



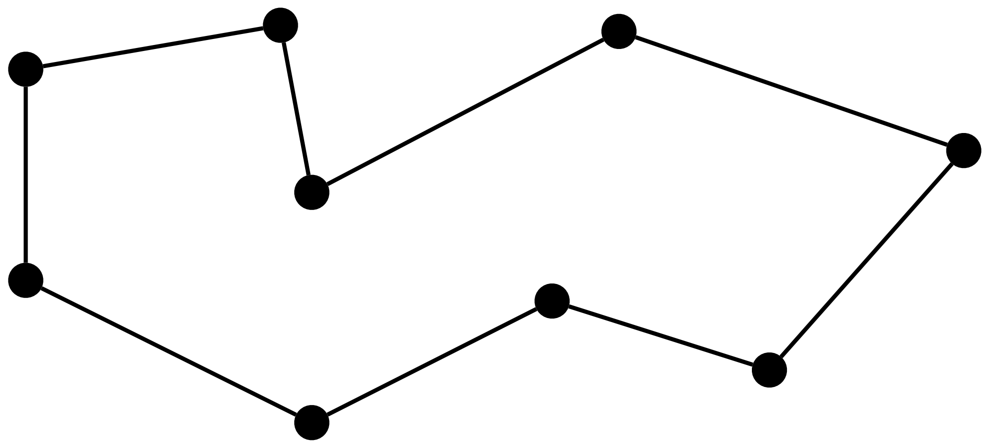
- Set of cities {1,…,n}
- $c_{ij} \geq 0$: cost of traveling from i to j.
- c_{ij} a metric:
  - $c_{ii} = 0$
  - $c_{ij} = c_{ji}$
  - $c_{ij} \leq c_{ik} + c_{kj}$ (triangle inequality)
- Goal: Find a *tour of minimum cost visiting every city exactly once*.

Traveling Salesman Problem (TSP)



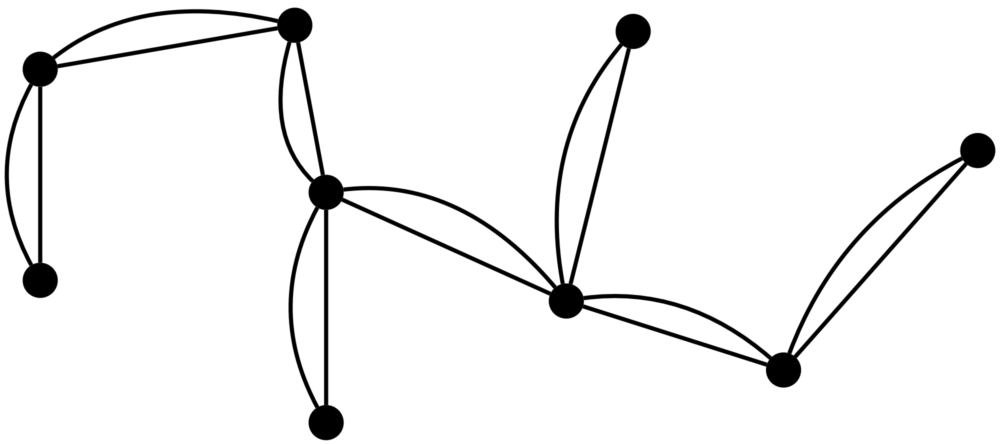
- Set of cities {1,…,n}
- $c_{ij} \geq 0$: cost of traveling from i to j.
- c_{ij} a metric:
  - $c_{ii} = 0$
  - $c_{ij} = c_{ji}$
  - $c_{ij} \leq c_{ik} + c_{kj}$
- Goal: Find a *tour of minimum cost visiting every city exactly once*.

Double tree algorithm



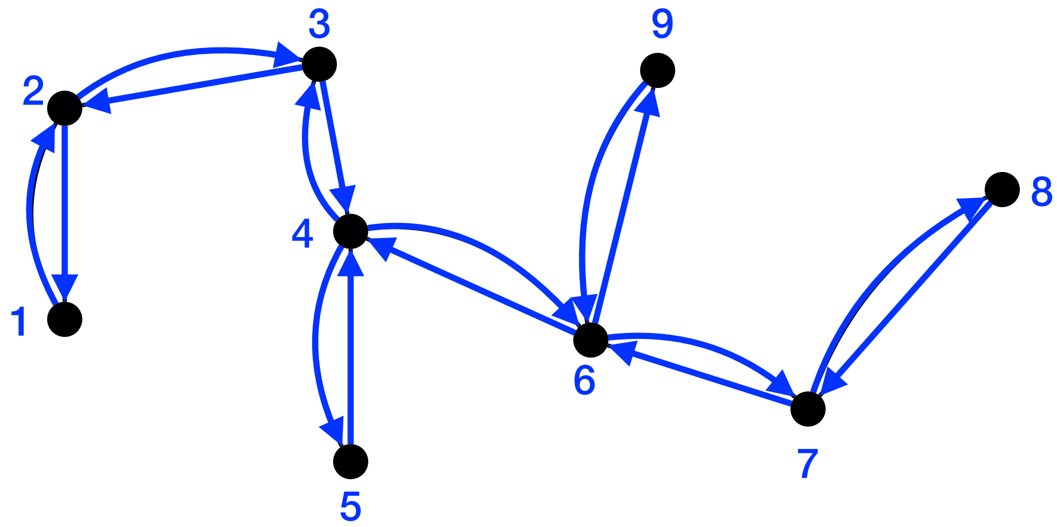
- MST is a lower bound on TSP.
  - Deleting an edge e from OPT gives a spanning tree.
  - OPT ≥ OPT - c_e ≥ MST.

Double tree algorithm



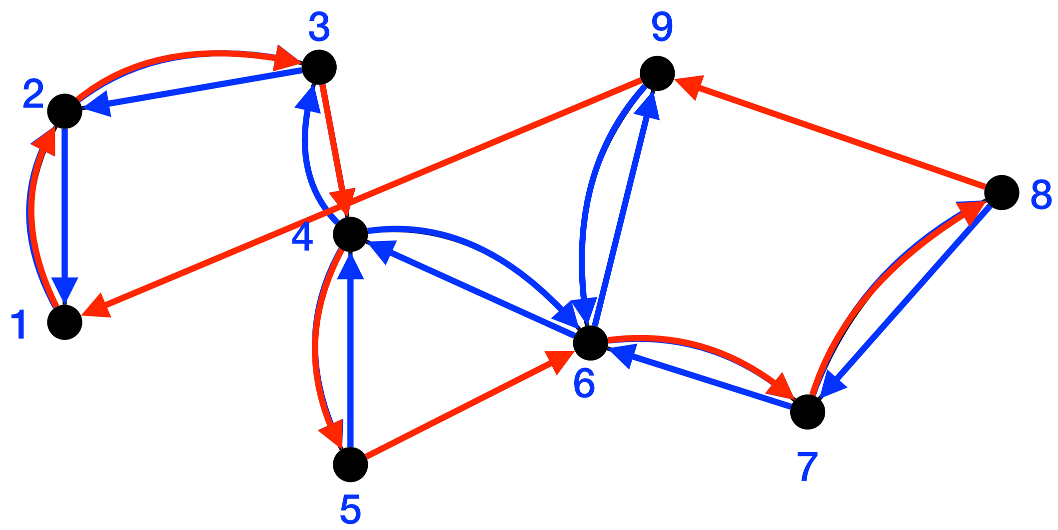
- Double tree algorithm
  - Compute MST T.
  - Double edges of T
  - Construct Euler tour **τ** (a tour visiting every edge exactly once).

Double tree algorithm



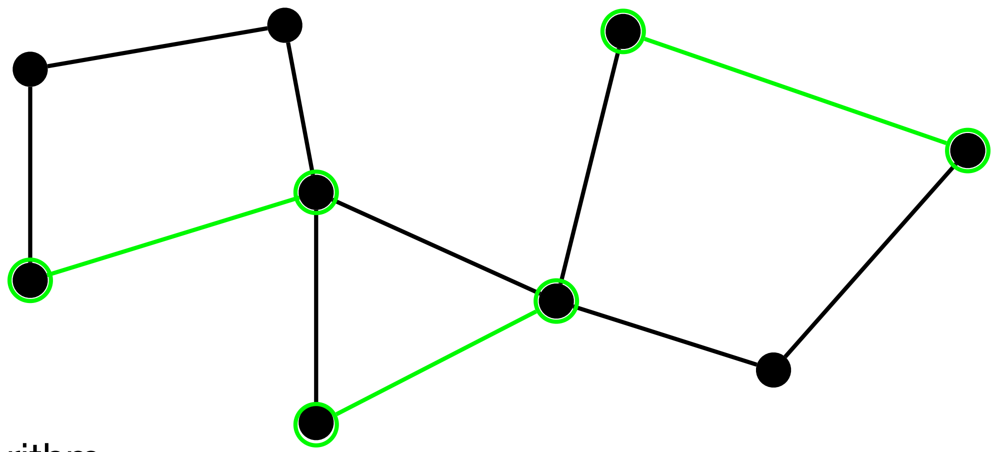
- Double tree algorithm
  - Compute MST T.
  - Double edges of T
  - Construct Euler tour **τ** (a tour visiting every edge exactly once).

Double tree algorithm



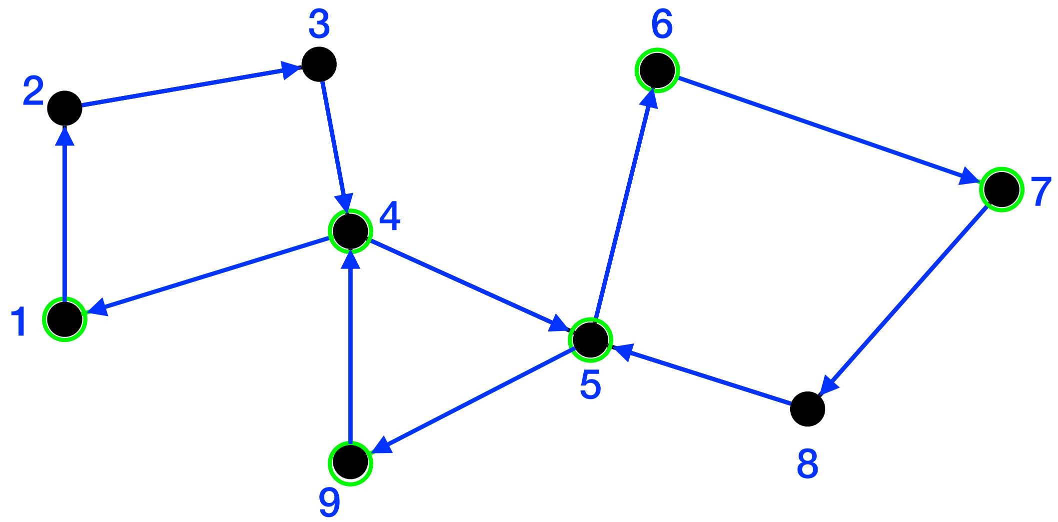
- Double tree algorithm
  - Compute MST T.
  - Double edges of T
  - Construct Euler tour τ (a tour visiting every edge exactly once).
  - Shortcut τ such that each vertex only visited once (τ')
- length(τ') ≤ length(τ) = 2 cost(T) ≤ 2 OPT.
- The double tree algorithm is a 2-approximation algorithm for TSP.

