Algorithms for Decision Support

(Integer) Linear Programming (2/3)

Outline

- More modeling optimization problems to (integer) programming problems
  - **Set cover**
  - **Shortest paths**
  - **Traveling Salesperson Problem**

- LP relaxation and upper/lower bound

- Solving ILP: Branch and bound method

Outline

- More modeling optimization problems to (integer) programming problems
  - **Set cover**
  - **Shortest paths**
  - **Traveling Salesperson Problem**

- LP relaxation and upper/lower bound

- Solving ILP: Branch and bound method

Set Cover Problem

- Given a certain number of regions, the problem is to decide where to install a set of emergency service centers. For each possible center the cost of installing a service center, and which regions it can service are known. The goal is to choose a minimum cost set of service centers so that each region is covered.

Set Cover Problem

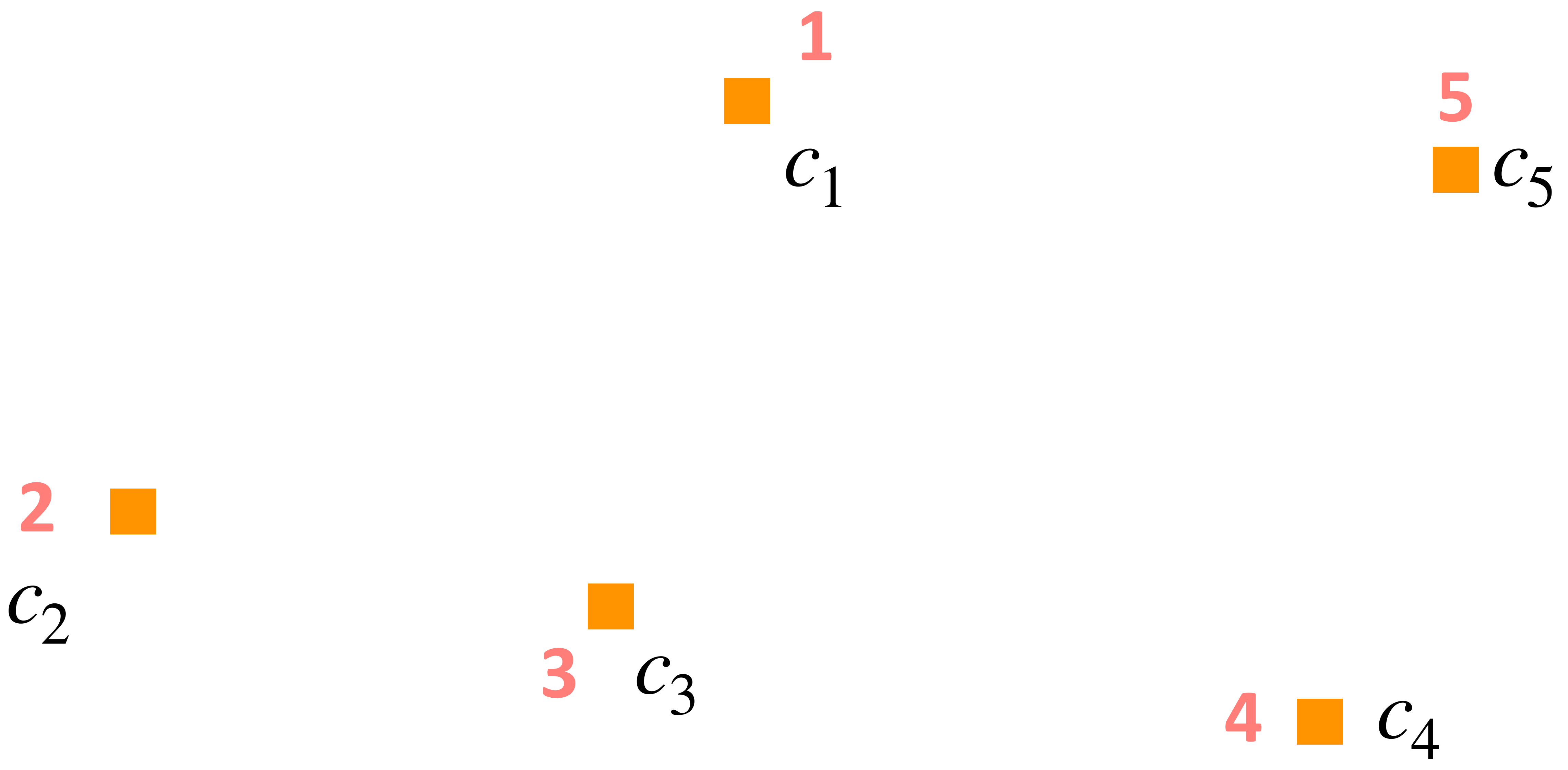
- Given a certain number of regions, the problem is to decide where to install a set of emergency service centers. For each possible center the cost of installing a service center, and which regions it can service are known. The goal is to choose a minimum cost set of service centers so that each region is covered.
- A more mathematical description:
  Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

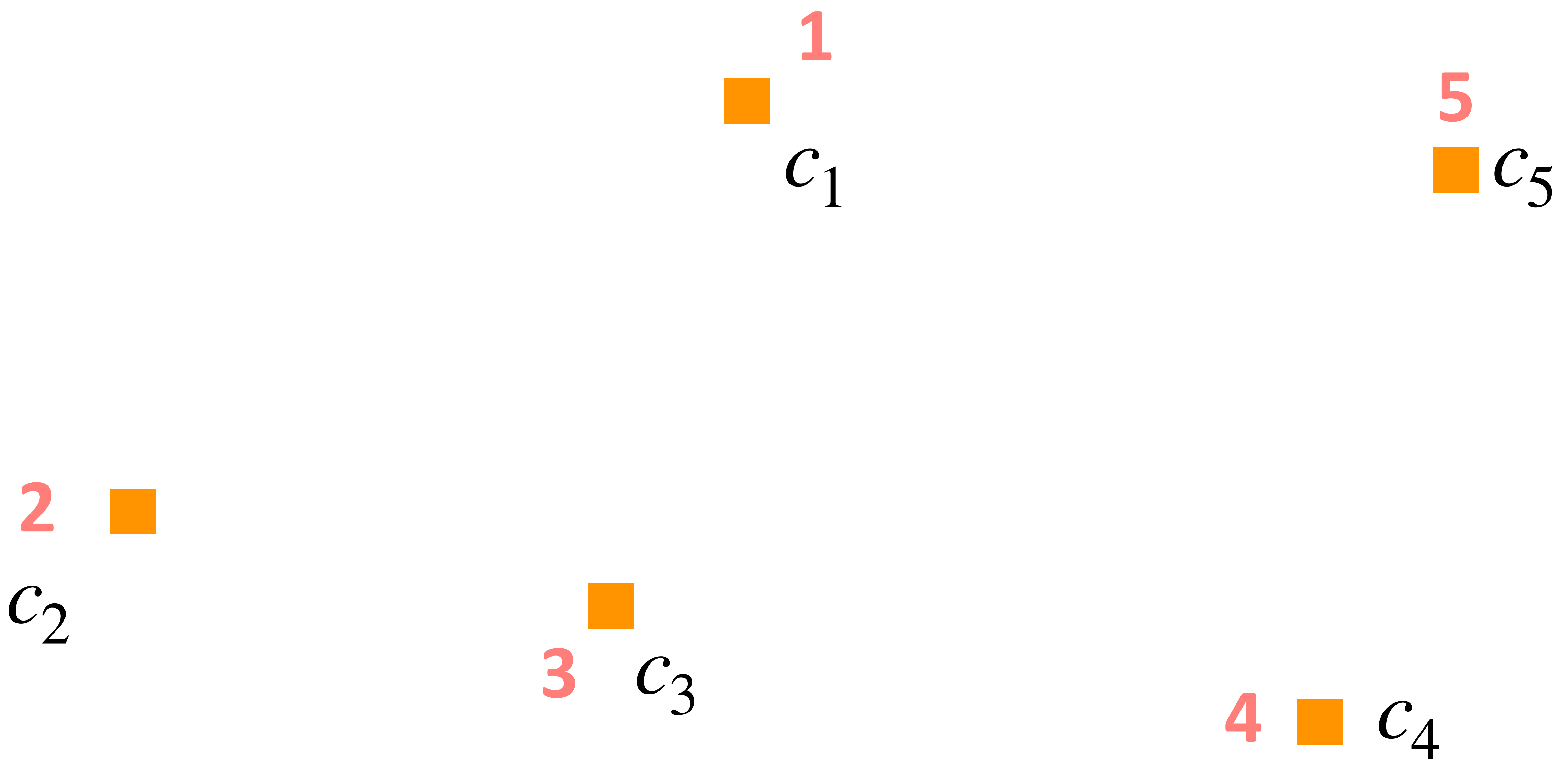
Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.



Set Cover Problem

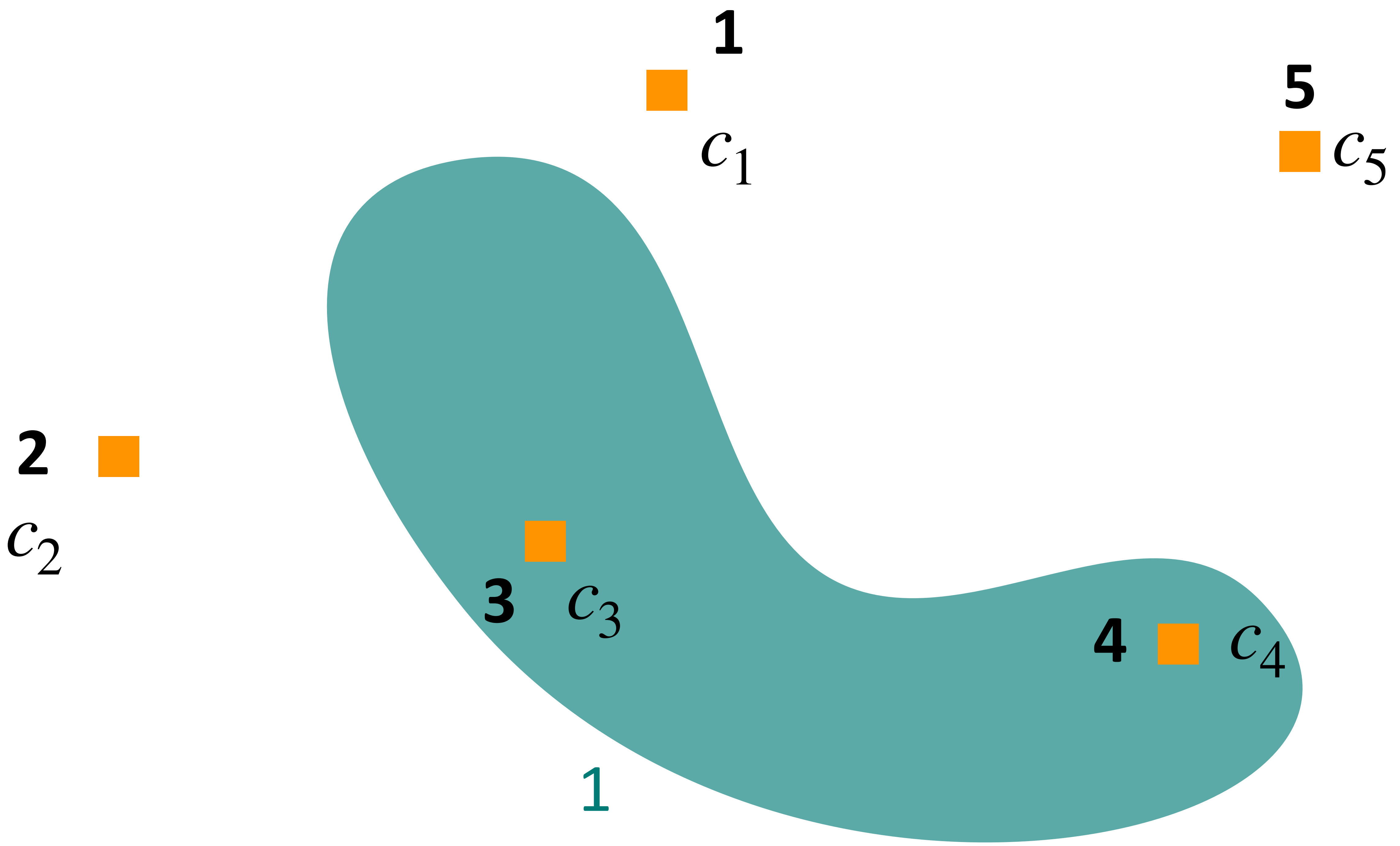
- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.



Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

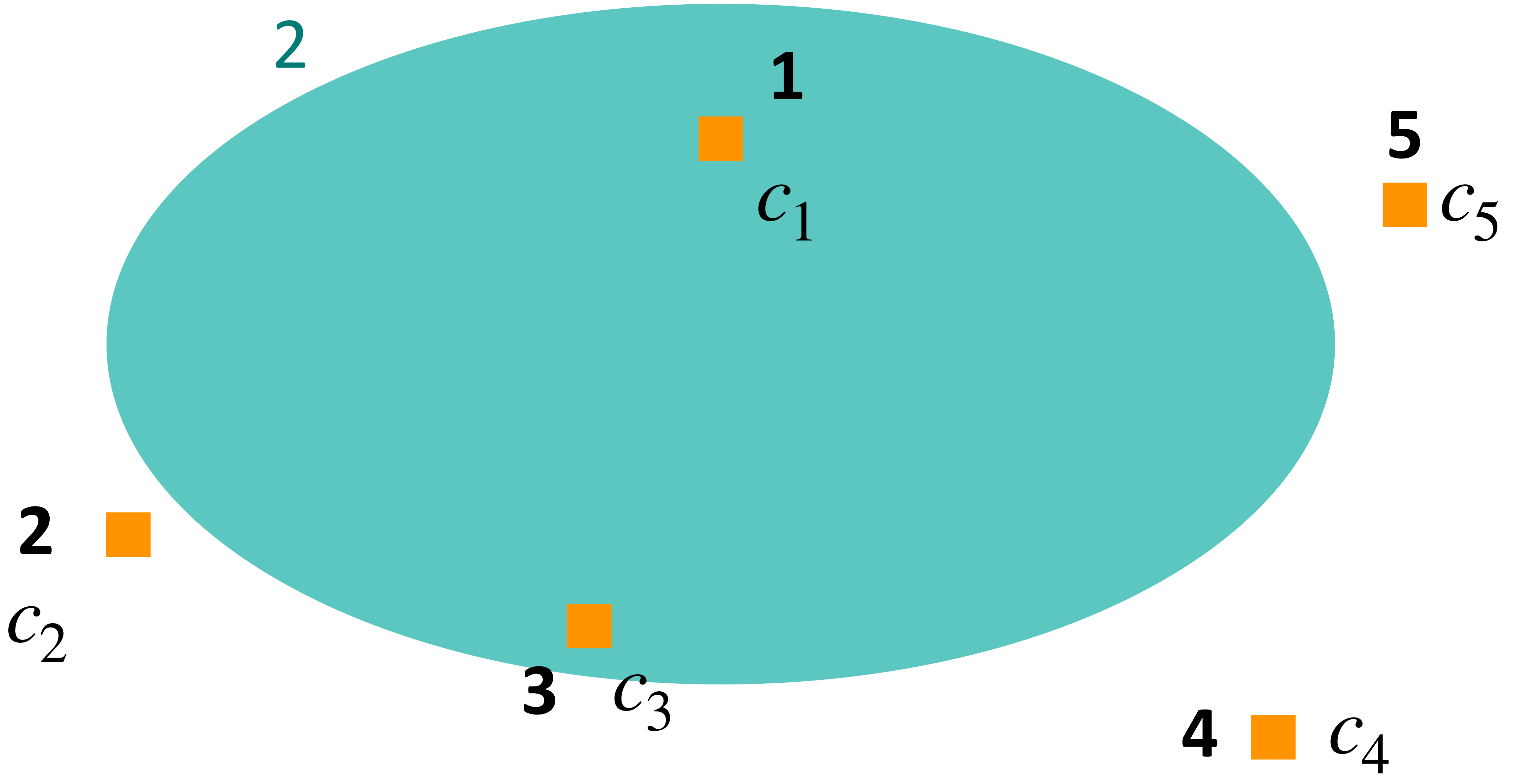
$S_1 = \{3, 4\}$



Set Cover Problem

- Let $M = \{1, 2, \cdots, m\}$ be the set of regions, and $N = \{1, 2, \cdots, n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