Christofides' algorithm



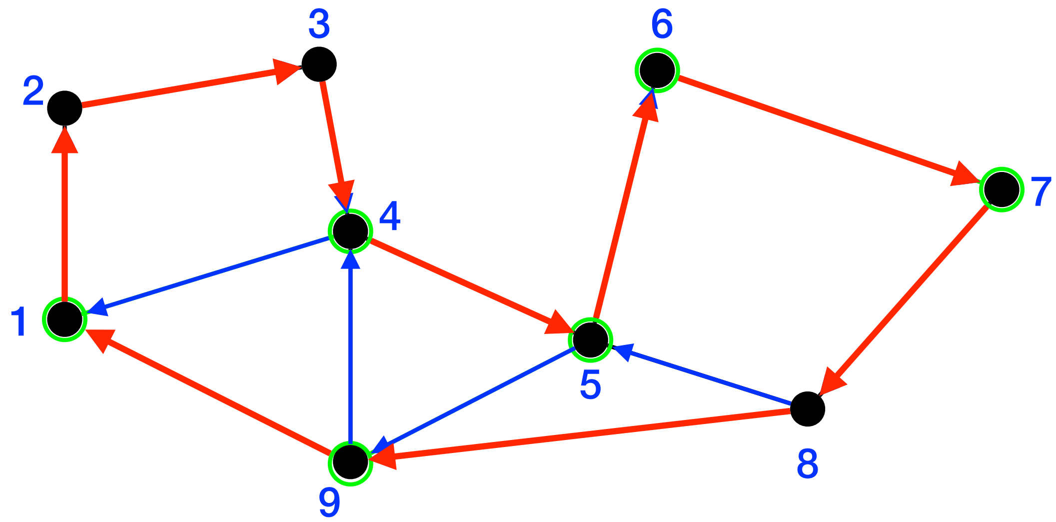
- Christofides' algorithm
  - Compute MST T.
  - No need to double all edges:
    - Enough to turn it into an Eulerian graph: *A graph Eulerian if there is a traversal of all edges visiting every edge exactly once.*
      - G Eulerian iff G connected and all nodes have even degree.
    - Consider set O of all odd degree vertices in T.
    - Find minimum cost perfect matching M on O.
      - Matching: no edges share an endpoint.
      - Perfect: all vertices matched.
      - Perfect matching on O exists: Number of odd vertices in a graph is even.
    - T + M is Eulerian (all vertices have even degree).

Christofides' algorithm



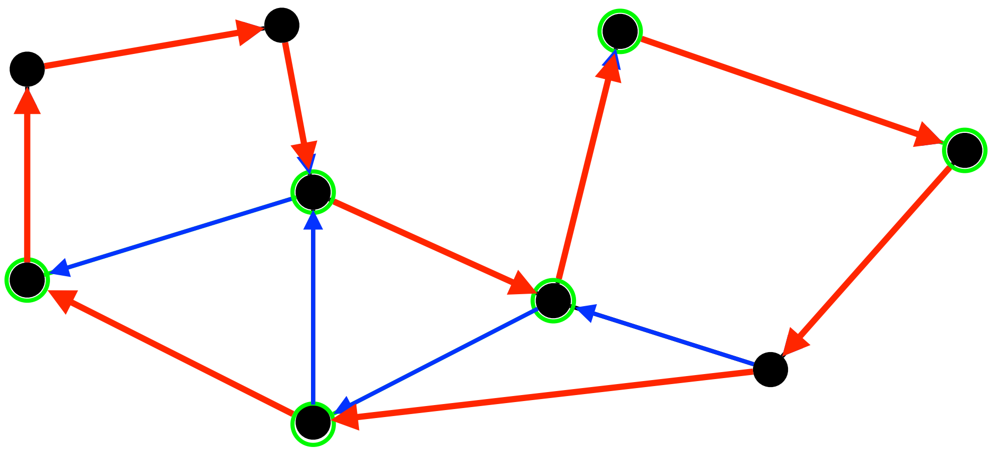
- Christofides' algorithm
  - Compute MST T.
  - O = {odd degree vertices in T}.
  - Compute minimum cost perfect matching M on O.
  - Construct Euler tour τ
  - Shortcut such that each vertex only visited once (τ')

Christofides' algorithm



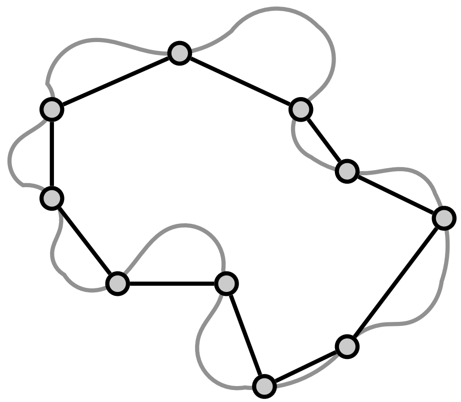
- Christofides' algorithm
    - Compute MST T.
    - O = {odd degree vertices in T}.
    - Compute minimum cost perfect matching M on O.
    - Construct Euler tour τ
    - Shortcut such that each vertex only visited once (τ')

Christofides' algorithm



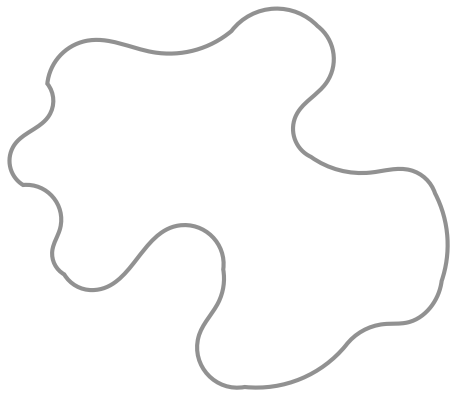
- Christofides' algorithm
  - Compute MST T.
  - O = {odd degree vertices in T}.
  - Compute minimum cost perfect matching **M** on O.
  - Construct Euler tour **τ**
  - Shortcut such that each vertex only visited once (**τ'**)
- length(τ') ≤ length(τ) = cost(T) + cost(**M**) ≤ OPT + cost(**M**).

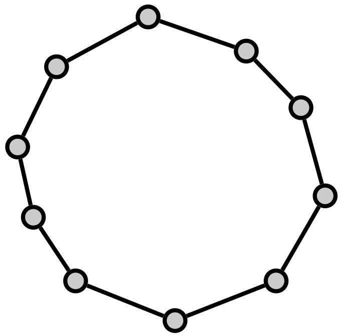
Analysis of Christofides' algorithm



- cost(M) ≤ OPT/2.
  - OPT_O = OPT restricted to O.
  - OPT_O ≤ OPT.

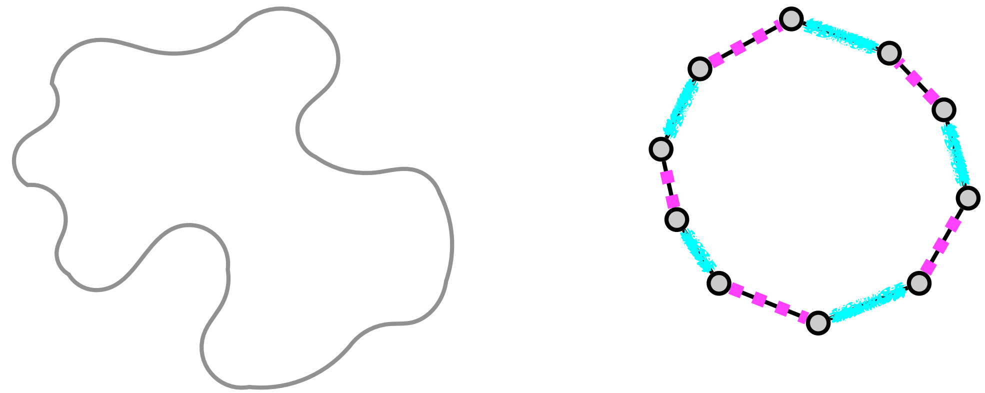
Analysis of Christofides' algorithm





- cost(M) ≤ OPT/2.
    - OPT_O = OPT restricted to O.
    - OPT_O ≤ OPT.

Analysis of Christofides' algorithm



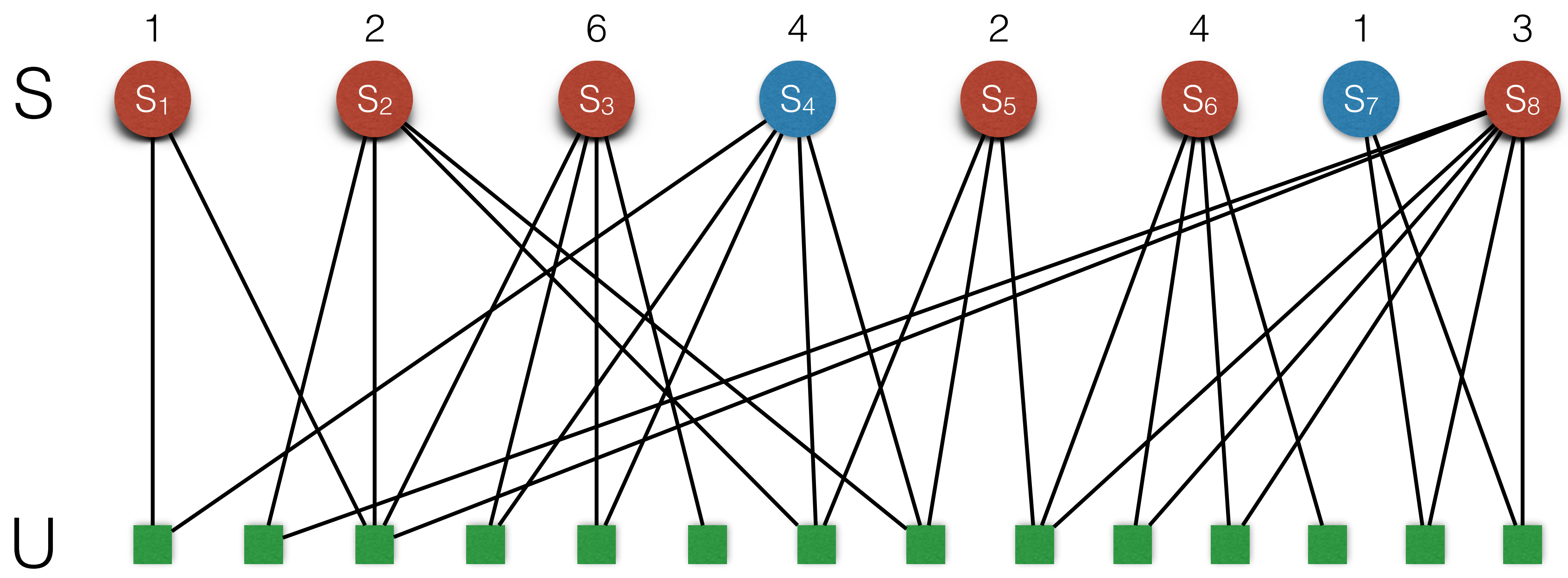
- cost(M) ≤ OPT/2:
  - OPT_O = OPT restricted to O.
  - $OPT_O \leq$ OPT.
  - can partition OPT_O into two perfect matchings O_1 and O_2.
  - cost(M) ≤ min(cost(O_1), cost(O_2)) ≤ OPT/2.
- length(τ') ≤ length(τ) = cost(T) + cost(M) ≤ OPT + OPT/2 = 3/2 OPT.
- Christofides' algorithm is a 3/2-approximation algorithm for TSP.