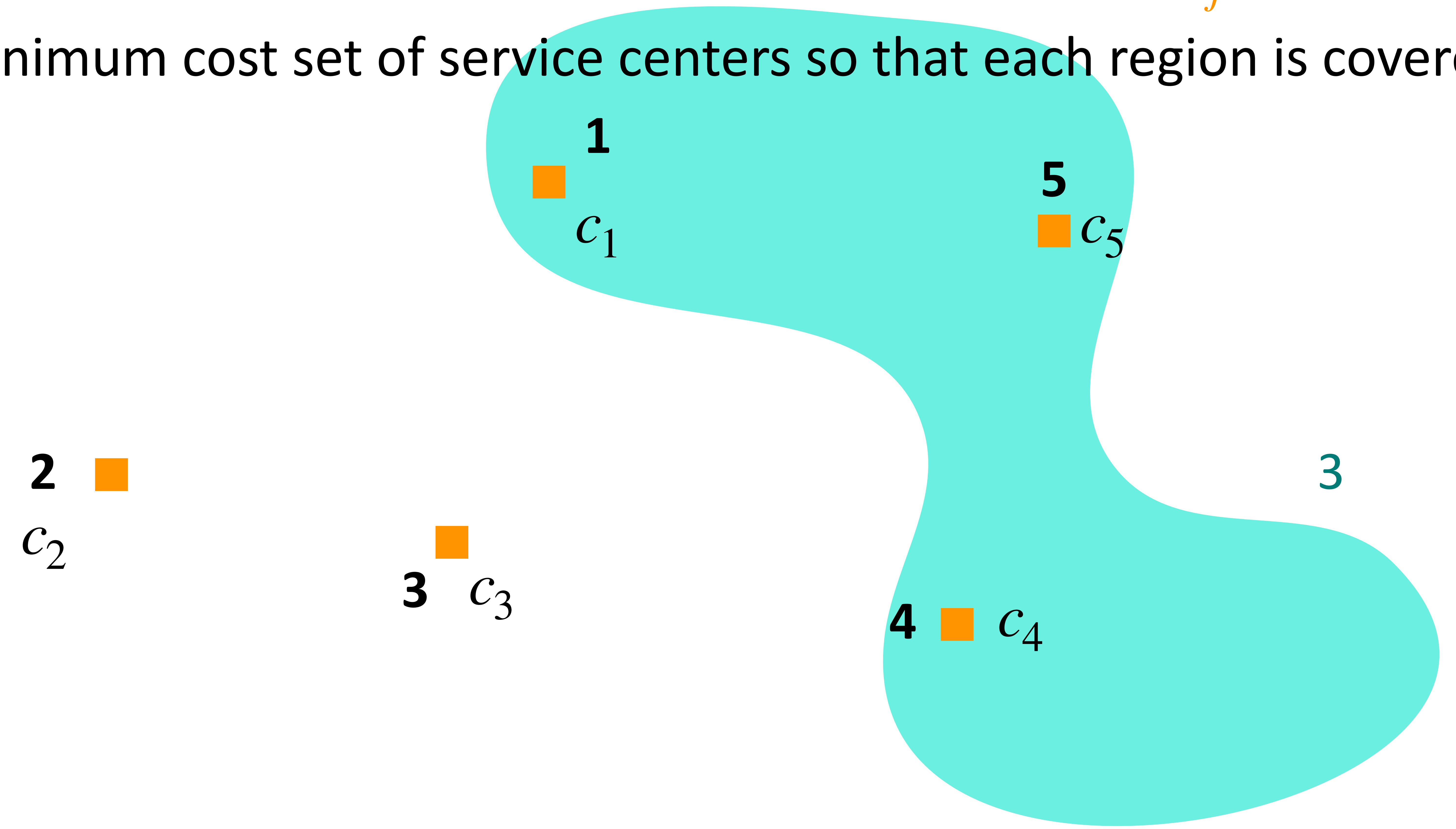
$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$



Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

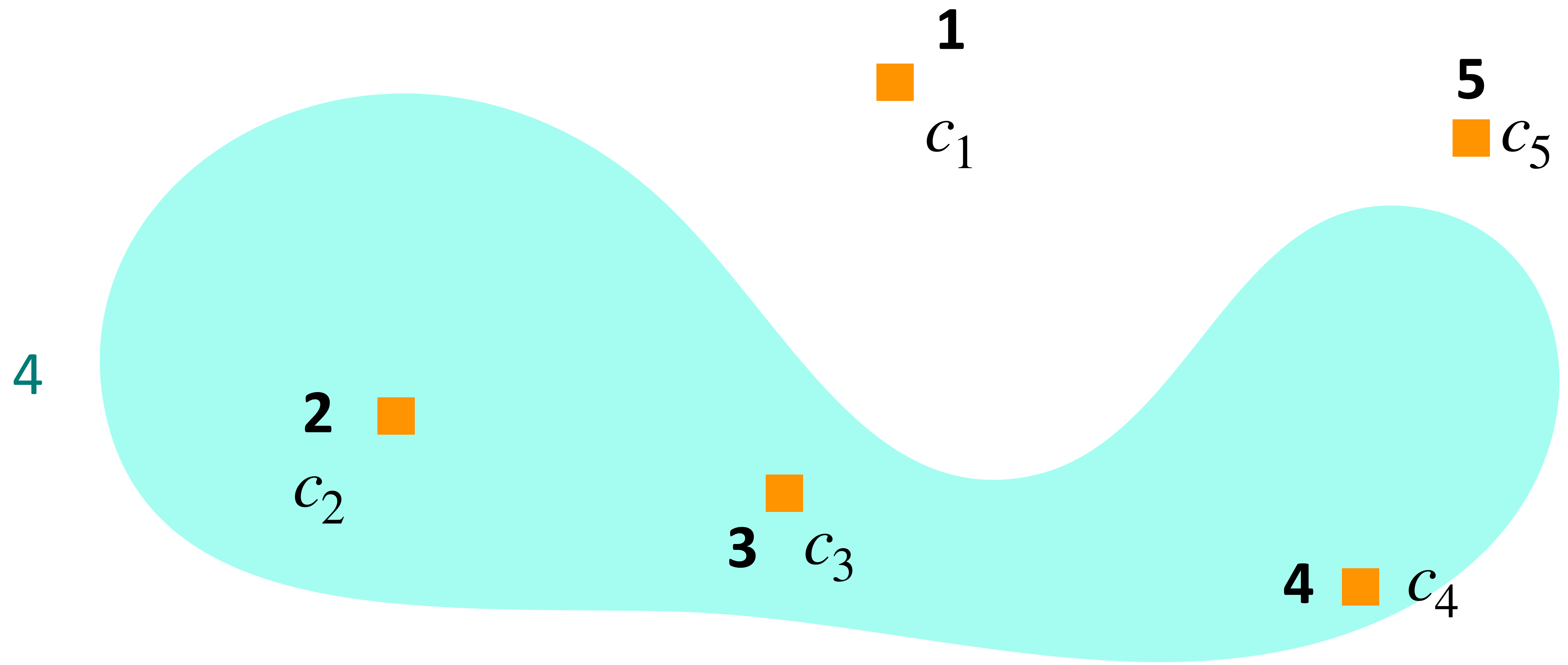
$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$
$S_3 = \{1, 4, 5\}$



Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

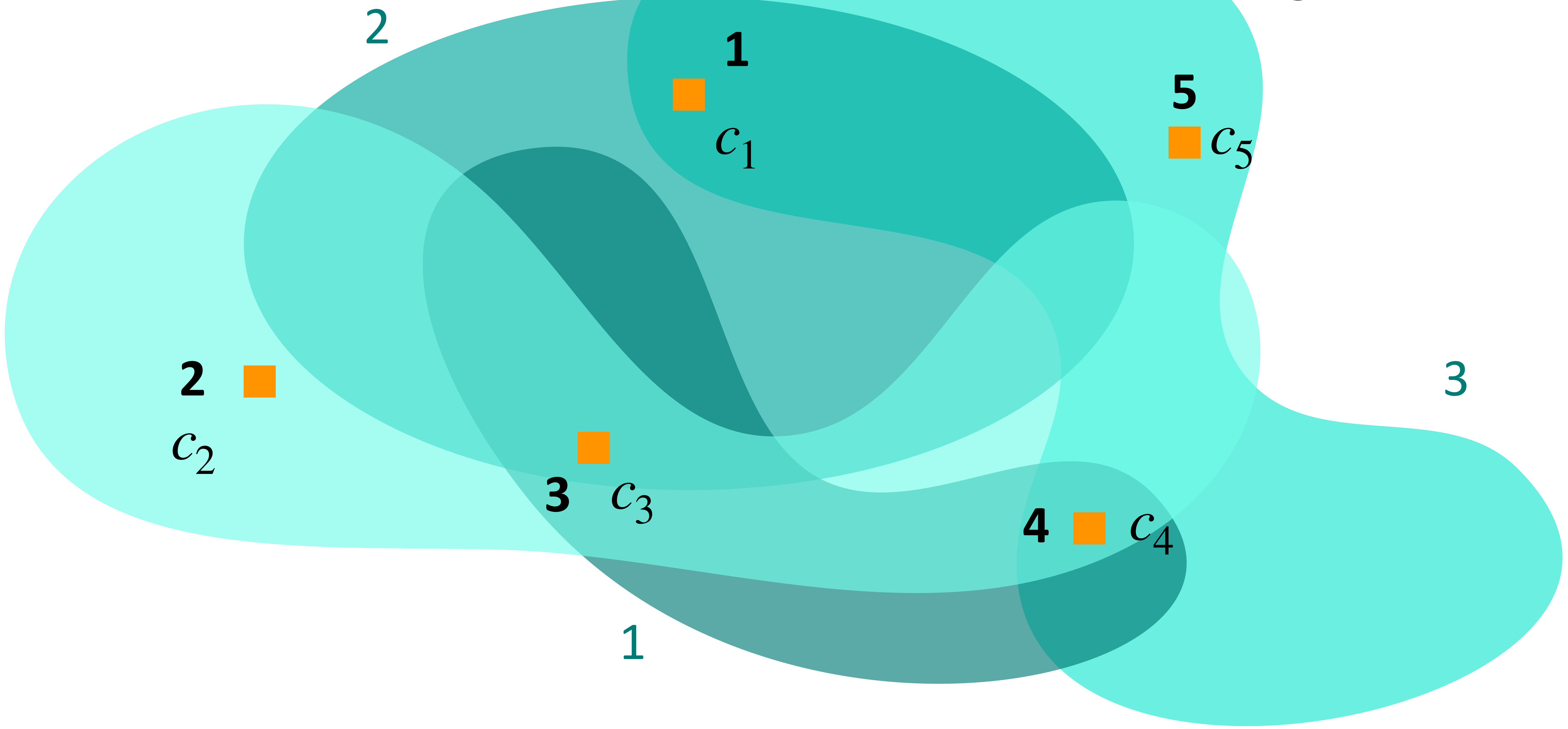
$$S_1 = \{3, 4\}$$
$$S_2 = \{1, 3\}$$
$$S_3 = \{1, 4, 5\}$$
$$S_4 = \{2, 3, 4\}$$



Set Cover Problem

- Let $M = \{1, 2, \cdots, m\}$ be the set of regions, and $N = \{1, 2, \cdots, n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

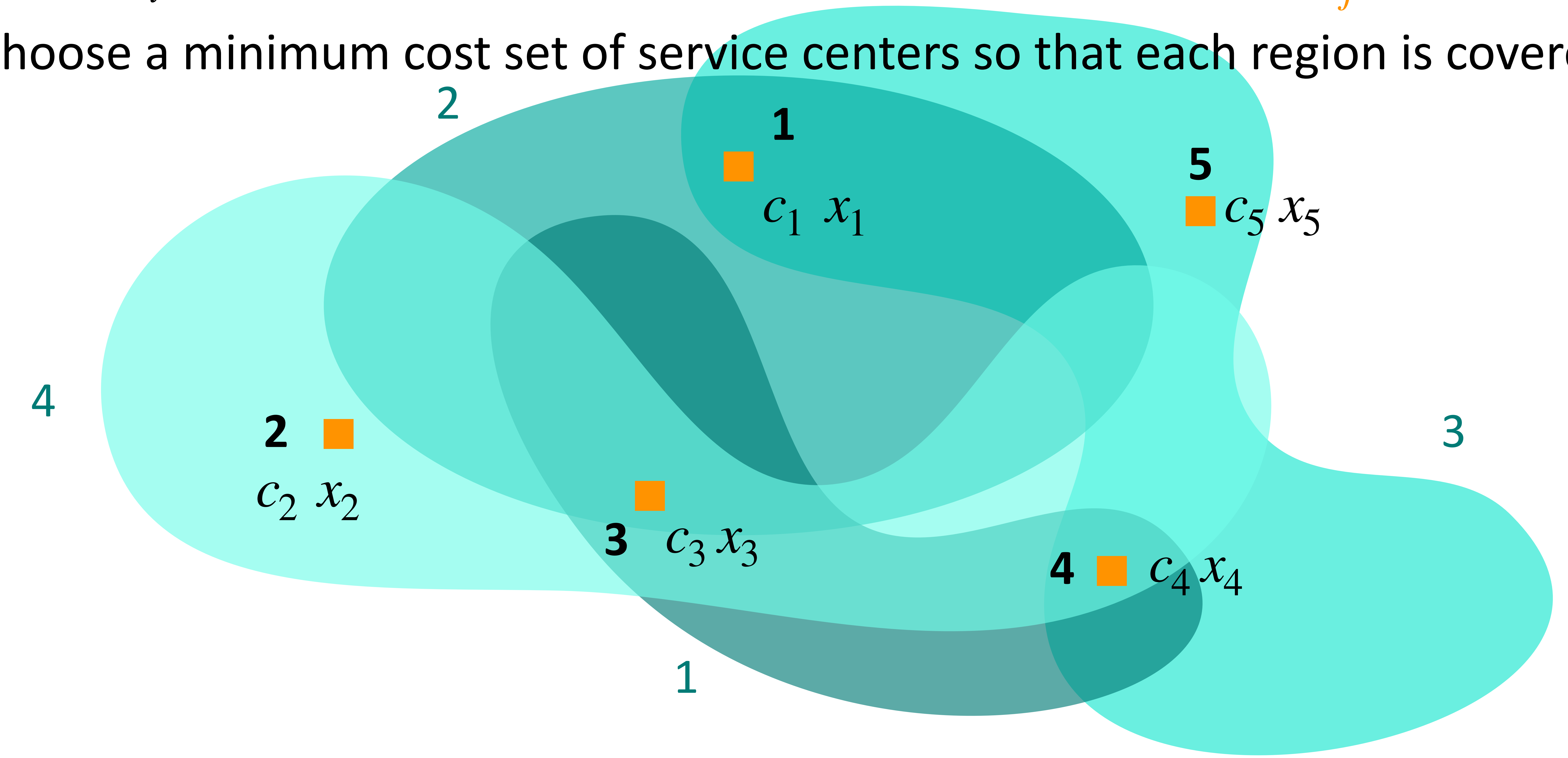
$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$
$S_3 = \{1, 4, 5\}$
$S_4 = \{2, 3, 4\}$



Set Cover Problem

- Let $M = \{1, 2, \cdots, m\}$ be the set of regions, and $N = \{1, 2, \cdots, n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

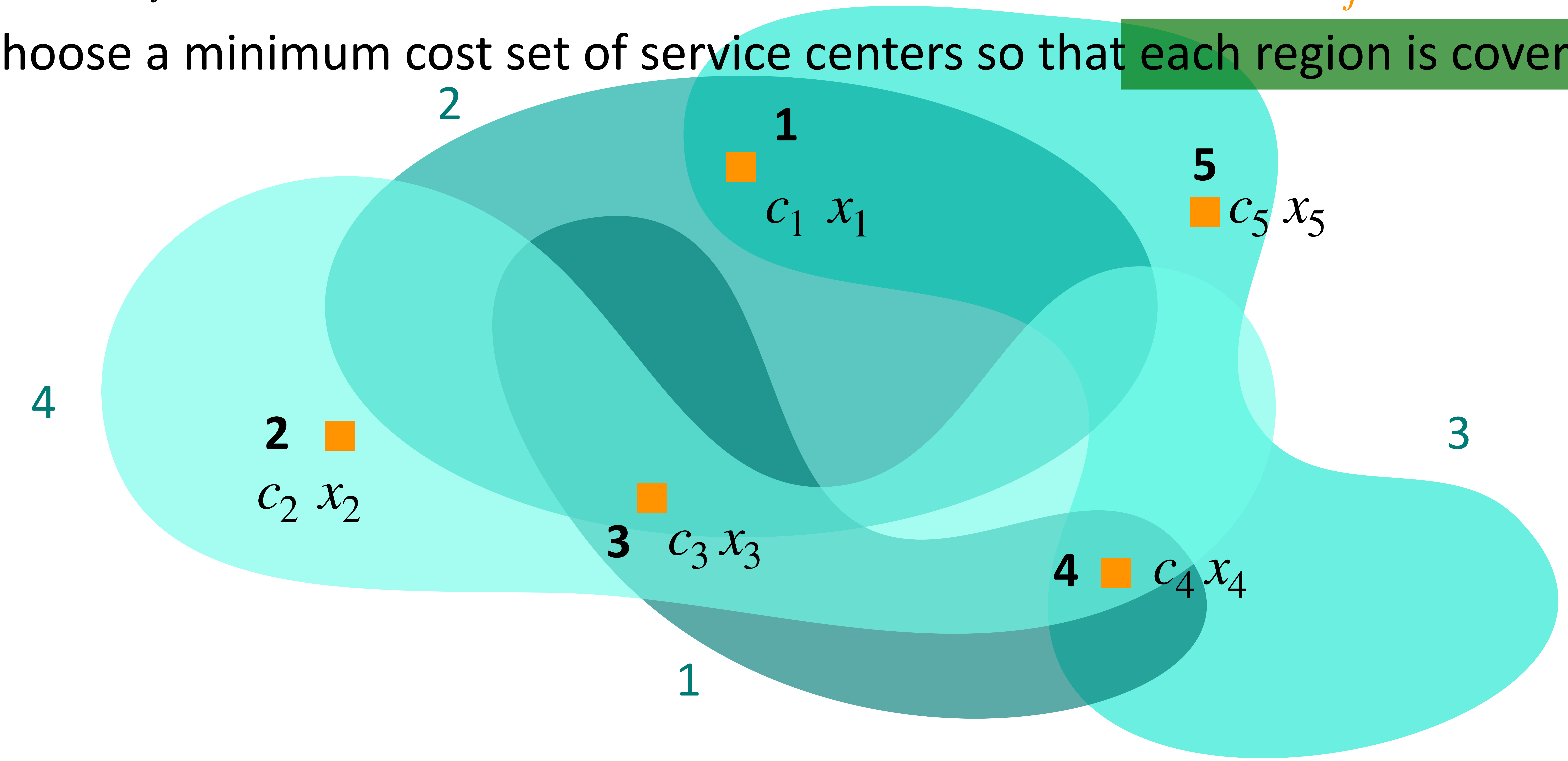
$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$
$S_3 = \{1, 4, 5\}$
$S_4 = \{2, 3, 4\}$



Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

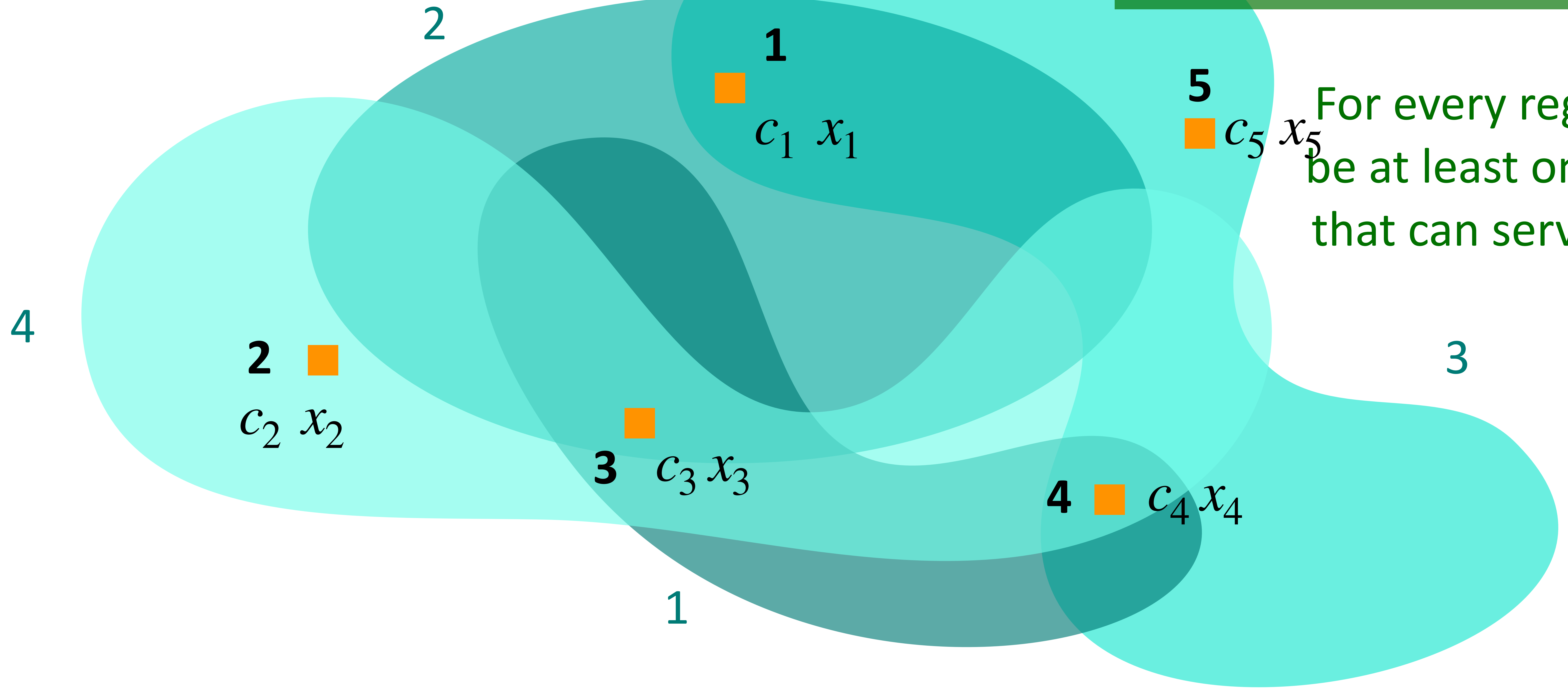
$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$
$S_3 = \{1, 4, 5\}$
$S_4 = \{2, 3, 4\}$



Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

$$S_1 = \{3, 4\}$$
$$S_2 = \{1, 3\}$$
$$S_3 = \{1, 4, 5\}$$
$$S_4 = \{2, 3, 4\}$$



For every region, there must be at least one service center that can service it is selected

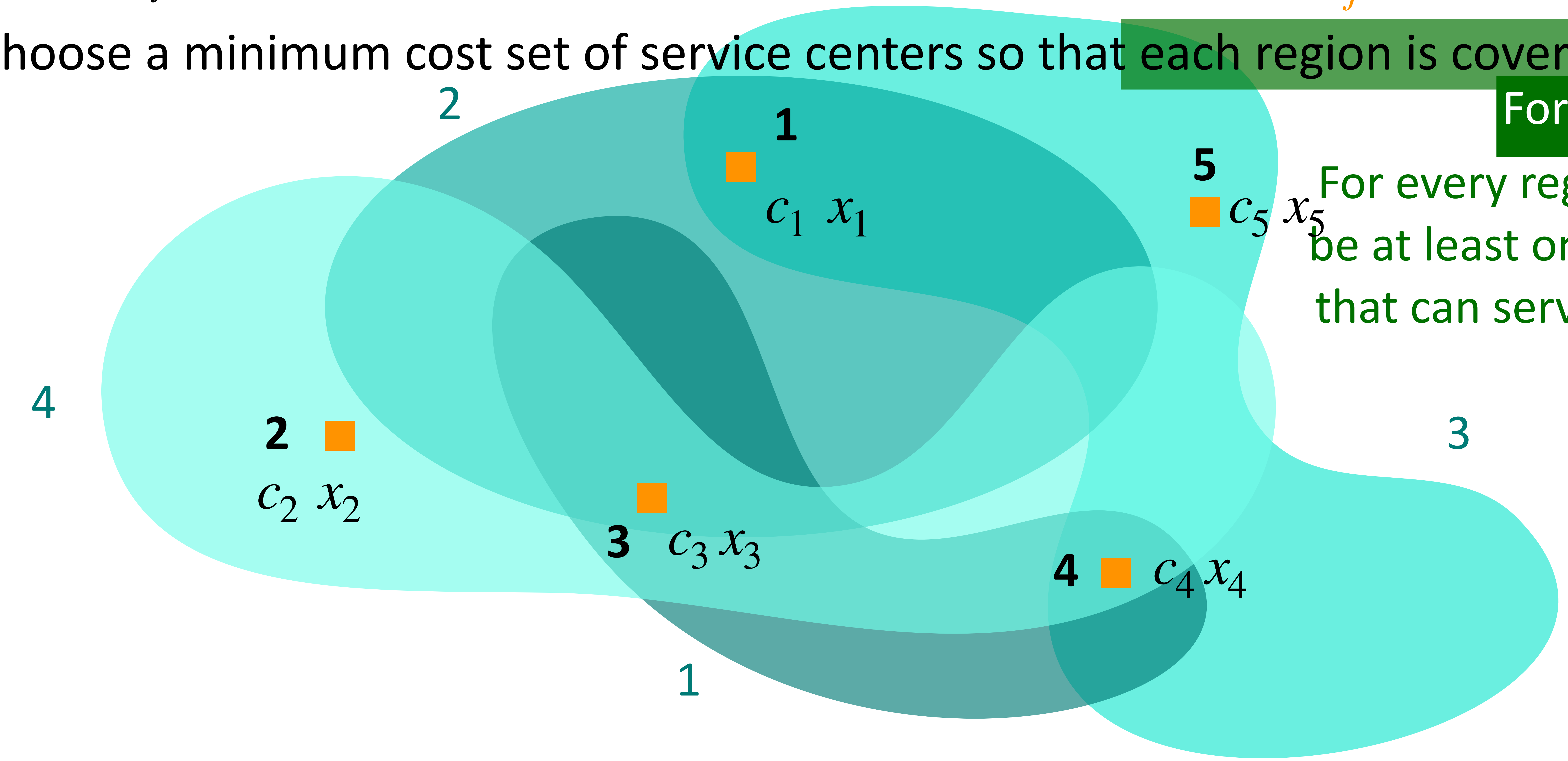
Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

For all i, $\Sigma_{j \in S_i} x_j \geq 1$

For every region, there must be at least one service center that can service it is selected

$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$
$S_3 = \{1, 4, 5\}$
$S_4 = \{2, 3, 4\}$

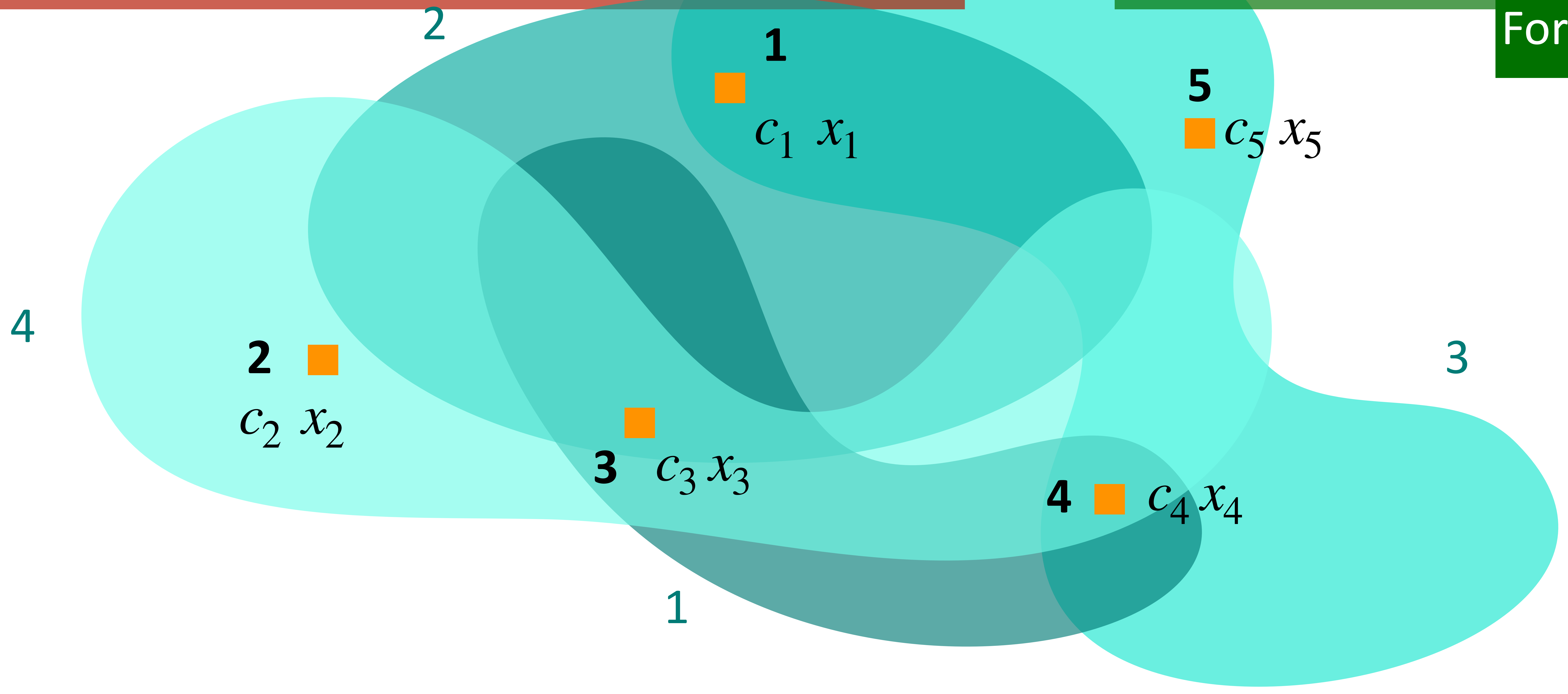


Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

For all i, $\Sigma_{j \in S_i} x_j \geq 1$

$S_1 = \{3, 4\}$
$S_2 = \{1, 3\}$
$S_3 = \{1, 4, 5\}$
$S_4 = \{2, 3, 4\}$



Set Cover Problem

- Let $M = \{1,2,\cdots,m\}$ be the set of regions, and $N = \{1,2,\cdots,n\}$ be the set of potential centers. Let $S_i \subseteq N$ be the centers j that can service set $i \in M$, and c_j its installation cost. Choose a minimum cost set of service centers so that each region is covered.

For all i, $\Sigma_{j\in S_i} x_j \geq 1$

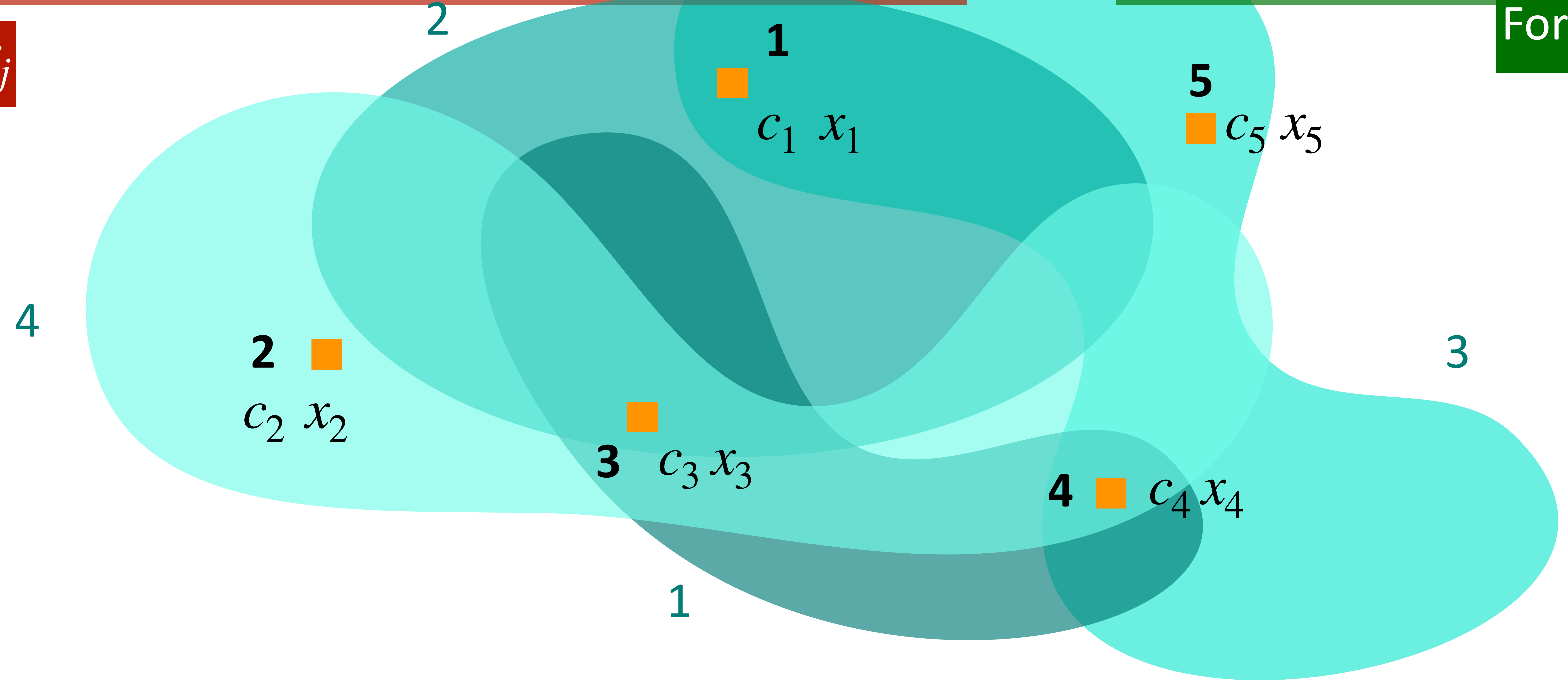
$\min \Sigma_{j=1}^{n} c_j x_j$

$S_1 = \{3, 4\}$

$S_2 = \{1, 3\}$

$S_3 = \{1, 4, 5\}$

$S_4 = \{2, 3, 4\}$



Set Cover Problem

- Variable: $x_j = 1$ if center j is selected, and $x_j = 0$ otherwise

- Minimize $\Sigma_{j=1}^{n} c_j x_j$

  subject to $\Sigma_{j \in S_i} x_j \geq 1$ for $i = 1, \cdots, m$

  $x_j \in \{0,1\}$ for $j = 1, \cdots, n$

Outline

- More modeling optimization problems to (integer) programming problems
  - **Set cover**
  - **Shortest paths**
  - **Traveling Salesperson Problem**
- LP relaxation and upper/lower bound
- Solving ILP: Branch and bound method