Set cover

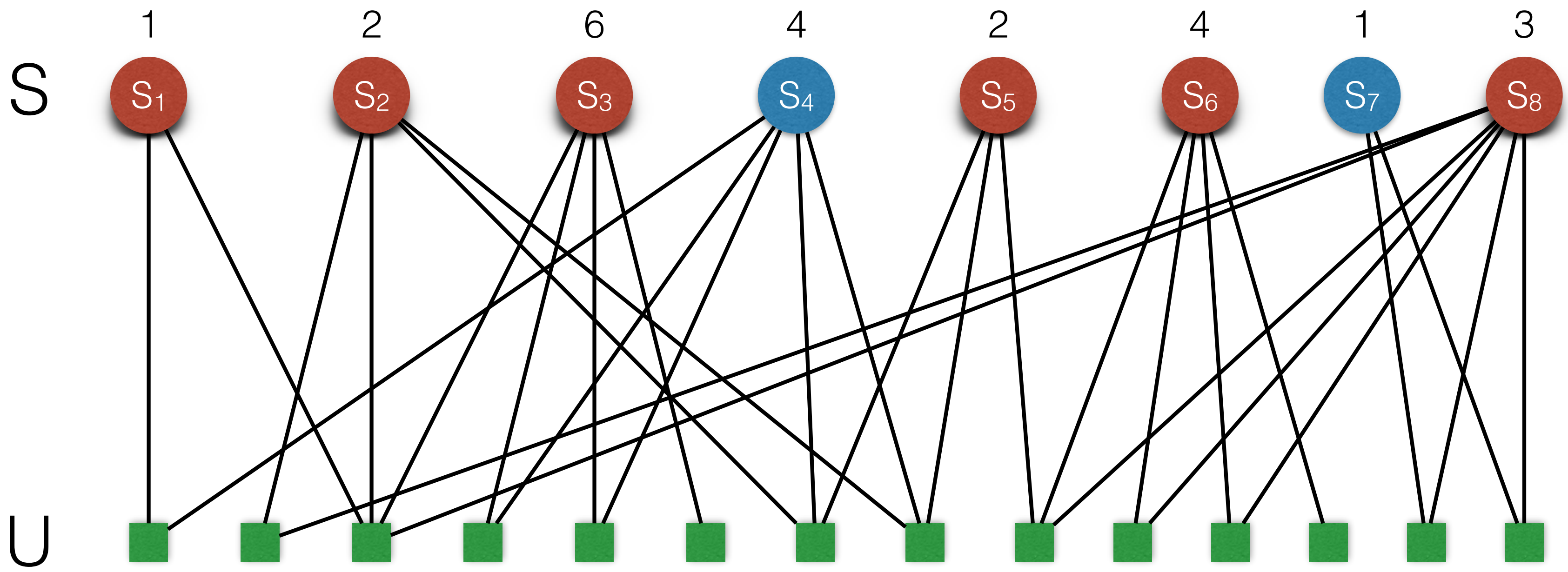
Set cover problem

- Set U of n elements.
- Subsets of U: $S_1,\ldots,S_m$.
- Each set S_i has a weight $w_i \geq 0$.
- Set cover. A collection of subsets C whose union is equal to U.
- Goal. find set cover of minimum weight.