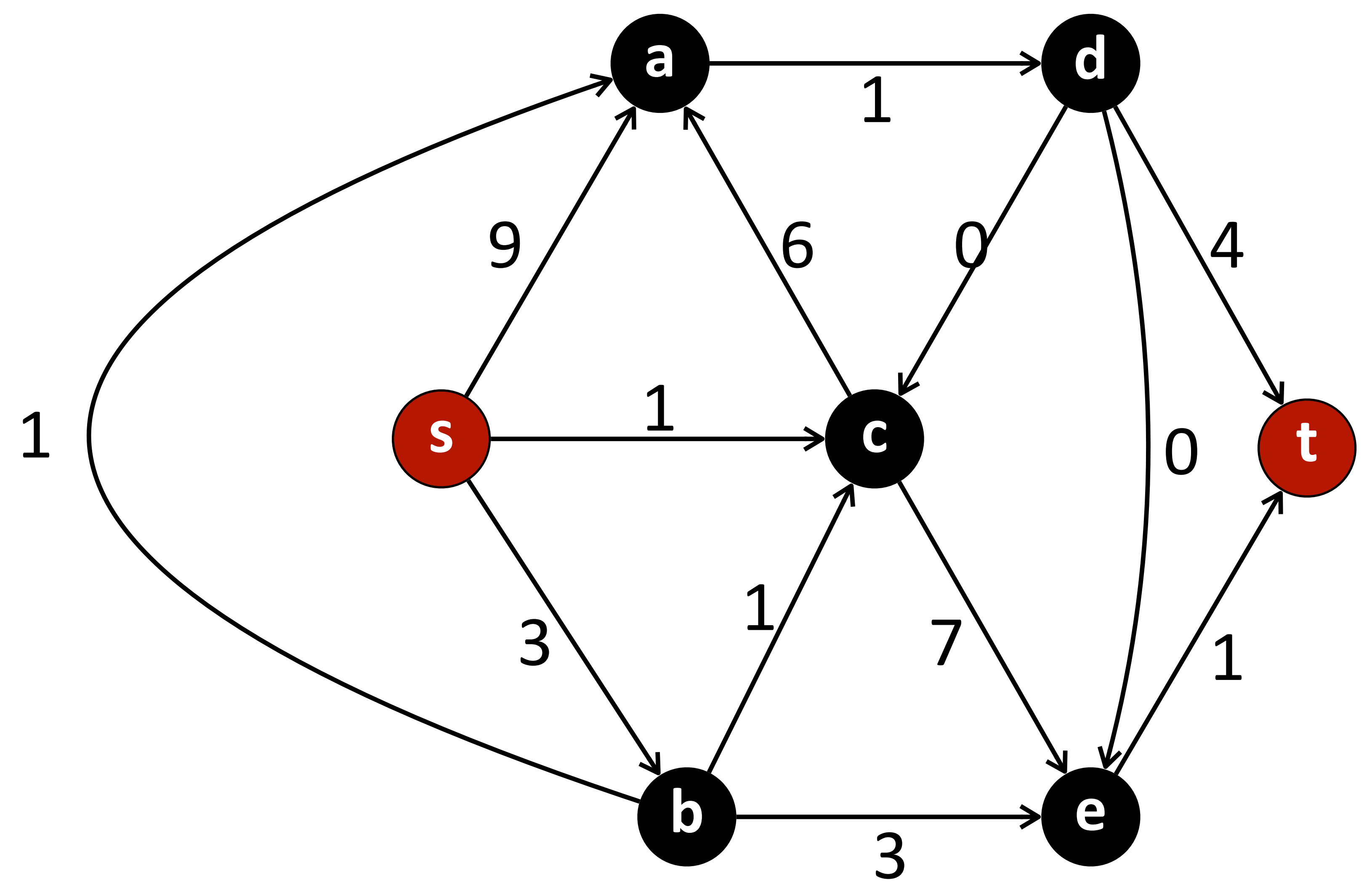
Shortest paths

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

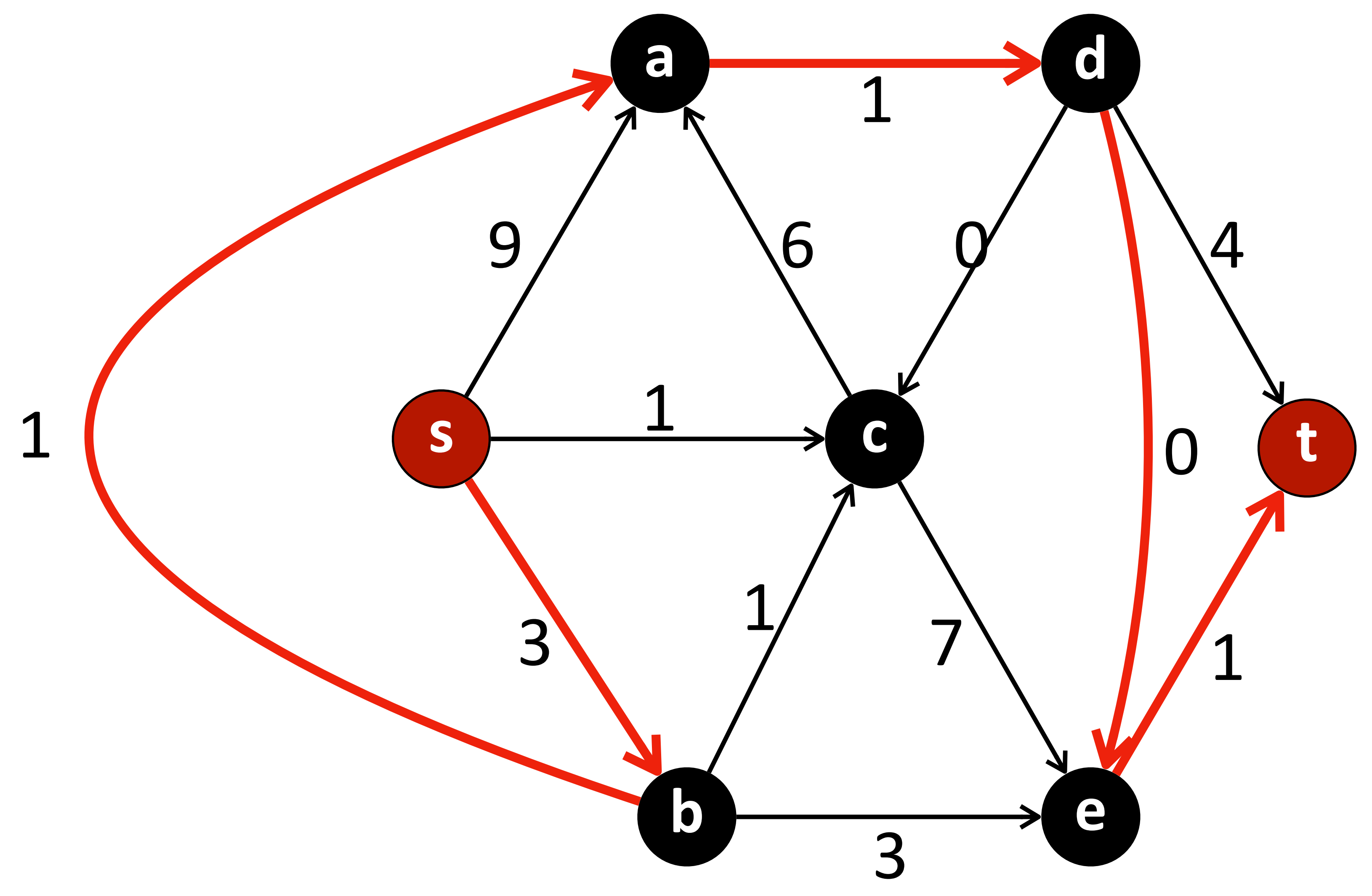
Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.



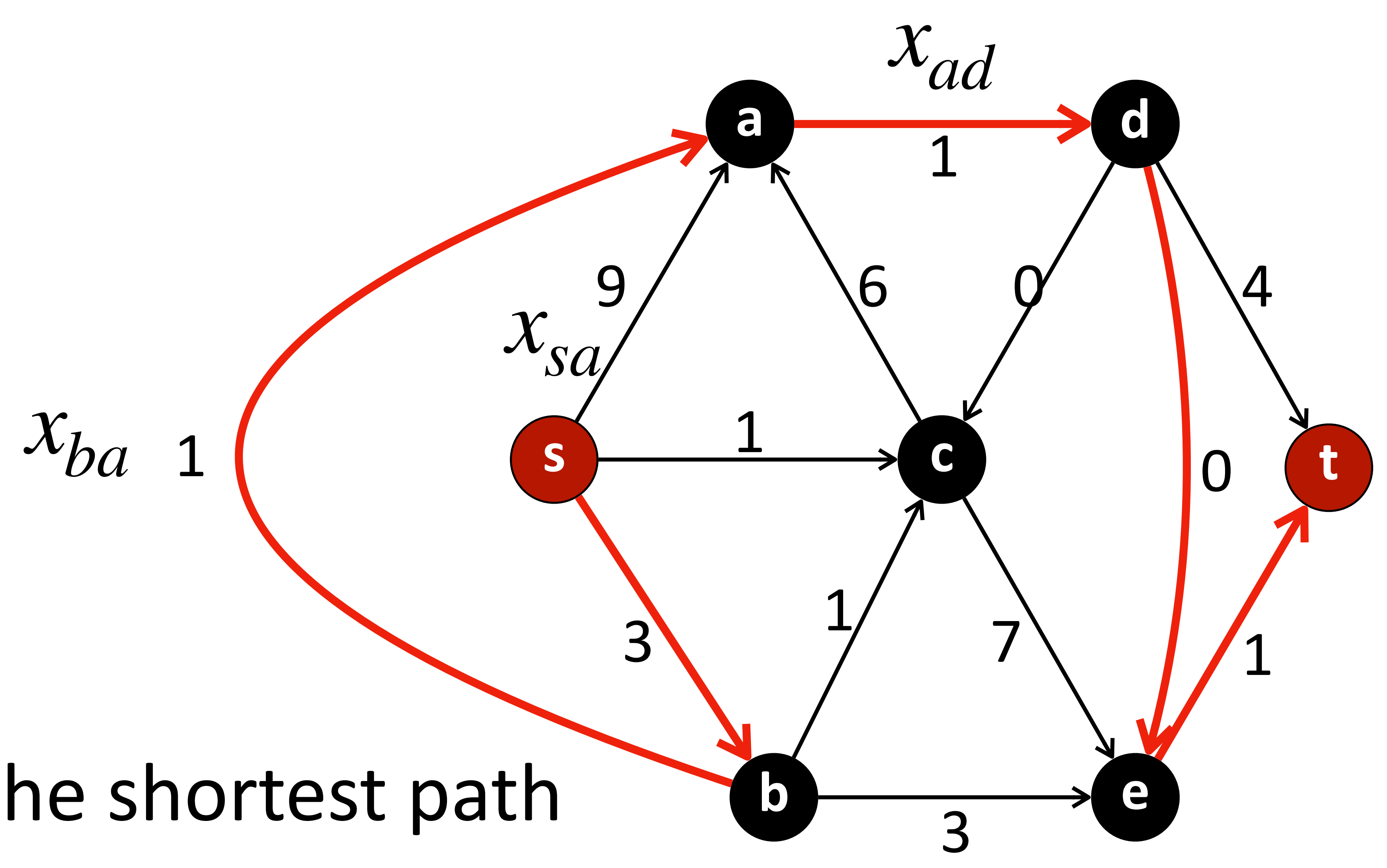
Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.



Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.



Decide if edge (u, v) is in the shortest path

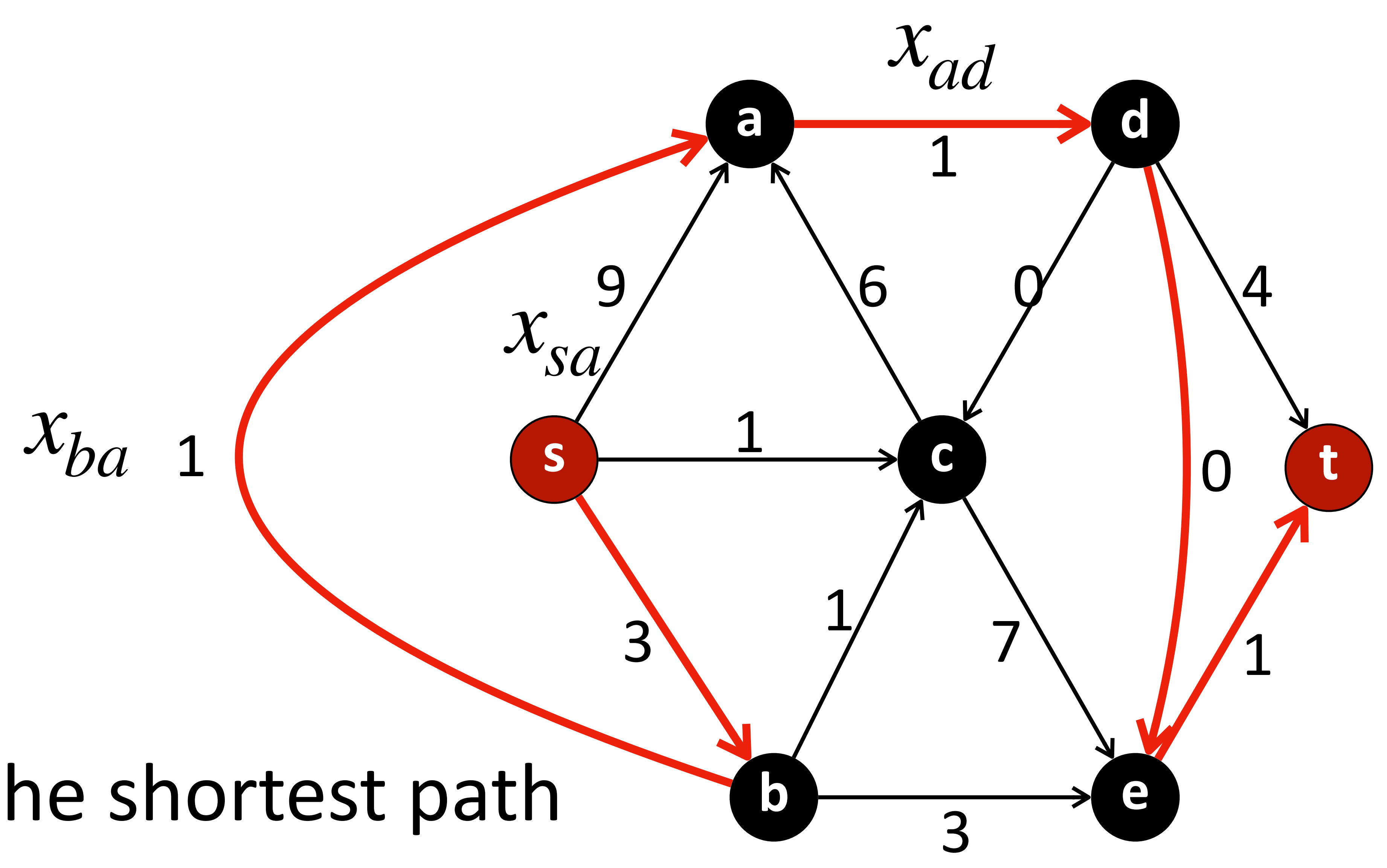
$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

$\min \Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$



Decide if edge (u, v) is in the shortest path

$x_{uv} = 1$ if (u, v) is in the shortest path

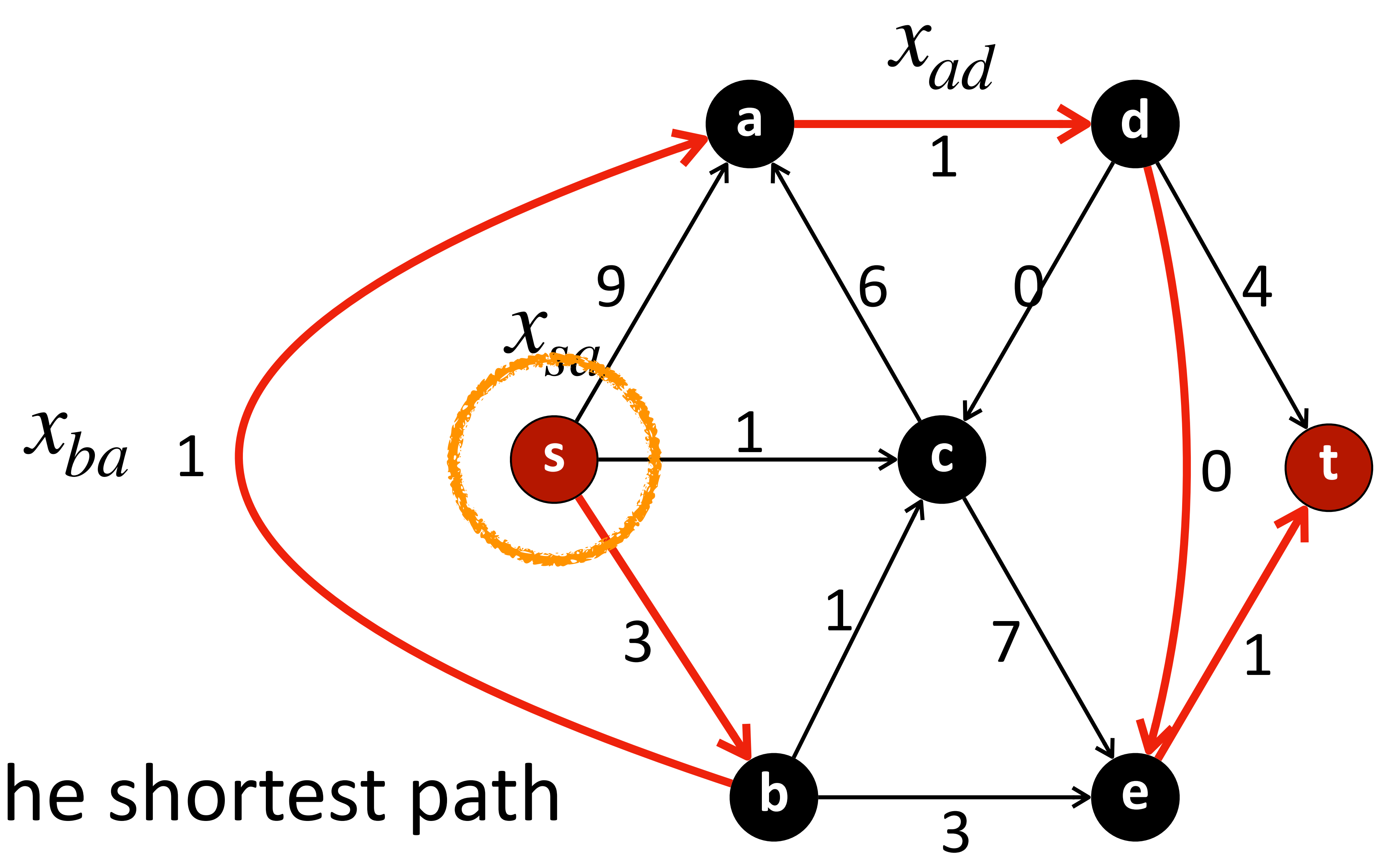
$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