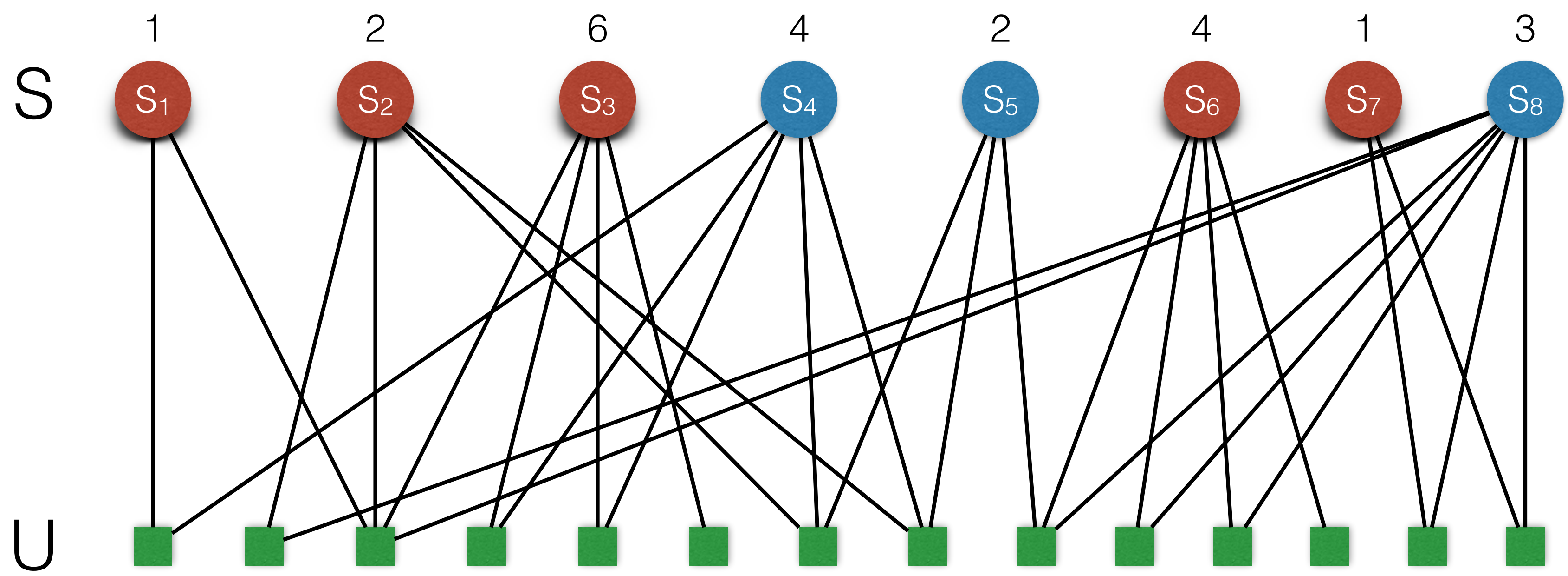
Set Cover



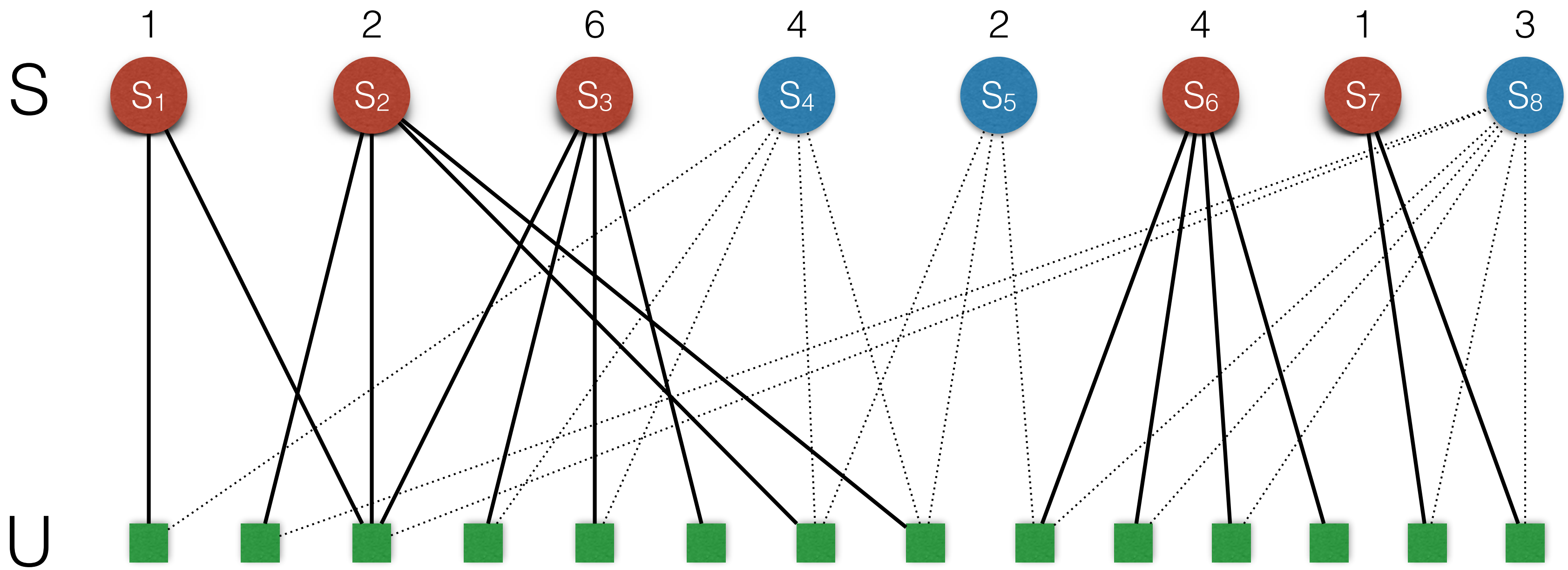
Set Cover



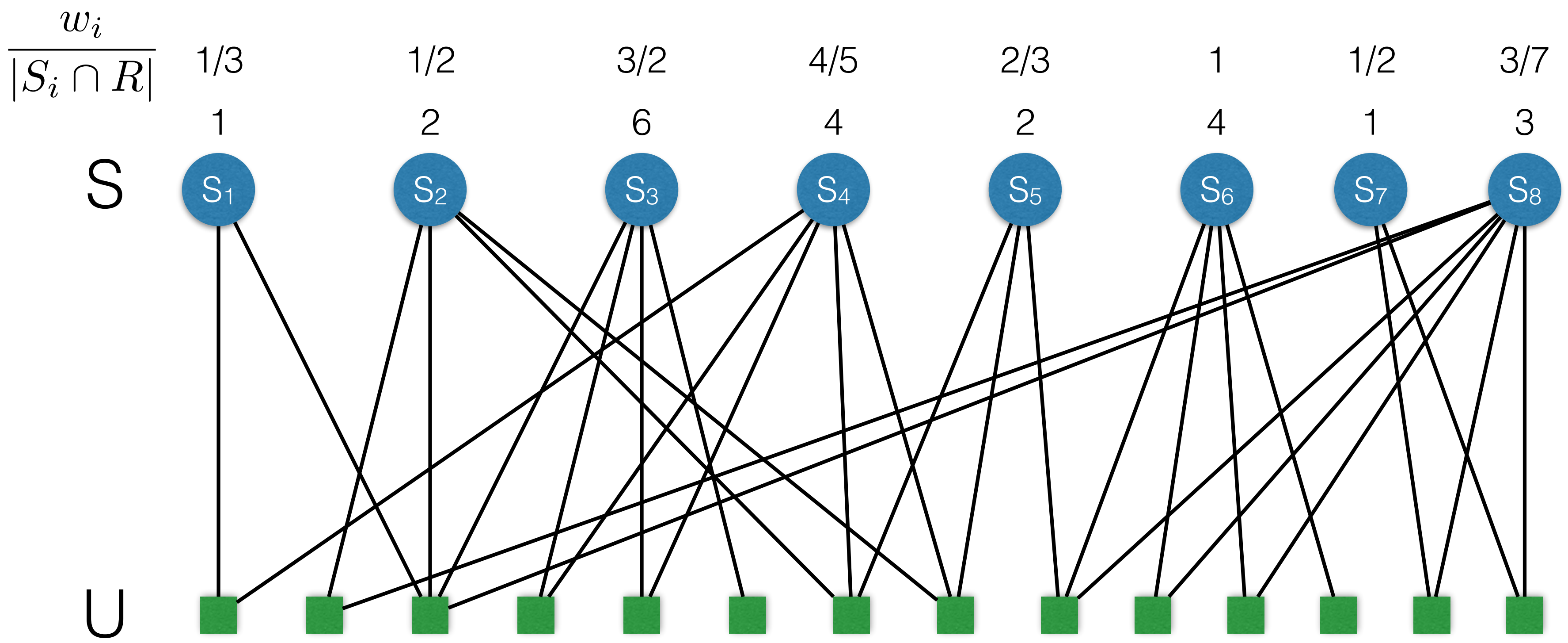
Set Cover



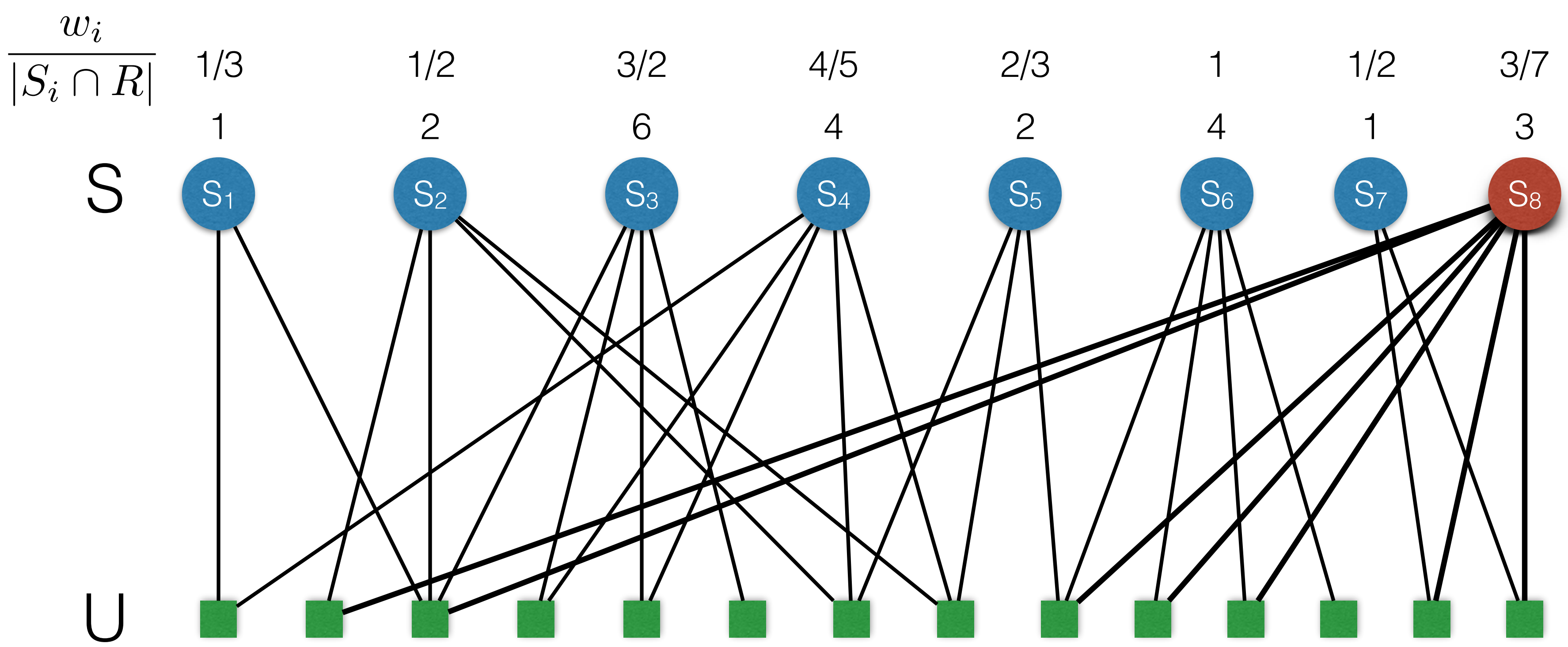
Set Cover



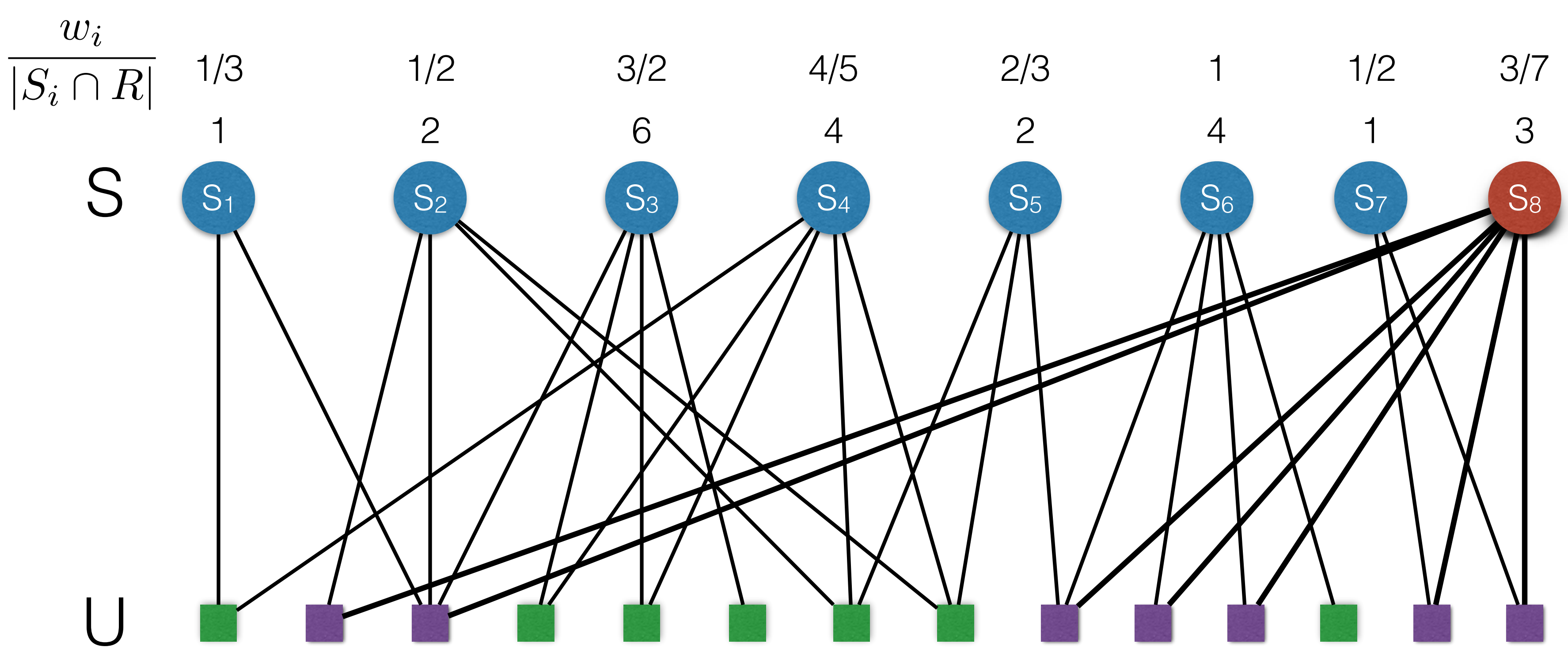
Set Cover: Greedy algorithm



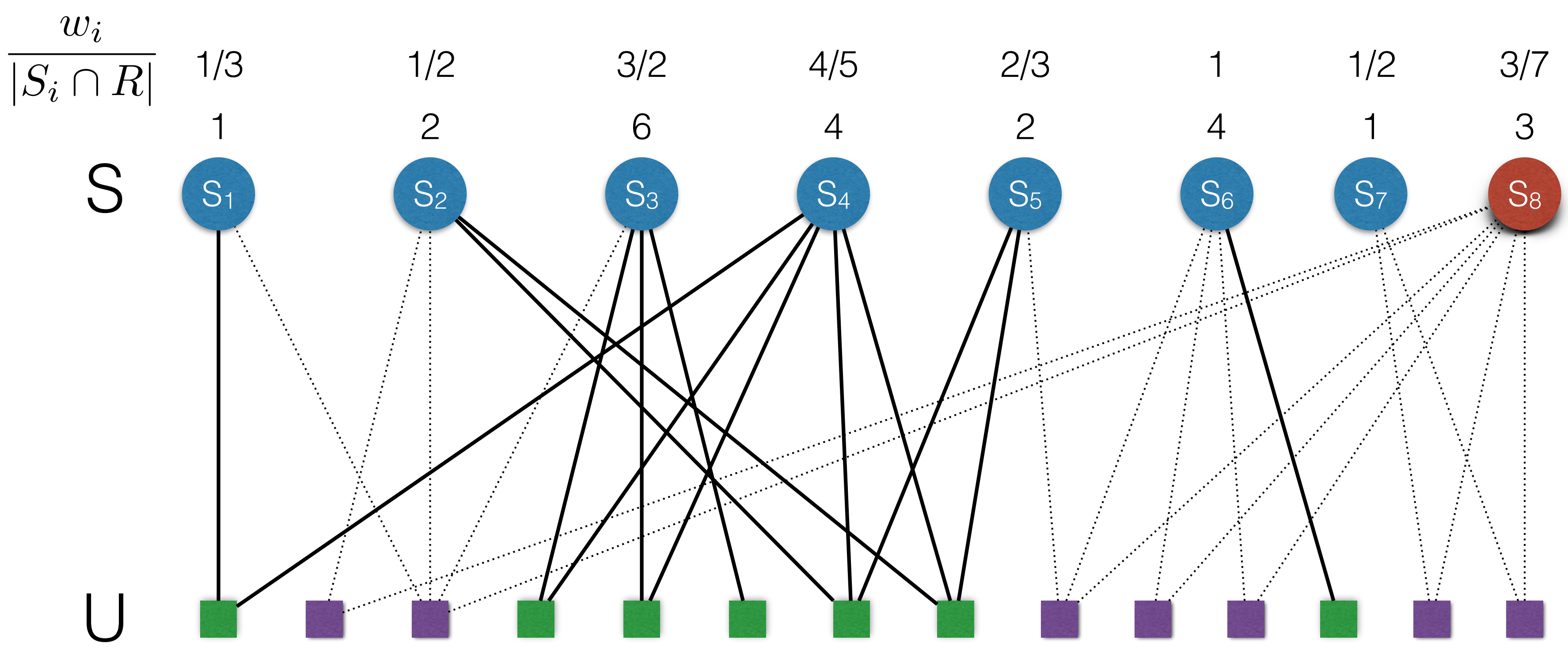
Set Cover: Greedy algorithm



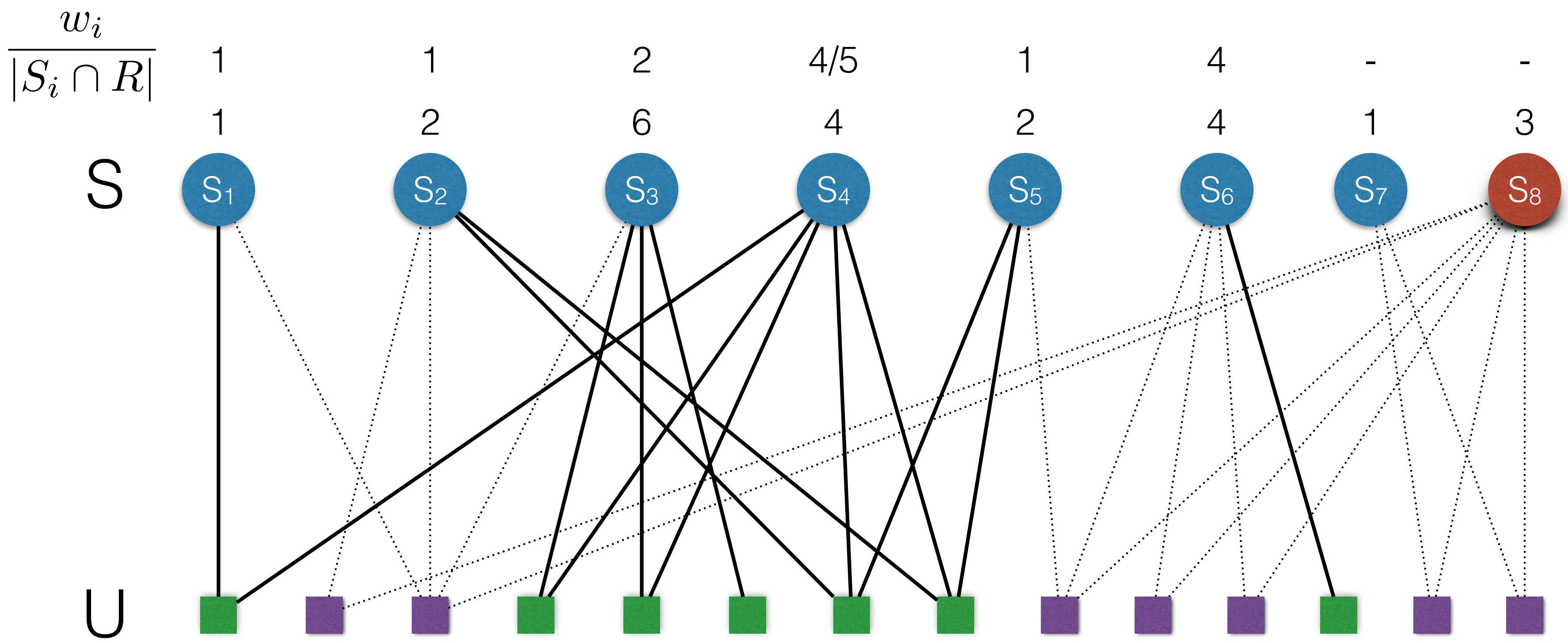