For s, $\Sigma_{(s,k)\in E} x_{sk} = 1$

$\min \Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$



Decide if edge (u, v) is in the shortest path

$x_{uv} = 1$ if (u, v) is in the shortest path

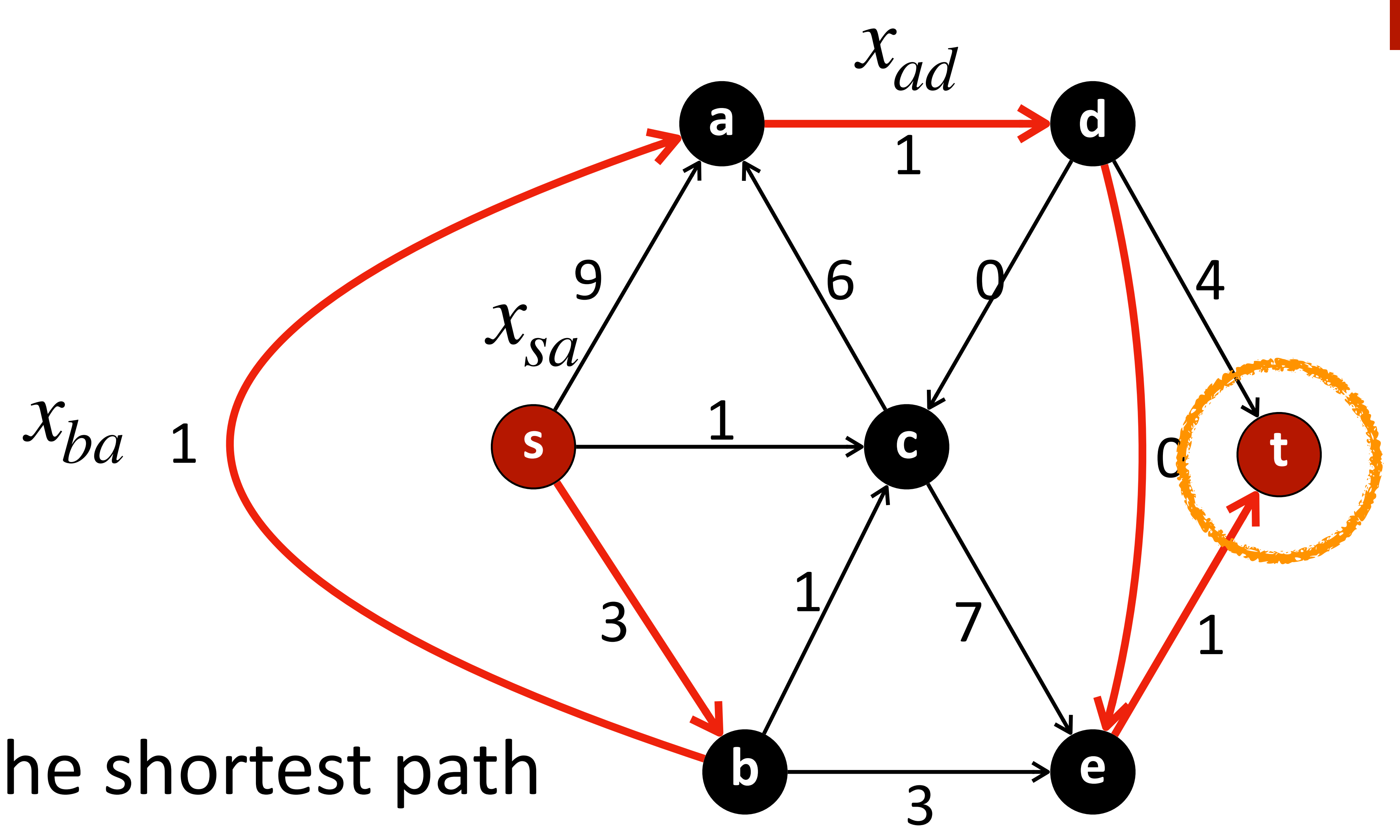
$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

$\min \Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$

For t, $\Sigma_{(k,t)\in E} x_{kt} = 1$



Decide if edge (u, v) is in the shortest path

$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

Shortest paths

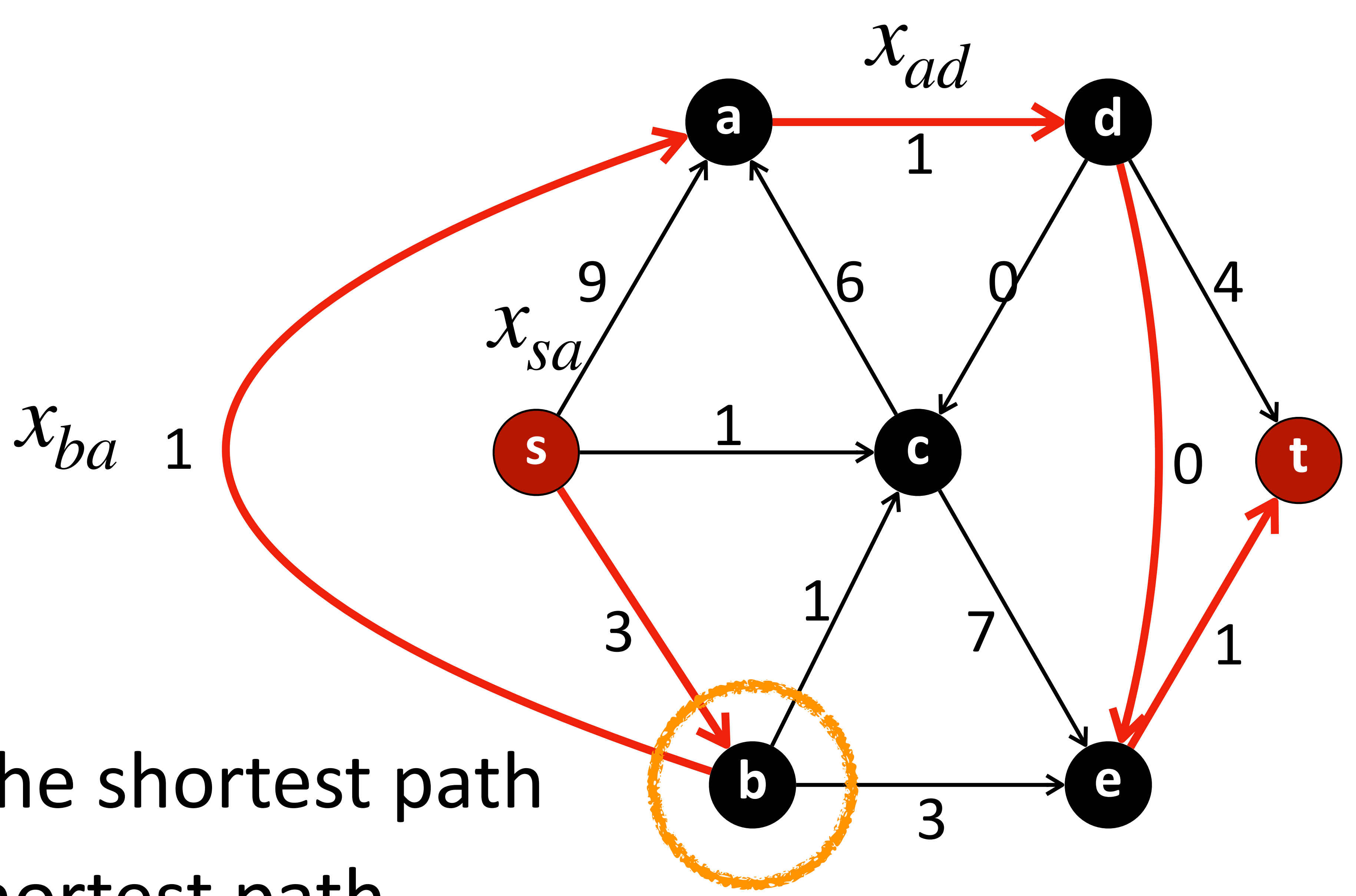
- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

$\min \Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$

For v on the shortest path,

$\Sigma_{(k,v)\in E} x_{kv} = 1$

$\Sigma_{(v,k)\in E} x_{vk} = 1$



Decide if edge (u, v) is in the shortest path

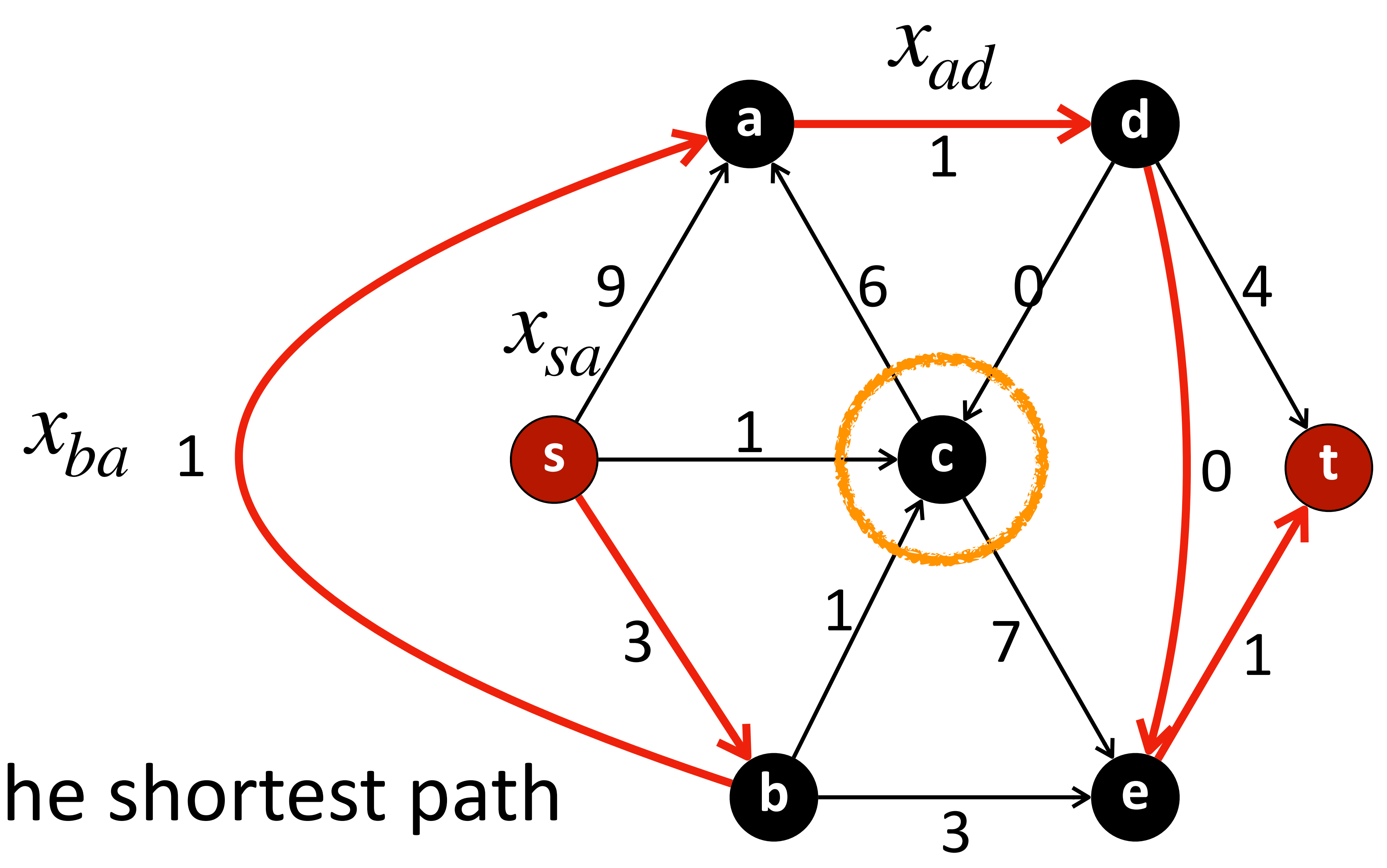
$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

$\min \Sigma_{(u,v)\in E} \ell_{uv} x_{uv}$



For v **not** on the shortest path,

$\Sigma_{(k,v)\in E} x_{kv} = 0$

$\Sigma_{(v,k)\in E} x_{vk} = 0$

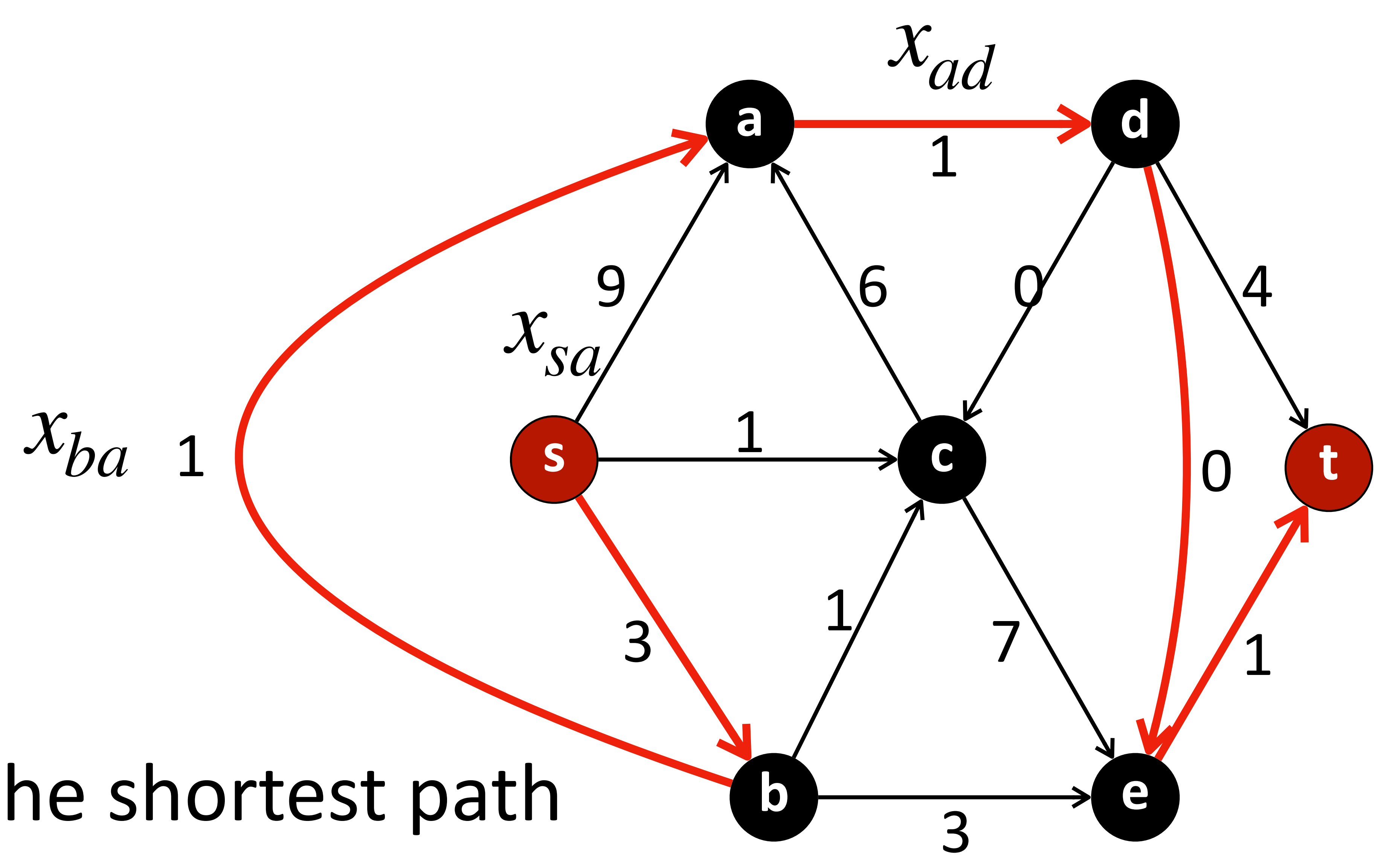
Decide if edge (u, v) is in the shortest path

$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.



$\min \Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$

For v on the shortest path,

$\Sigma_{(k,v)\in E} x_{kv} = 1$

$\Sigma_{(v,k)\in E} x_{vk} = 1$

For v **not** on the shortest path,

$\Sigma_{(k,v)\in E} x_{kv} = 0$

$\Sigma_{(v,k)\in E} x_{vk} = 0$

Decide if edge (u, v) is in the shortest path

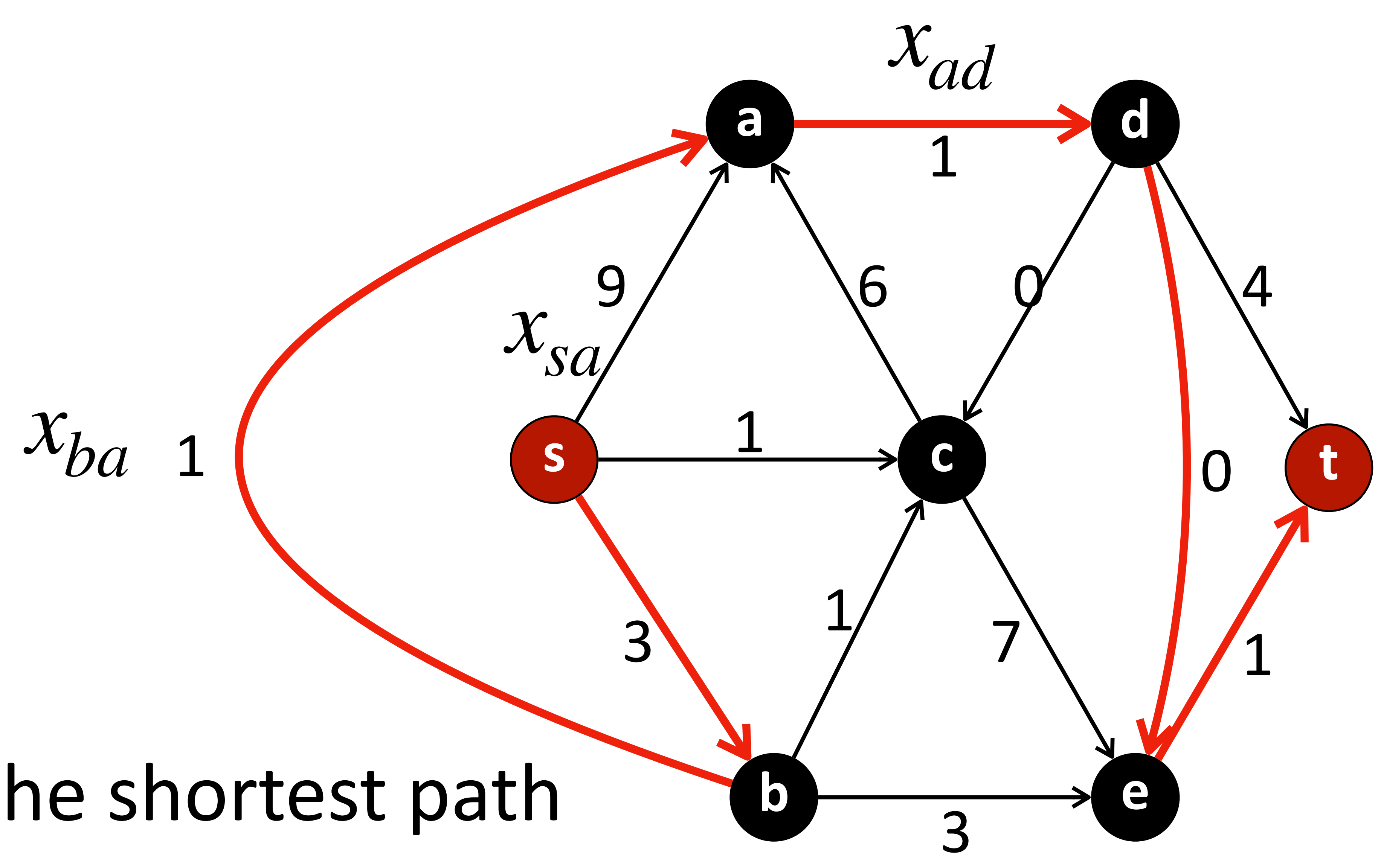
$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