Set Cover: Greedy algorithm



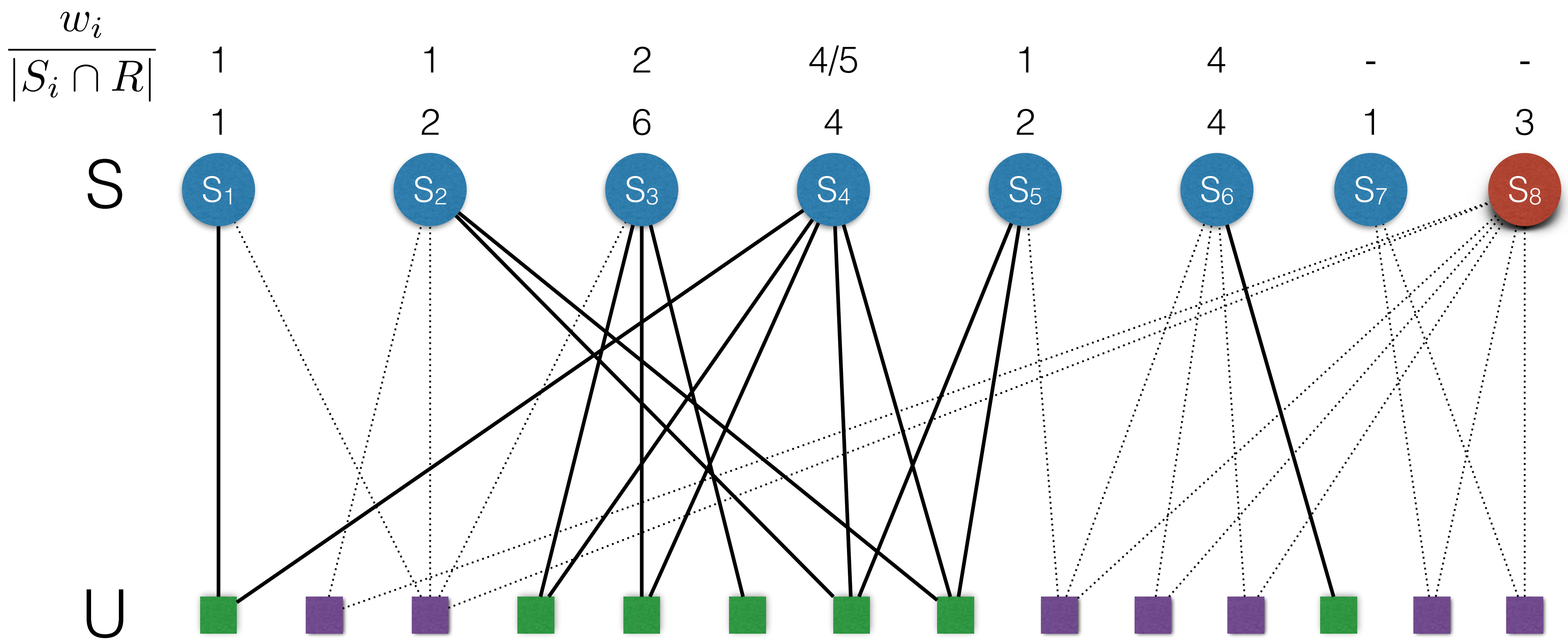
Set Cover: Greedy algorithm



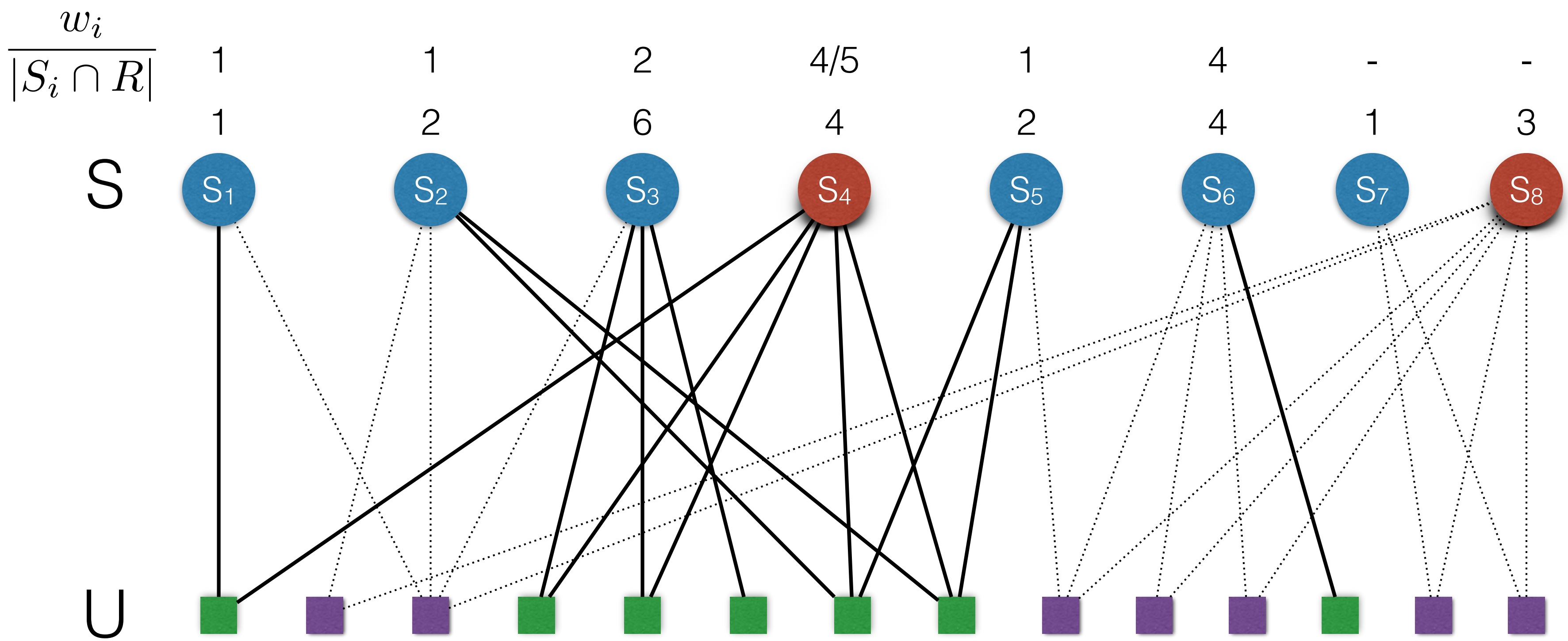
Set Cover: Greedy algorithm



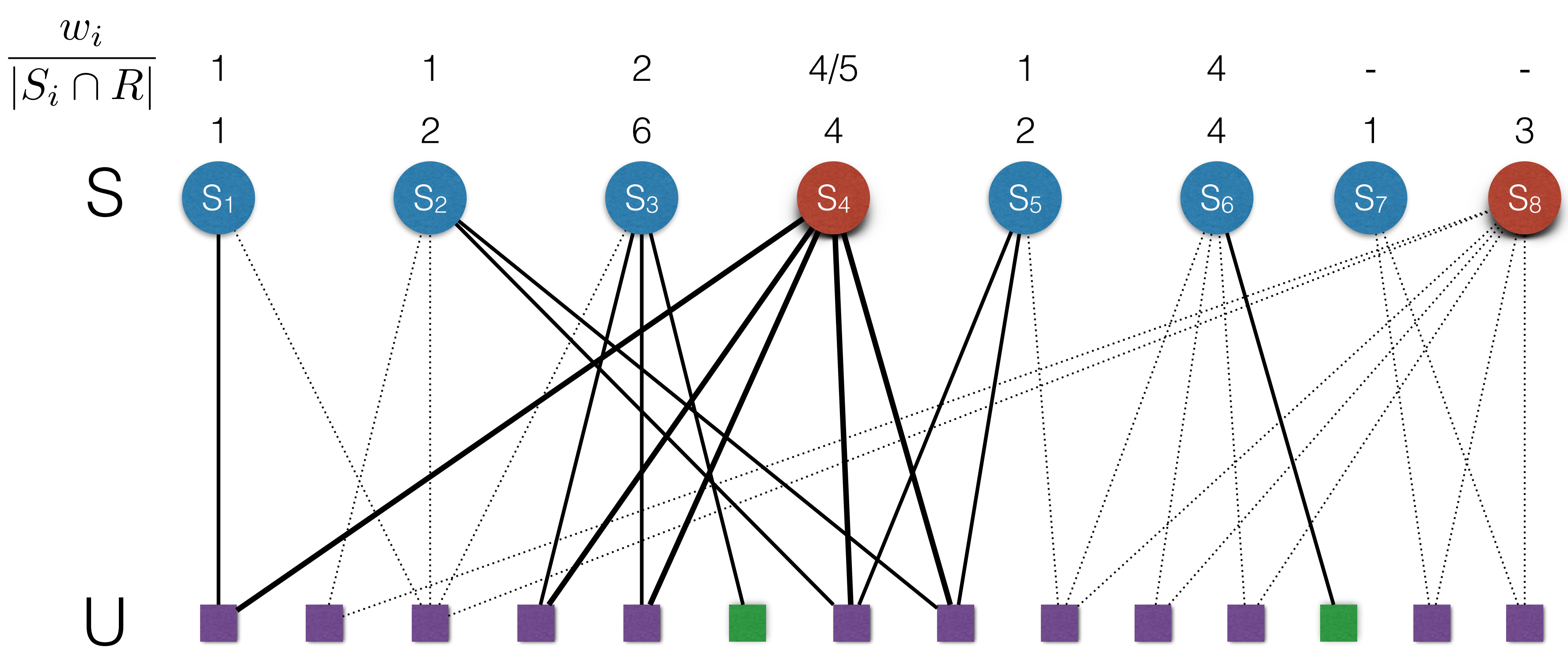
Set Cover: Greedy algorithm



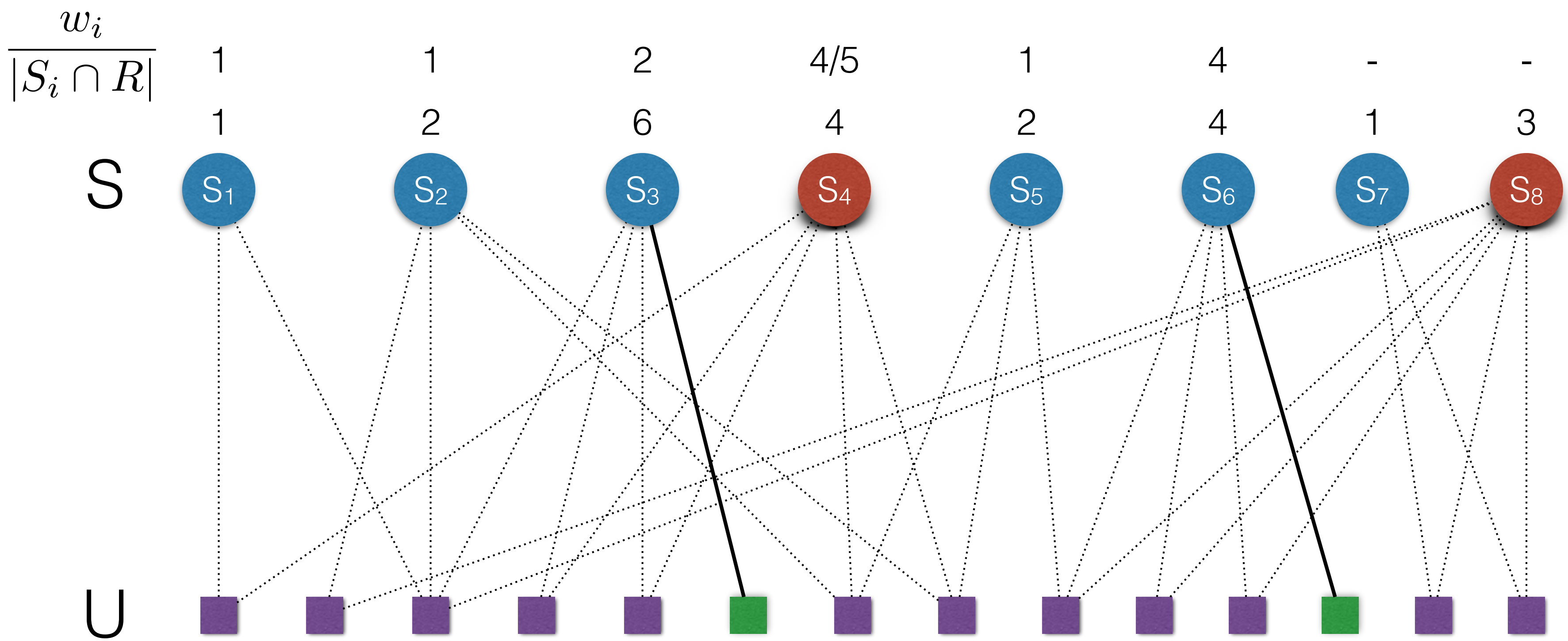
Set Cover: Greedy algorithm



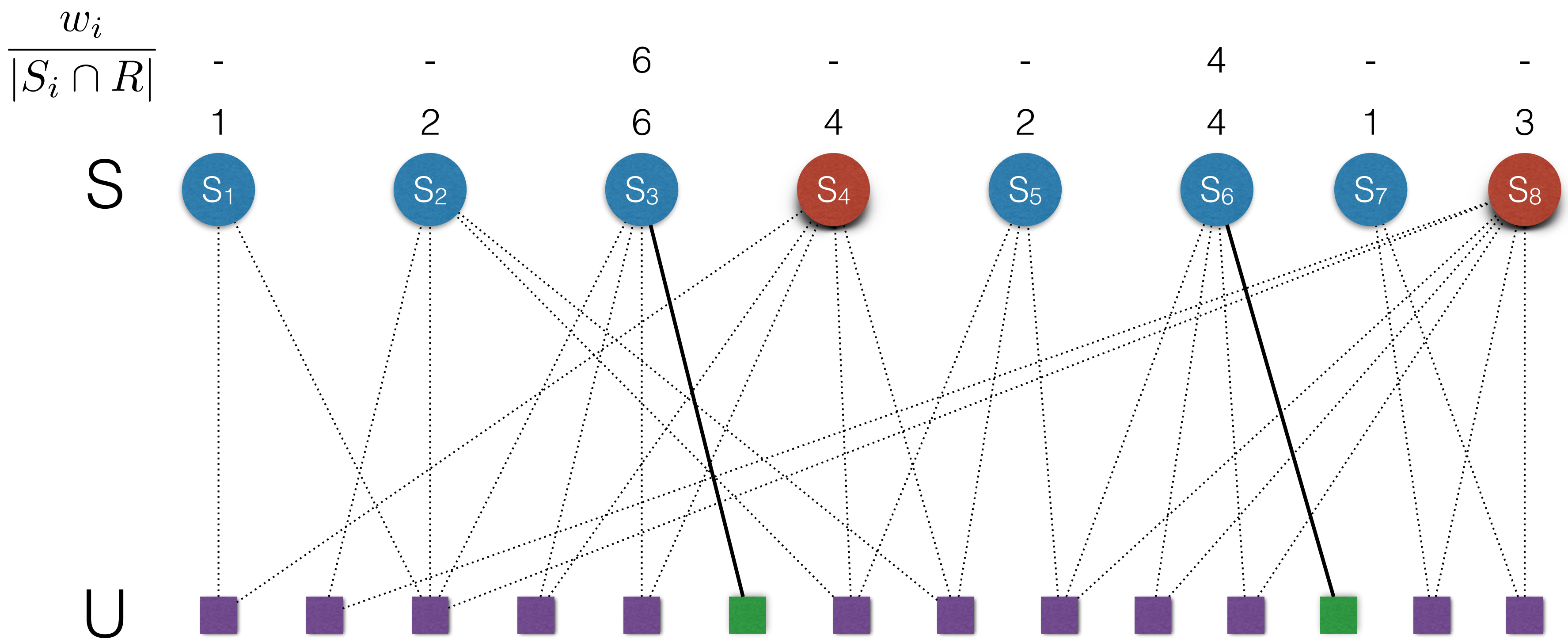
Set Cover: Greedy algorithm



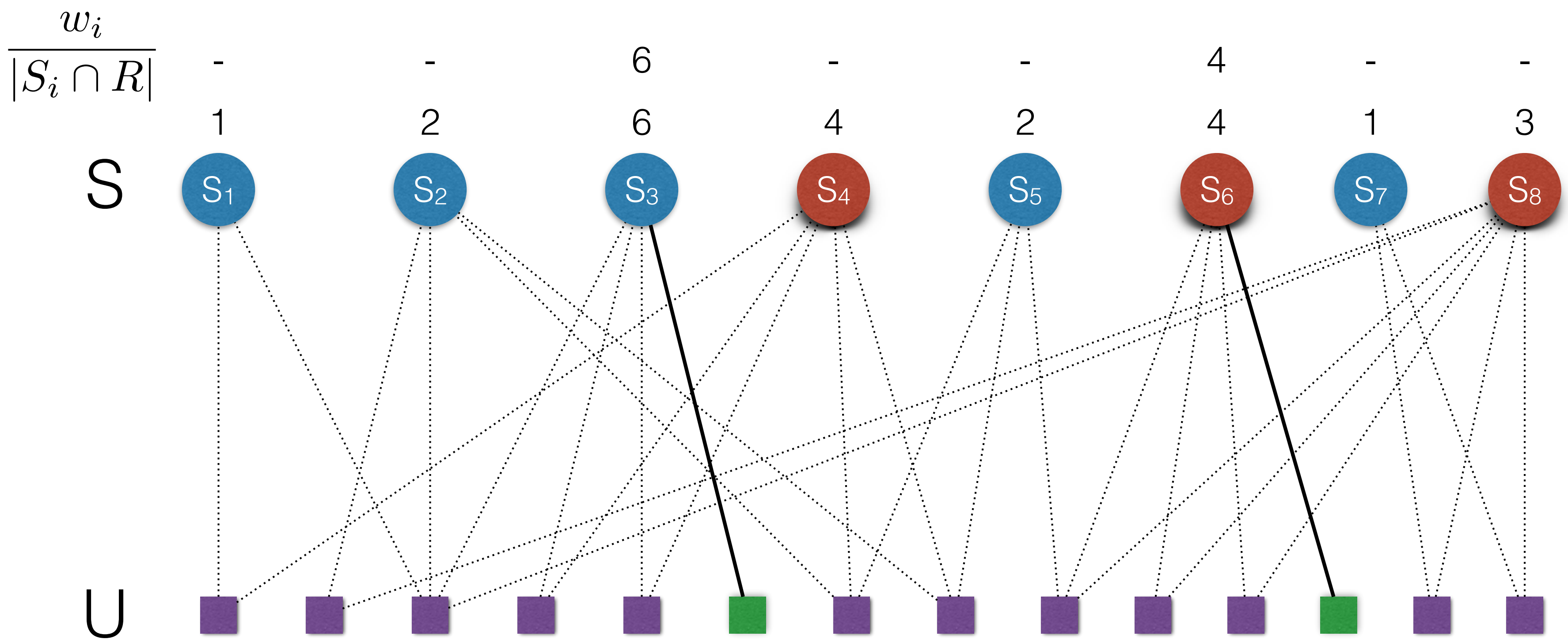
Set Cover: Greedy algorithm



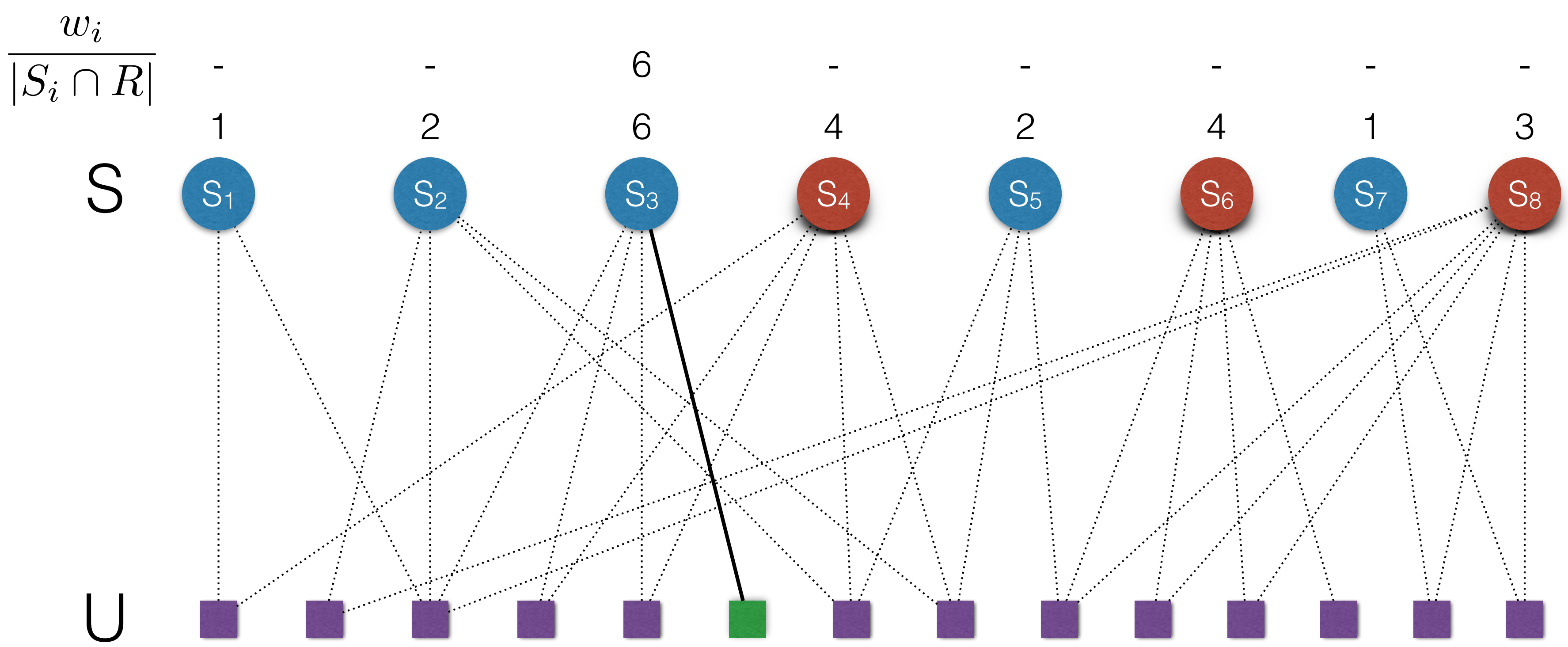
Set Cover: Greedy algorithm



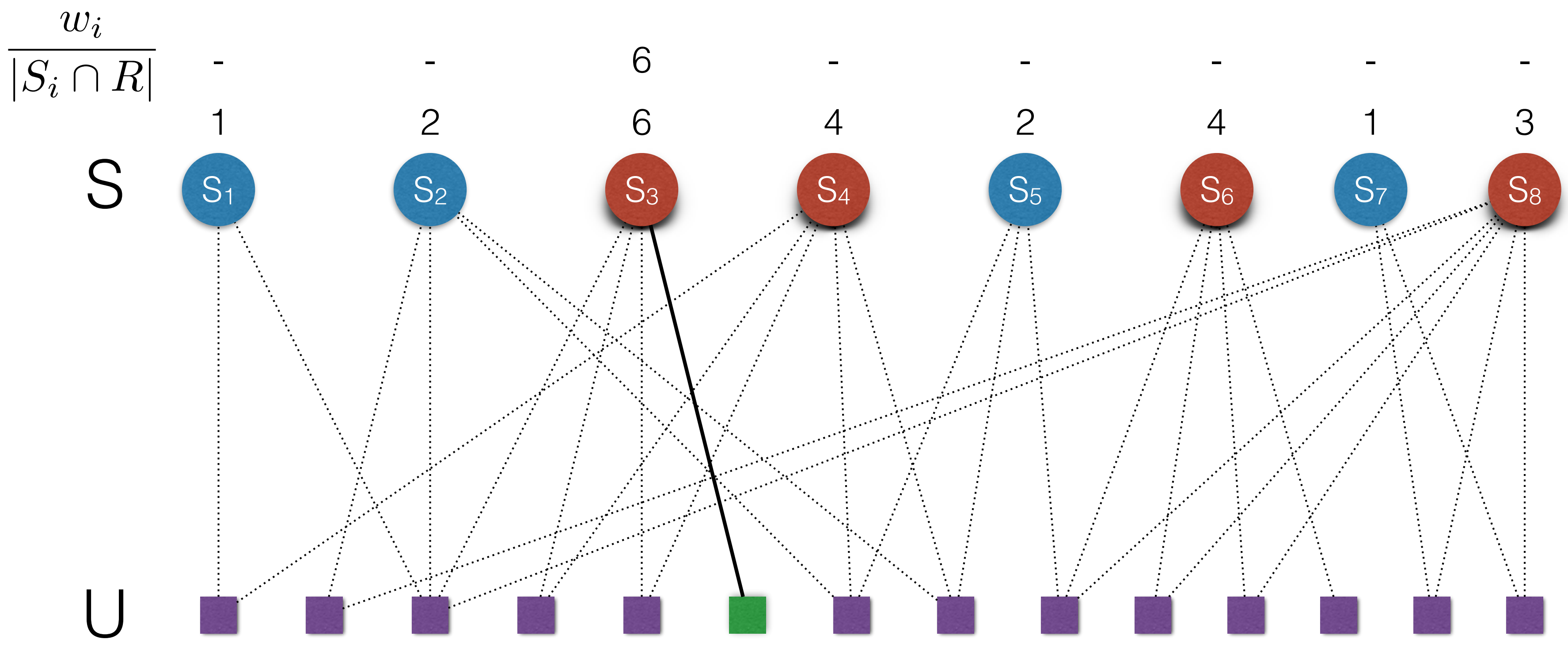
Set Cover: Greedy algorithm



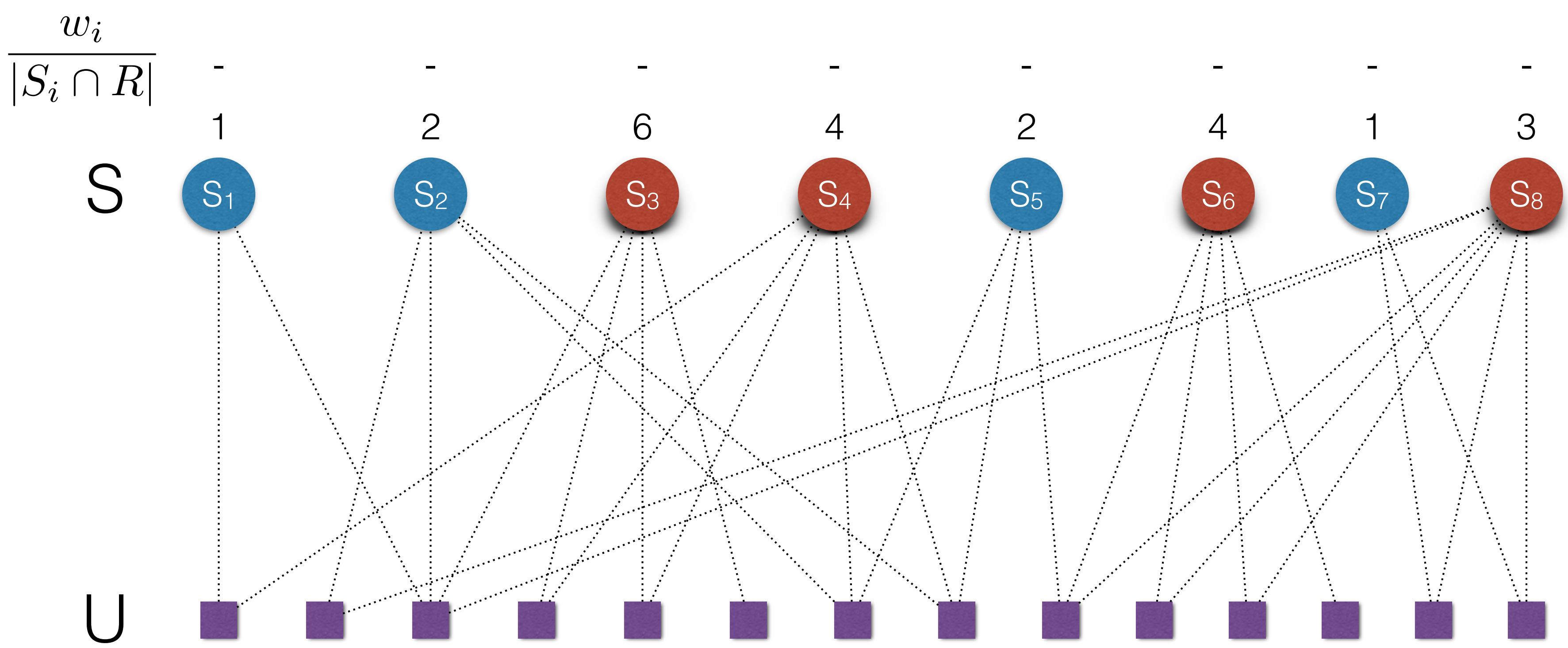
Set Cover: Greedy algorithm



Set Cover: Greedy algorithm



Set Cover: Greedy algorithm