$\min \Sigma_{(u,v)\in E} \ell_{uv} x_{uv}$



Decide if edge (u, v) is in the shortest path

$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

For v on the shortest path,

$\Sigma_{(k,v)\in E} x_{kv} = 1$

$\Sigma_{(v,k)\in E} x_{vk} = 1$

For v **not** on the shortest path,

$\Sigma_{(k,v)\in E} x_{kv} = 0$

$\Sigma_{(v,k)\in E} x_{vk} = 0$

For $v \in V \backslash \{s, t\}$,

$\Sigma_{(k,v)\in E} x_{kv} = \Sigma_{(v,k)\in E} x_{vk}$

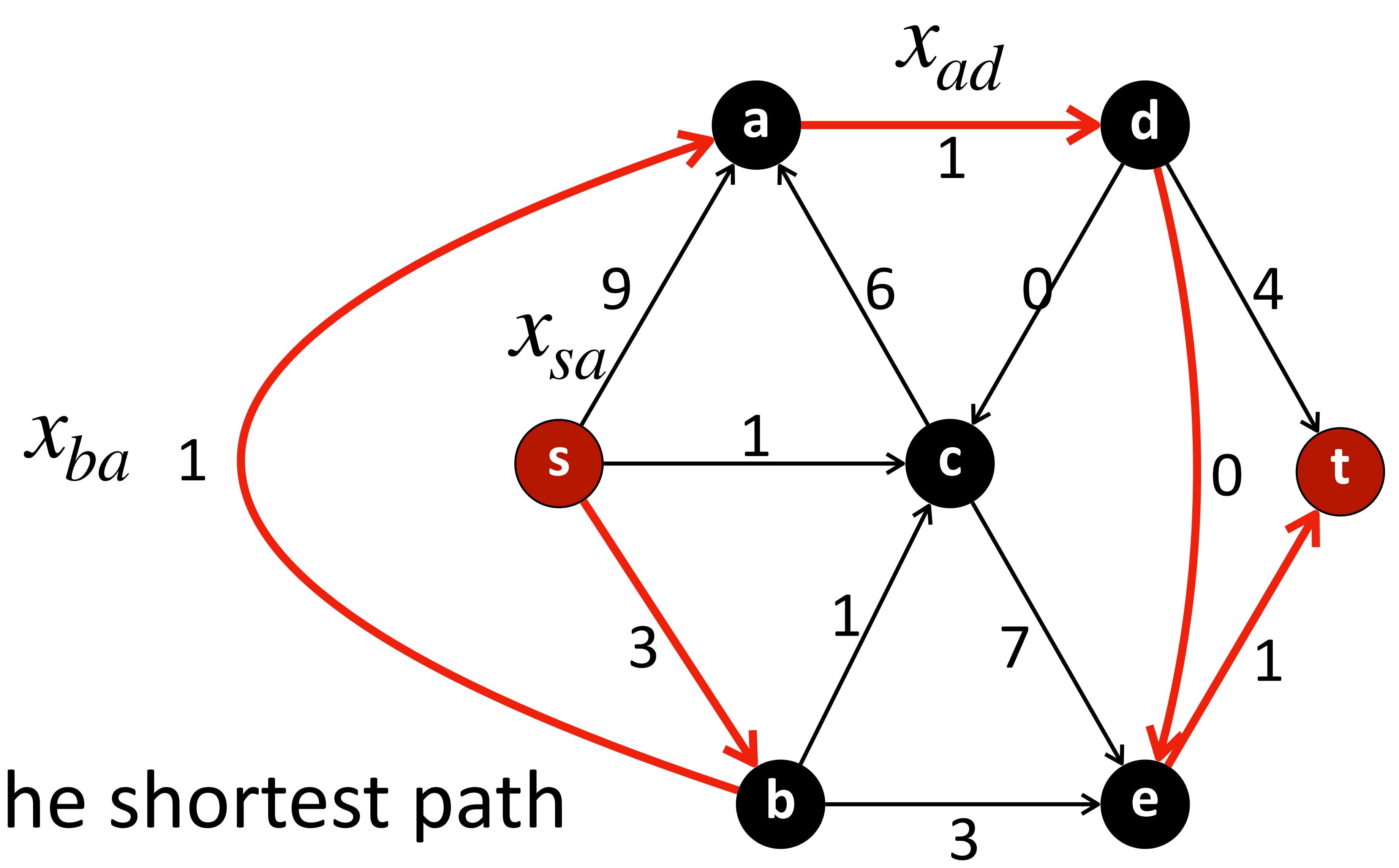
Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.

For s, $\Sigma_{(s,k)\in E} x_{sk} = 1$

For t, $\Sigma_{(k,t)\in E} x_{kt} = 1$

$\min \Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$



For v on the shortest path,
$\Sigma_{(k,v)\in E} x_{kv} = 1$
$\Sigma_{(v,k)\in E} x_{vk} = 1$

For v **not** on the shortest path,
$\Sigma_{(k,v)\in E} x_{kv} = 0$
$\Sigma_{(v,k)\in E} x_{vk} = 0$

For $v \in V \backslash \{s, t\}$,
$\Sigma_{(k,v)\in E} x_{kv} = \Sigma_{(v,k)\in E} x_{vk}$

Decide if edge (u, v) is in the shortest path

$x_{uv} = 1$ if (u, v) is in the shortest path

$x_{uv} = 0$ otherwise

Shortest paths

- Given a directed graph $G = (V, E)$, each edge (u, v) has a non-negative length ℓ_{uv}. We want to find a path from $s \in V$ to $t \in V$ with the shortest length.
- Variable: $x_{uv} = 1$ if the edge (u, v) is in the $s - t$ shortest path
- minimize $\Sigma_{(u,v)\in E}\, \ell_{uv} x_{uv}$

subject to $\Sigma_{(s,k)\in E}\, x_{sk} = 1$

$\Sigma_{(k,t)\in E}\, x_{kt} = 1$

$\Sigma_{(k,v)\in E}\, x_{kv} - \Sigma_{(v,k)\in E}\, x_{vk} = 0$ for all $v \in V \setminus \{s, t\}$

$x_{uv} \in \{0,1\}$ for all $u, v \in V$

Shortest paths

- Sometimes, the constraints are not explicitly stated and need to be figured out by analyzing the desired solution's property

- Trick:
  - There are two types of *vertices*, depending on if it is on the shortest path from s to t
  - Different types vertices have different characterizations $\rightarrow$ observe the property and make a set of constraints for any vertex v that does not rely on whether v is in the shortest path

Outline

- More modeling optimization problems to (integer) programming problems
  - **Set cover**
  - **Shortest paths**
  - **Traveling Salesperson Problem**

- LP relaxation and upper/lower bound

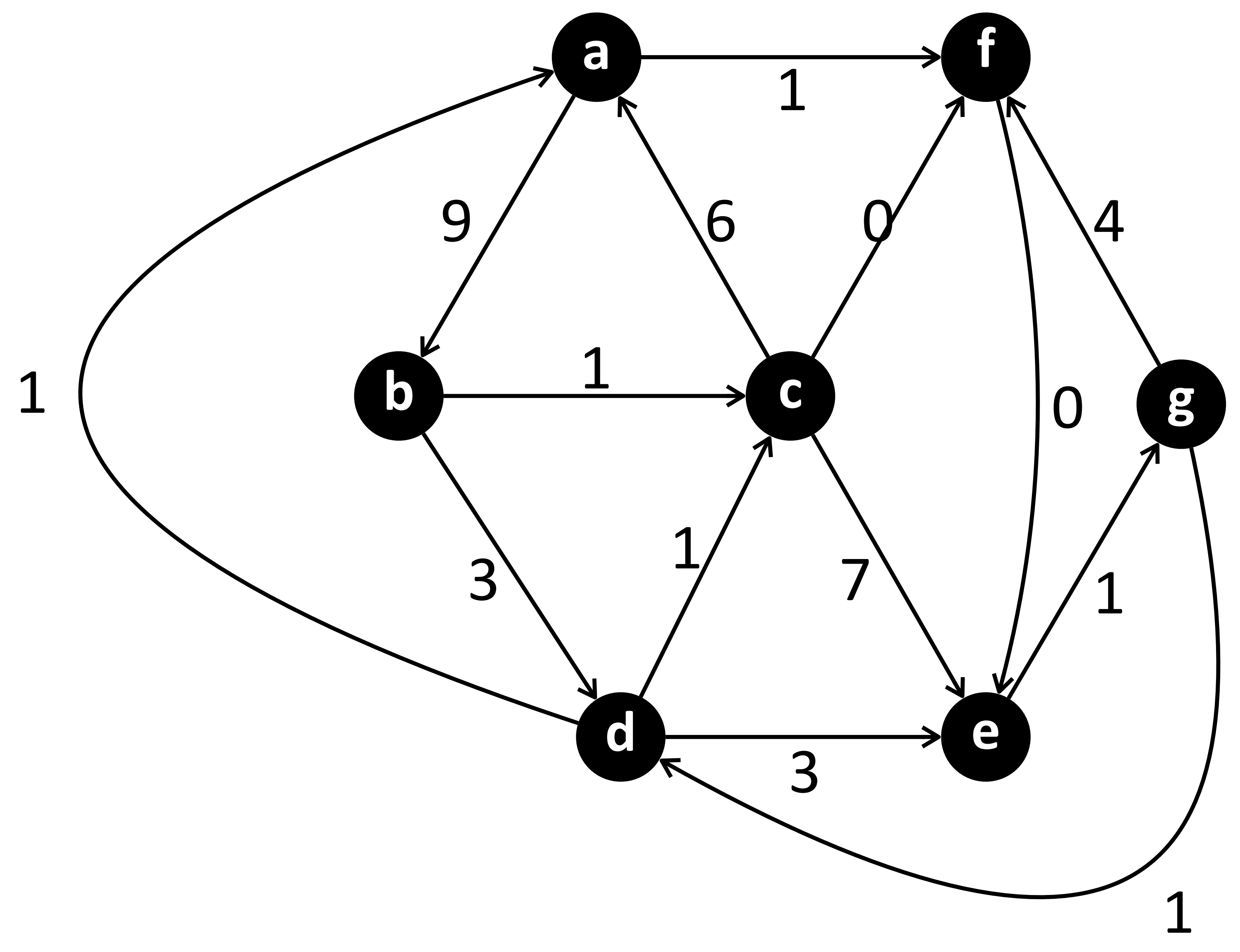
- Solving ILP: Branch and bound method

Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

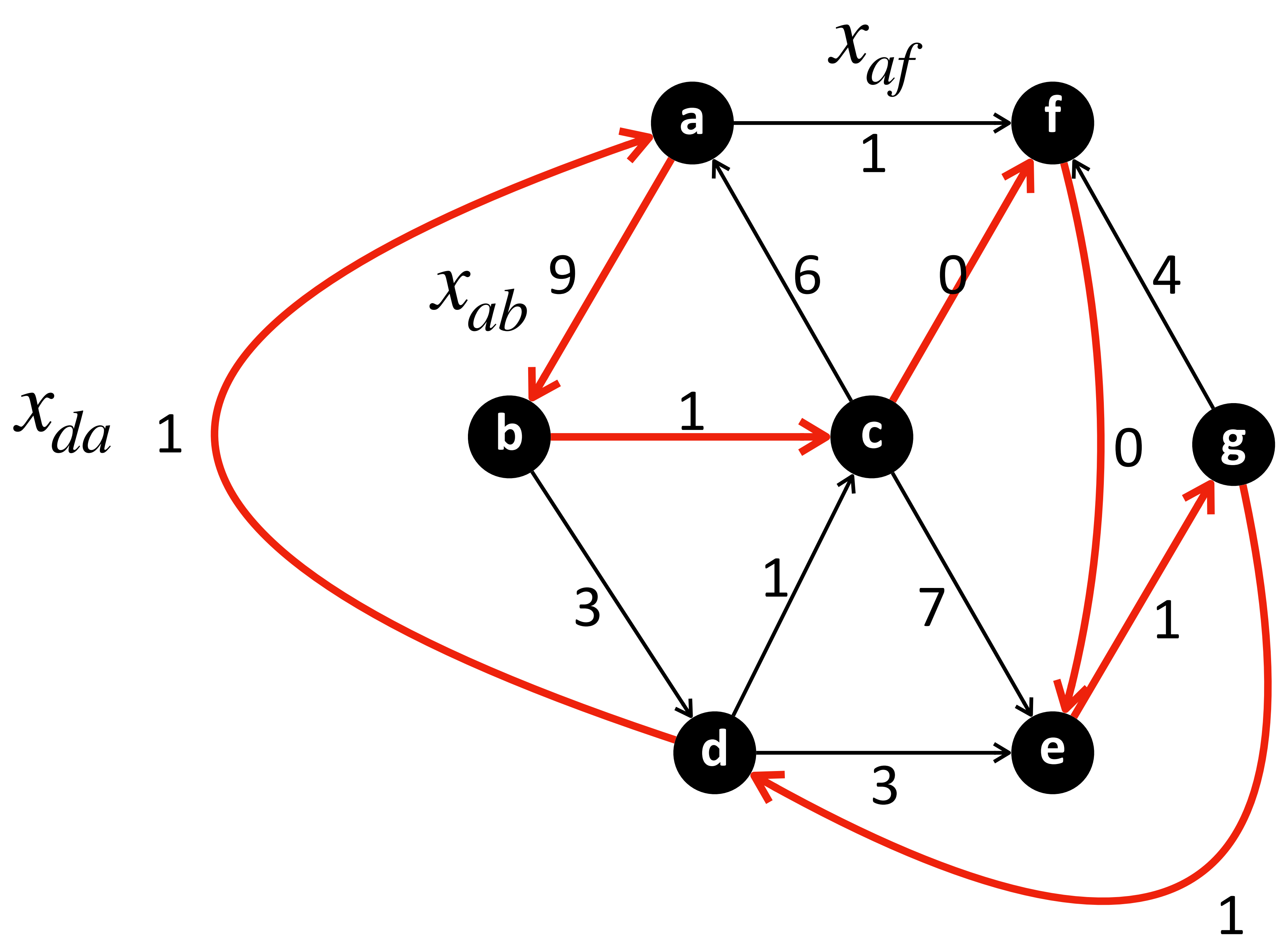
Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.



Traveling Salesperson Problem

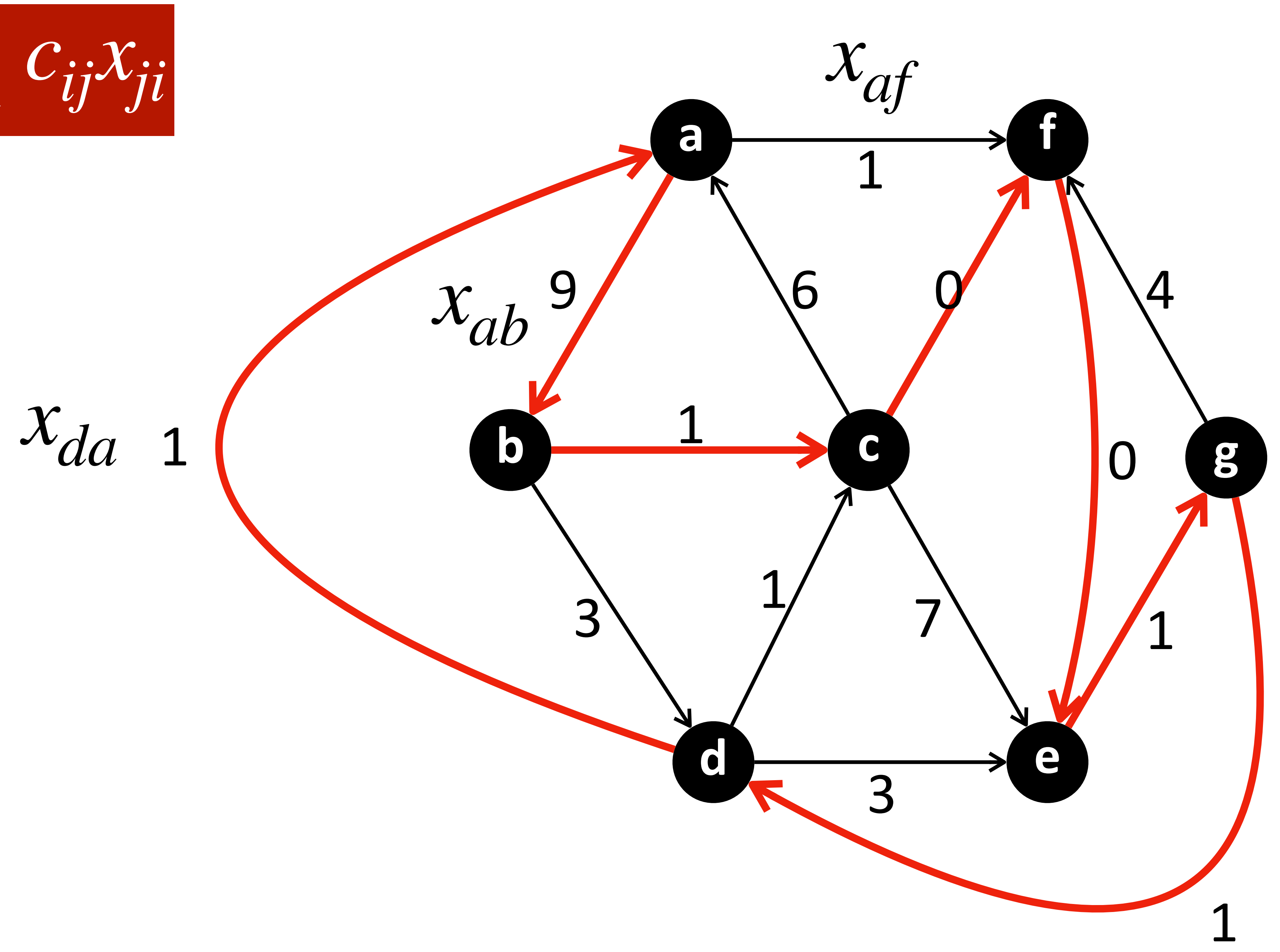
- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.



Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

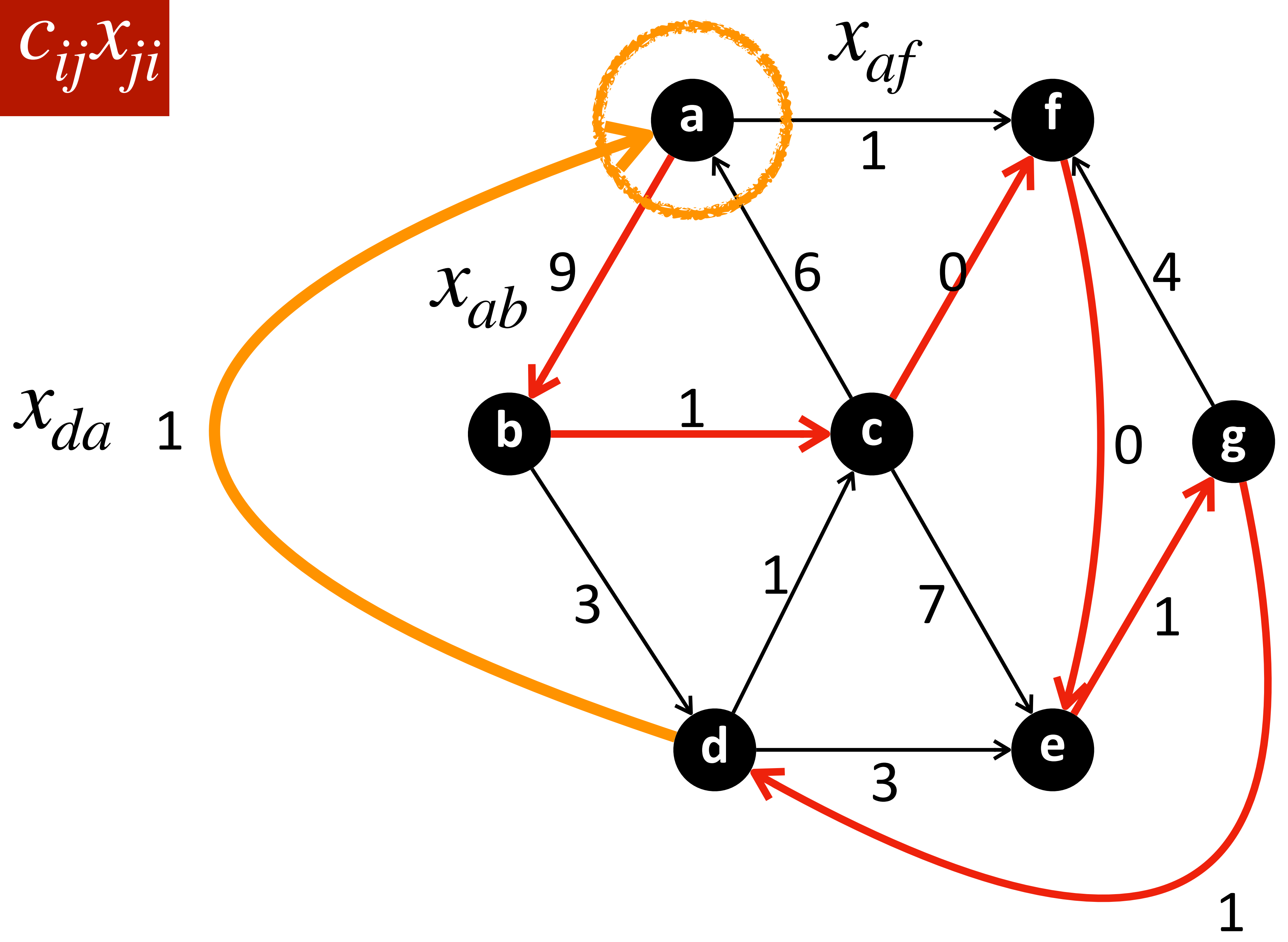
$$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$$



Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$$

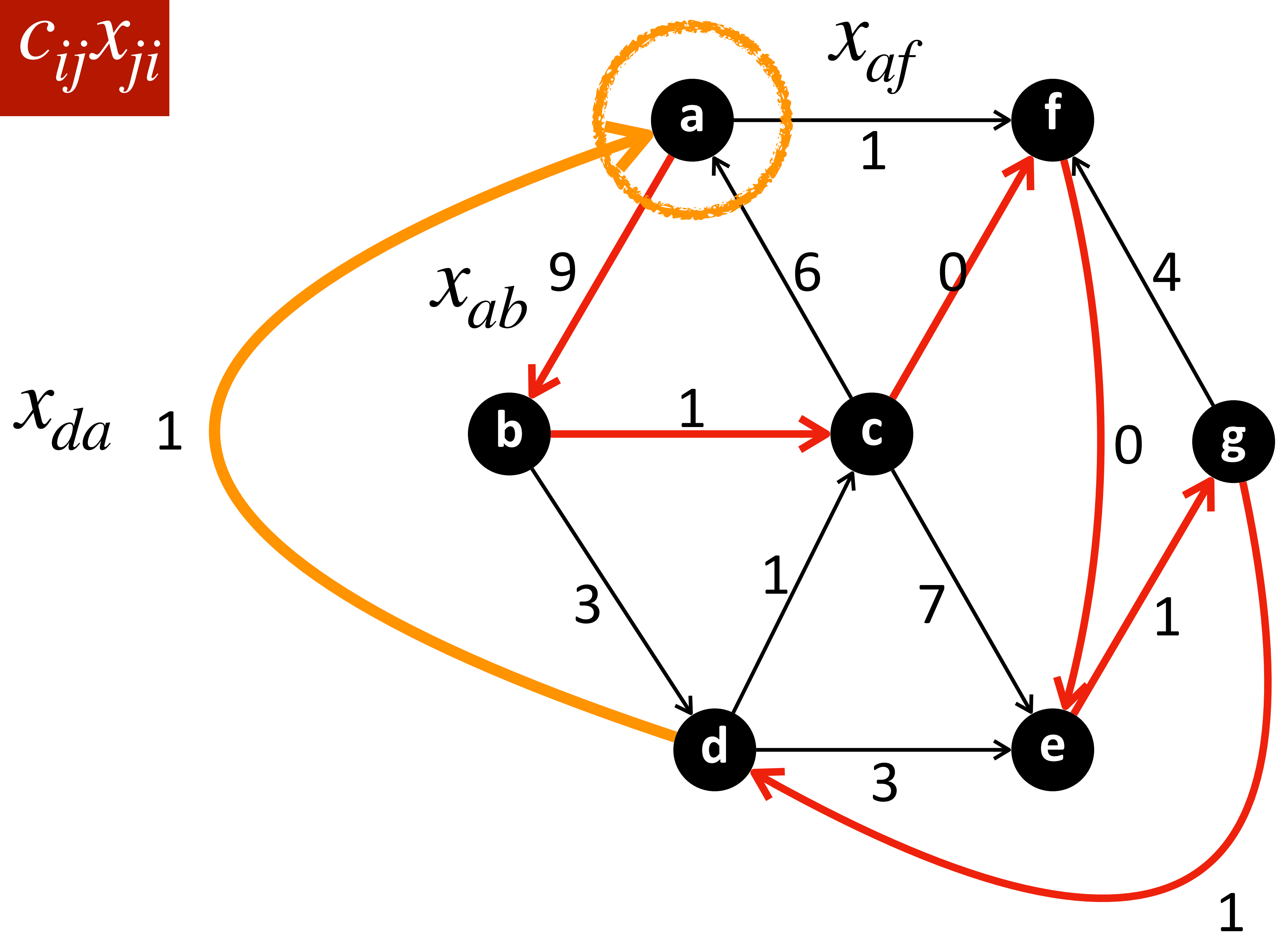


Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$

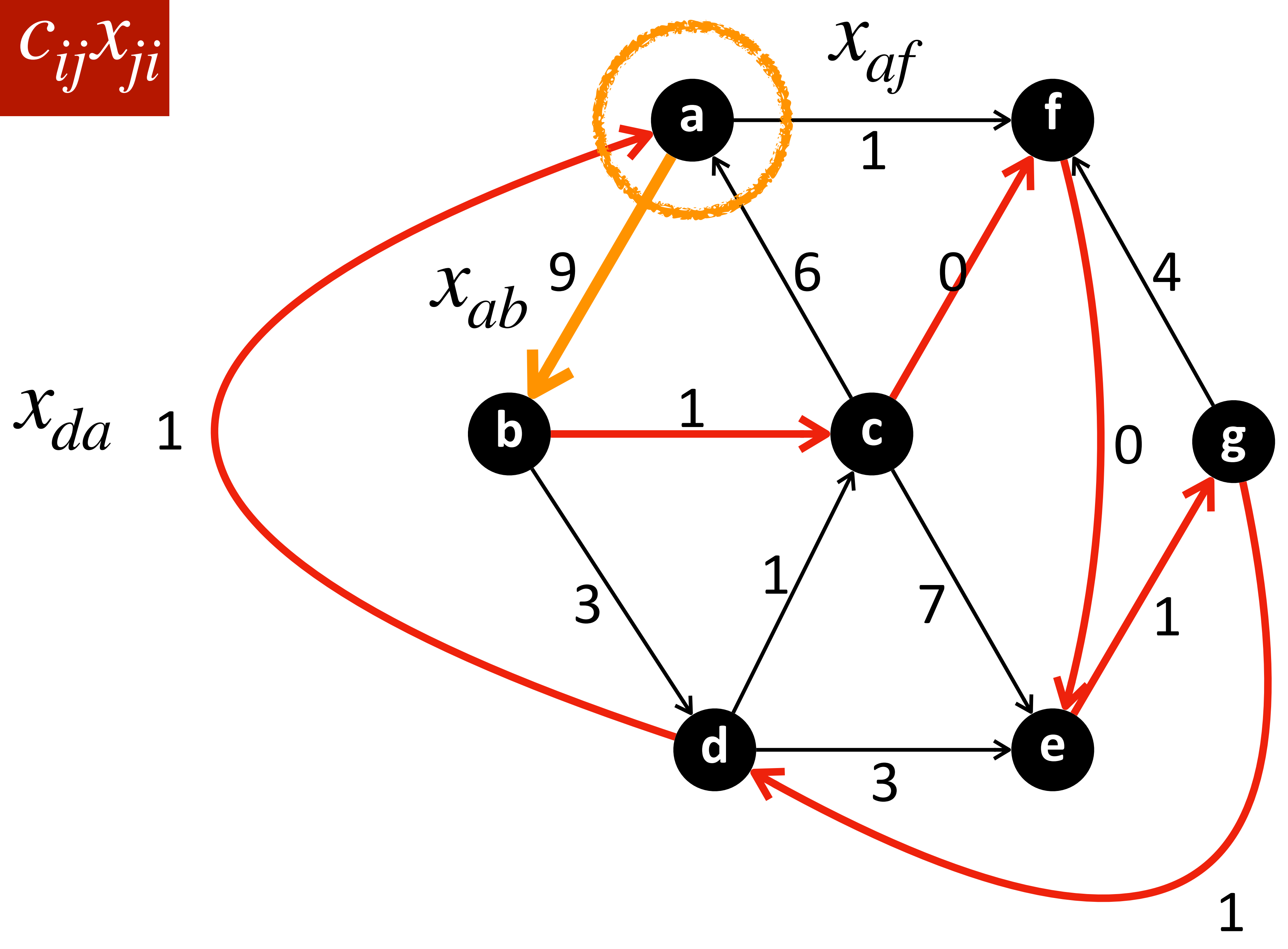
For every city i, $\Sigma_{j \neq i} x_{ji} = 1$



Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$$

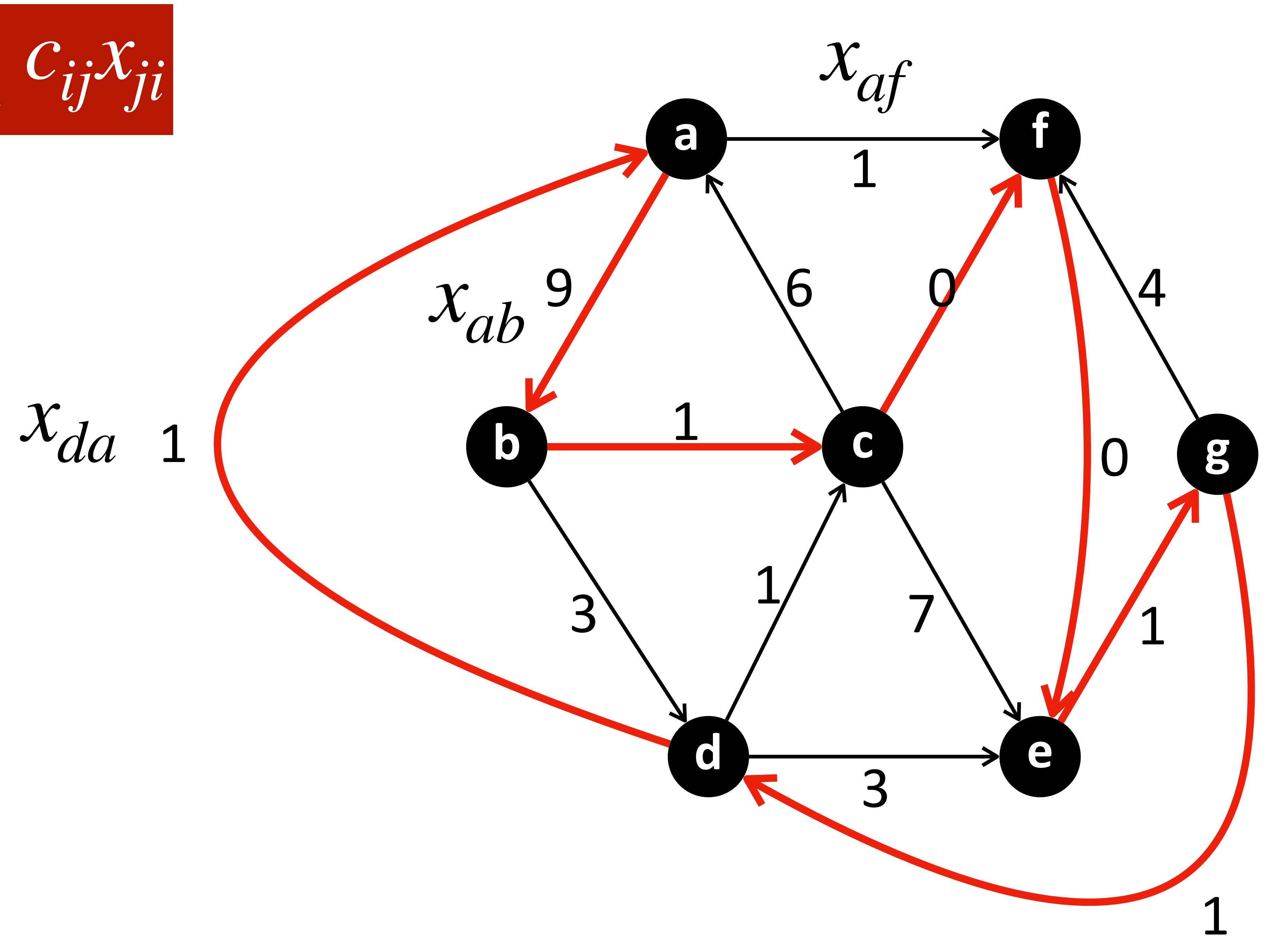


For every city i, $\Sigma_{j \neq i} x_{ij} = 1$

Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ji}$



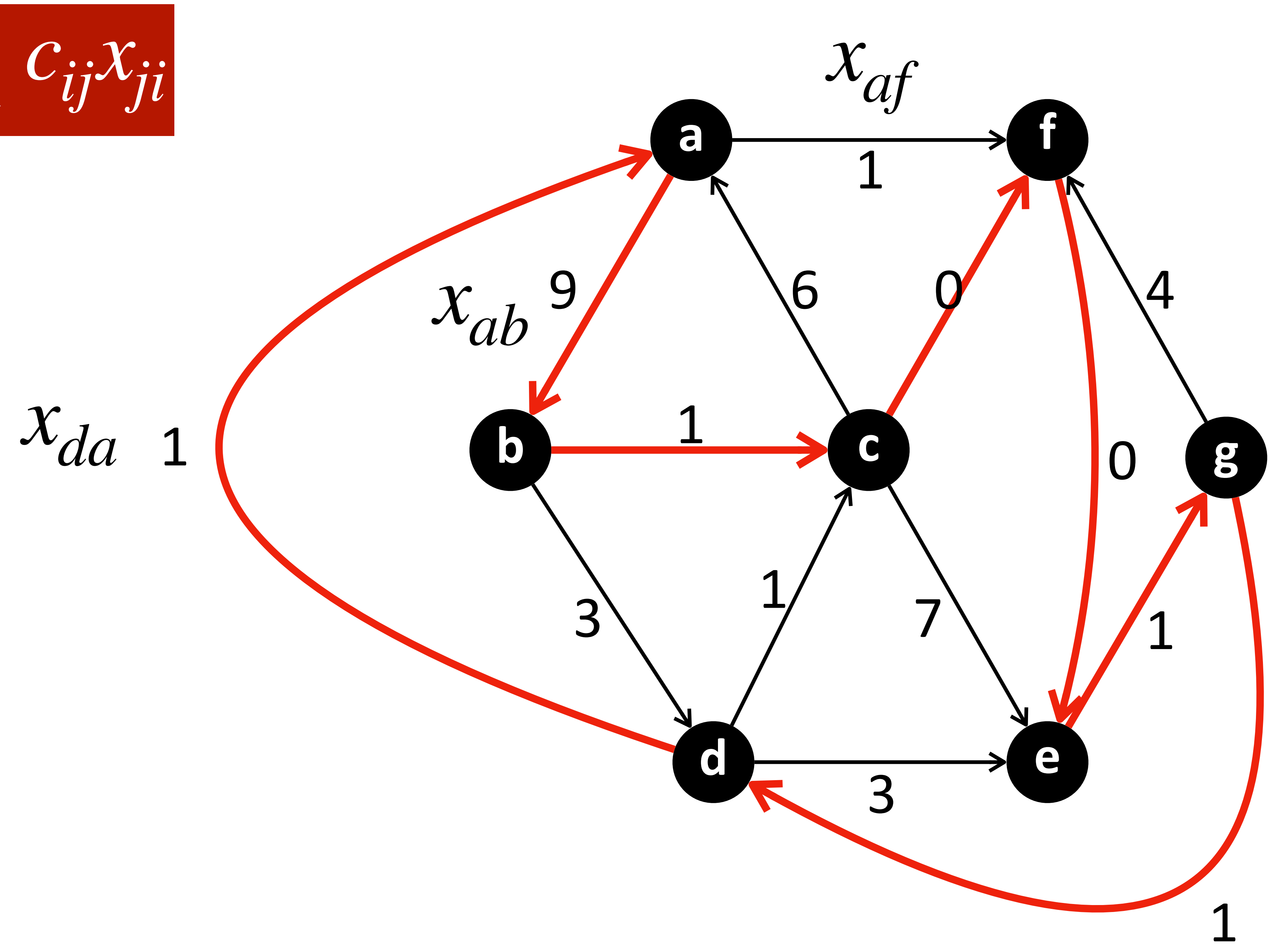
For every city i, $\sum_{j \neq i} x_{ji} = 1$