Set cover: greedy algorithm

Greedy-set-cover

Set R := U and C := Ø

while R ≠ Ø

Select the set S_i minimizing $\frac{w_i}{|S_i \cap R|}$

Delete the elements from S_i from R.

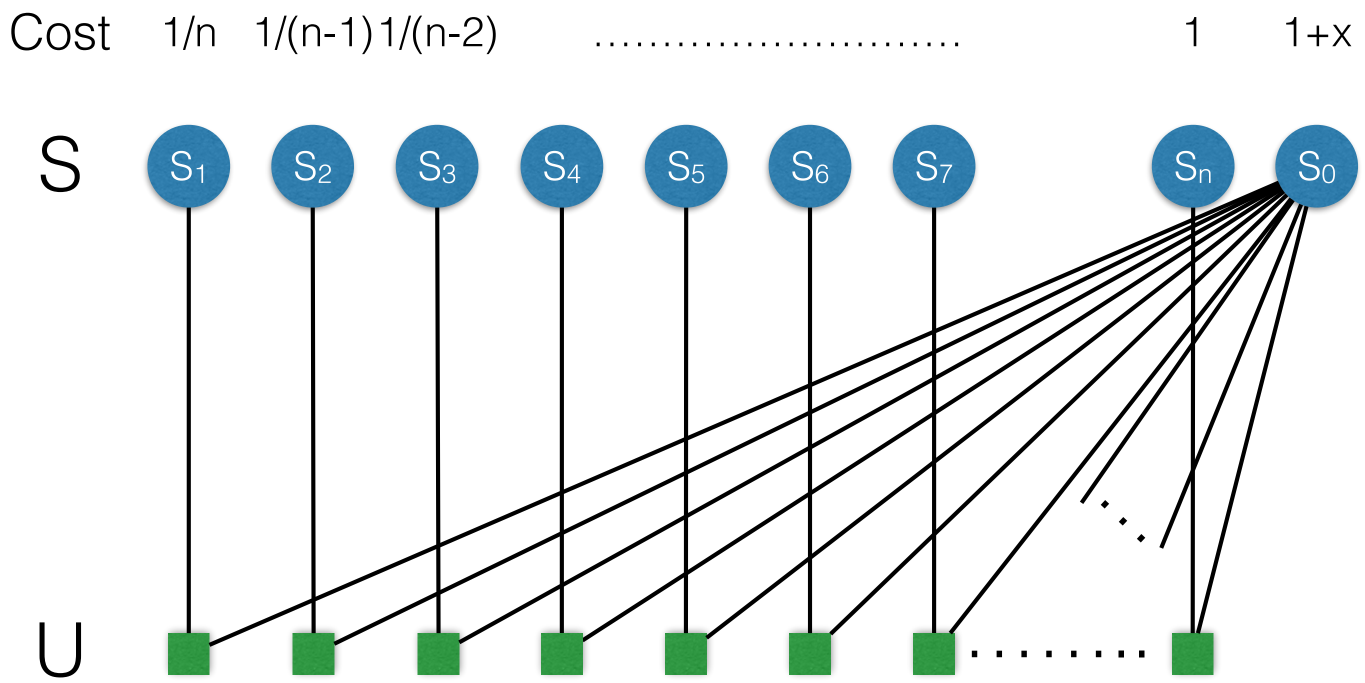
Add S_i to C.

endwhile

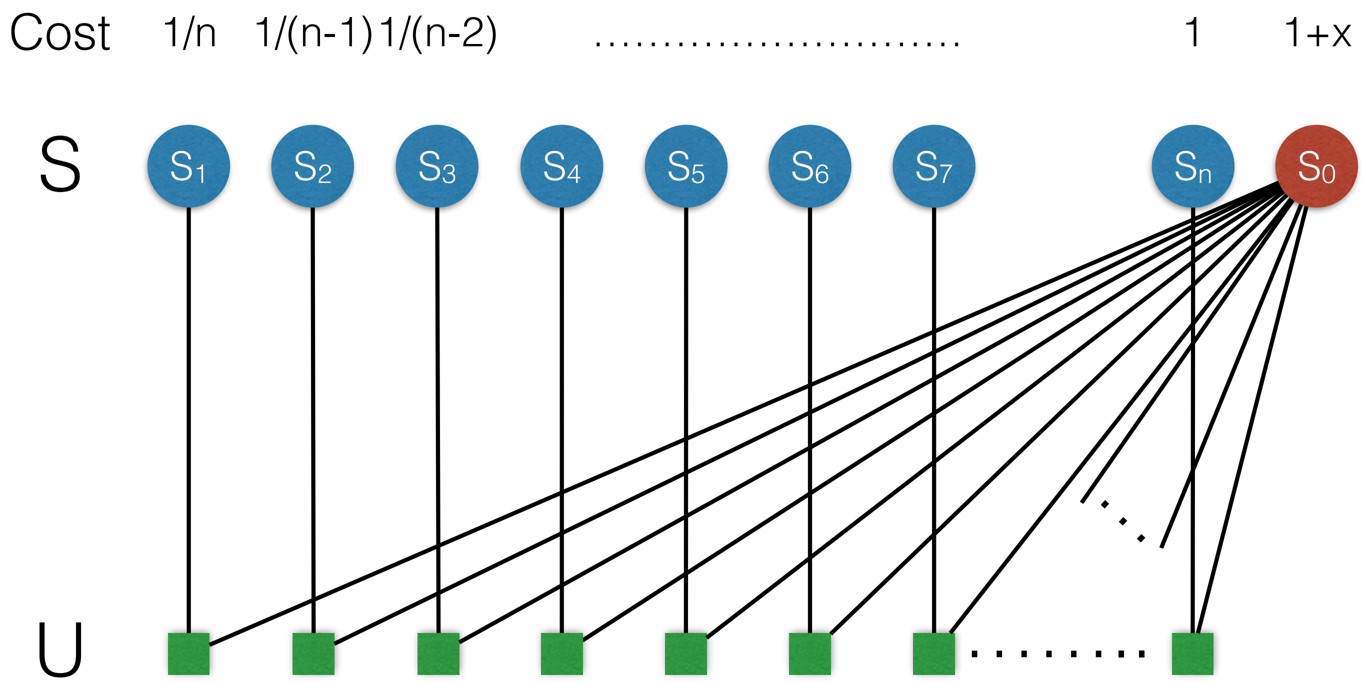
Return C.

- Greedy-set-cover is a n O(log n)-approximation algorithm:
  - polynomial time ✓
  - valid solution ✓
  - factor O(log n)

Set Cover: Greedy algorithm - tight example



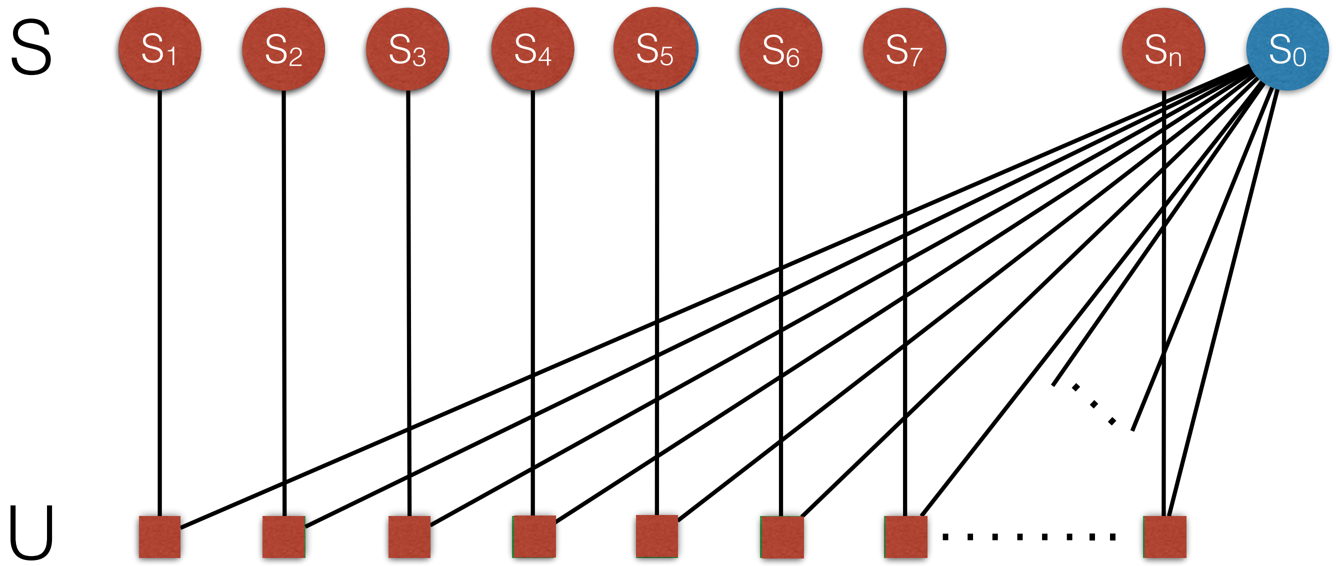
Set Cover: Greedy algorithm - tight example



OPT = 1+x

Set Cover: Greedy algorithm - tight example

$\frac{w_i}{\lvert S_i \cap R \rvert}$	1/n	1/(n-1)	1/(n-2)		1	(1+x)/(n-2)
Cost	1/n	1/(n-1)	1/(n-2)	....	1	1+x



OPT = 1+x

Greedy = 1/n + 1/(n-1) + 1/(n-2) .... = H_n