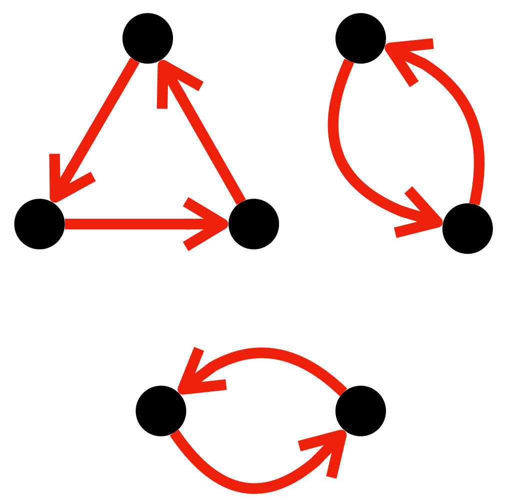
For every city i, $\sum_{j \neq i} x_{ij} = 1$

Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$$





For every city i, $\Sigma_{j \neq i} x_{ji} = 1$

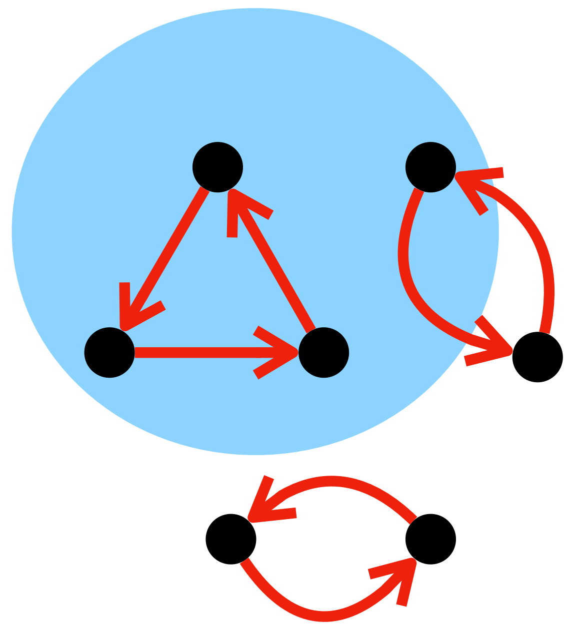
For every city i, $\Sigma_{j \neq i} x_{ij} = 1$

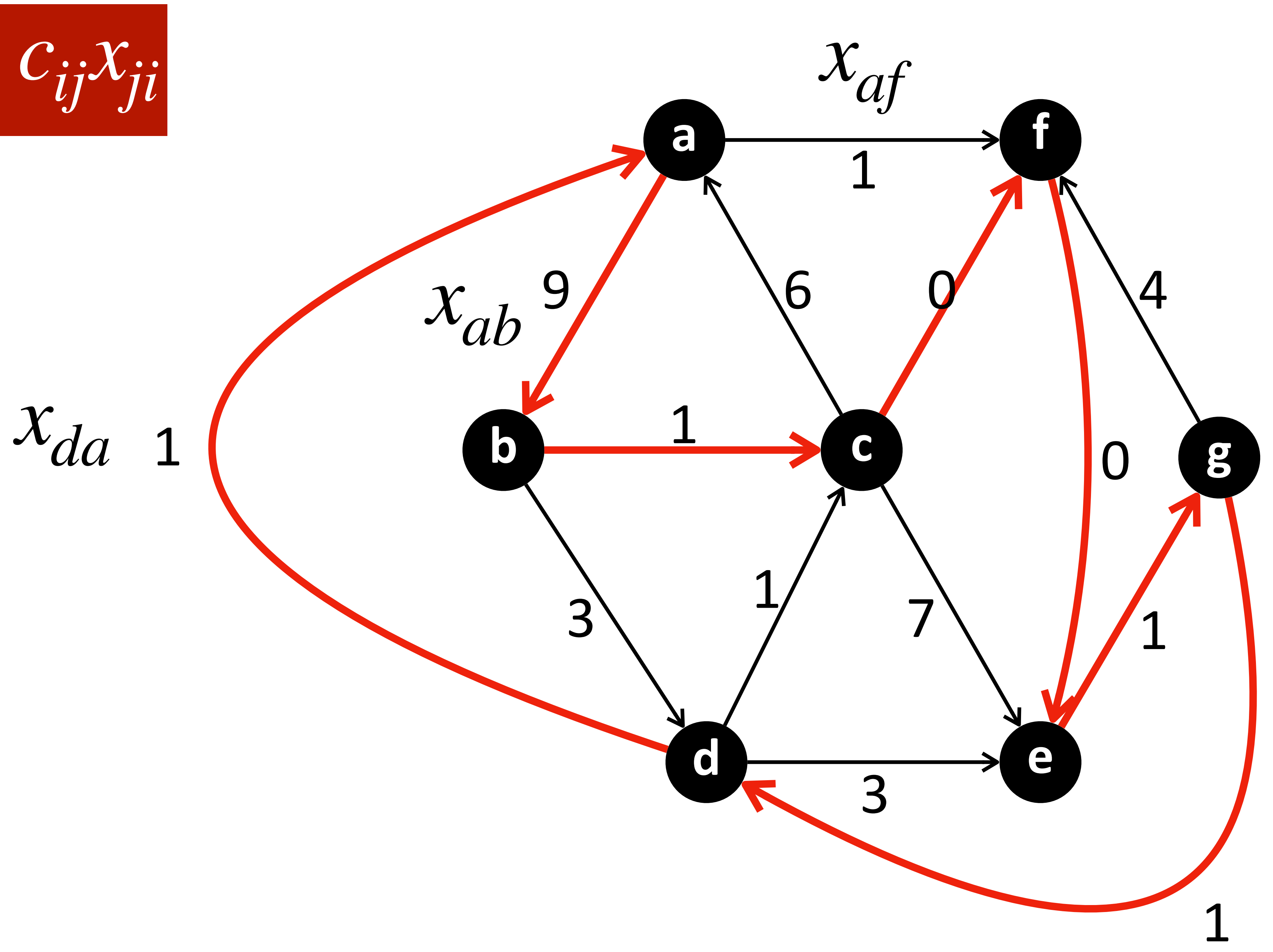
Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$

any non-empty subset of cities S





For every city i, $\Sigma_{j \neq i} x_{ji} = 1$

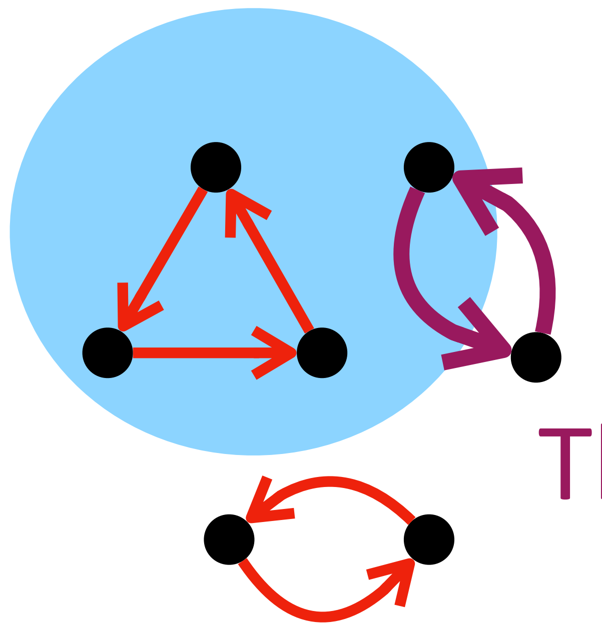
For every city i, $\Sigma_{j \neq i} x_{ij} = 1$

Traveling Salesperson Problem

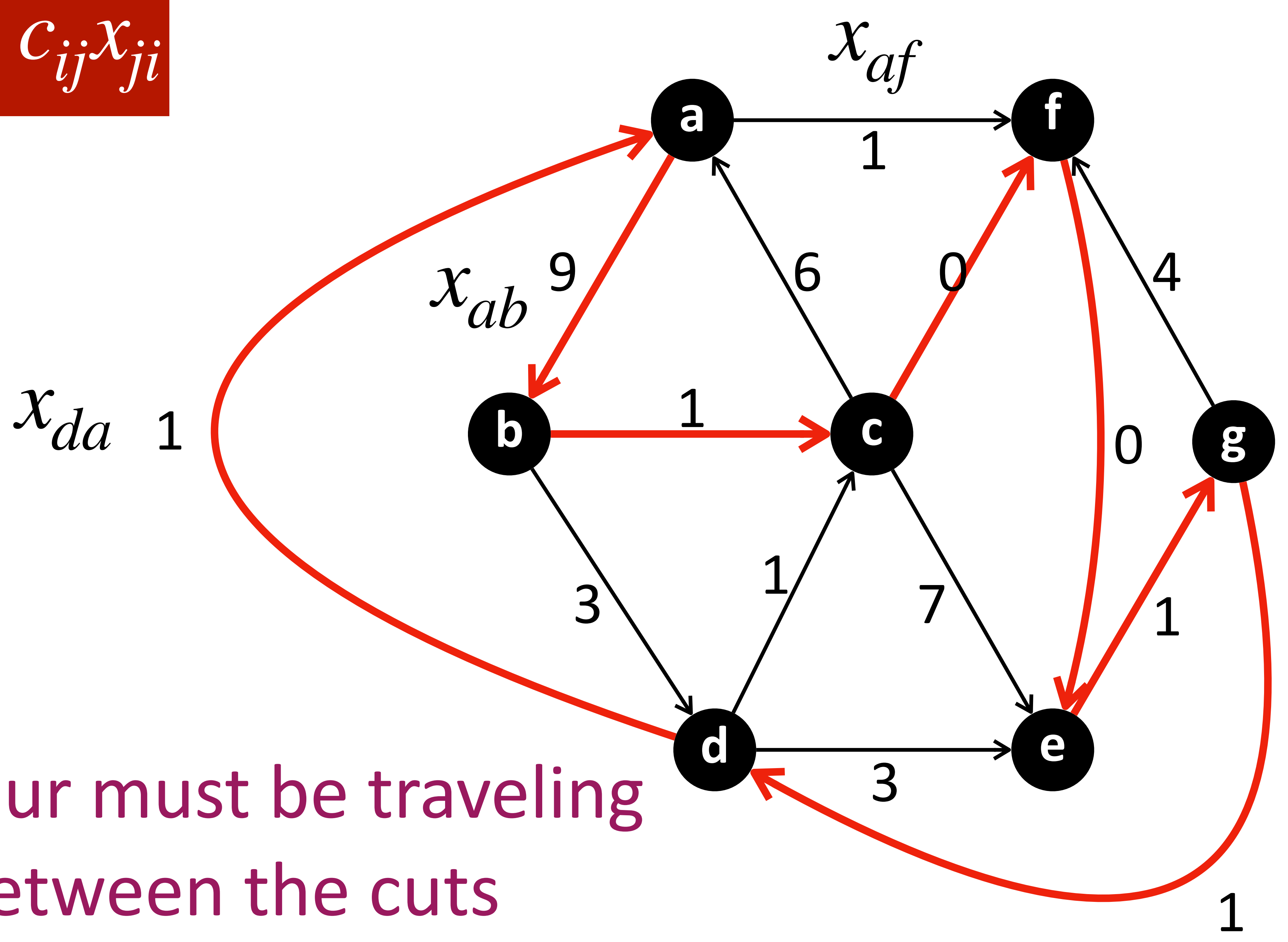
- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

$$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$$

any non-empty subset of cities S



The tour must be traveling between the cuts



For every city i, $\Sigma_{j \neq i} x_{ji} = 1$

For every city i, $\Sigma_{j \neq i} x_{ij} = 1$

For $S \subsetneq N$ and $S \neq \phi$, $\Sigma_{i \in S} \Sigma_{j \notin S} x_{ij} \geq 1$

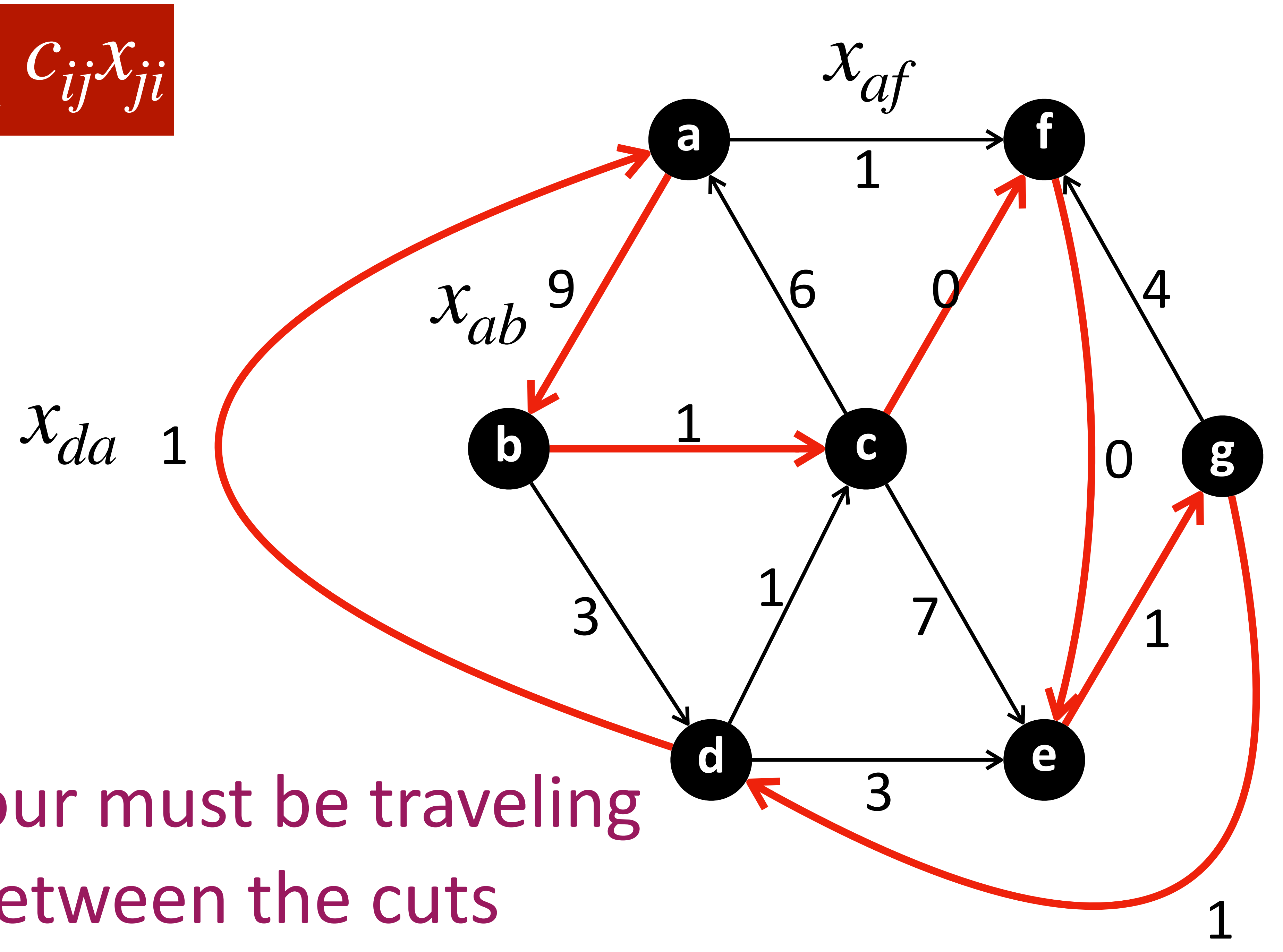
Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

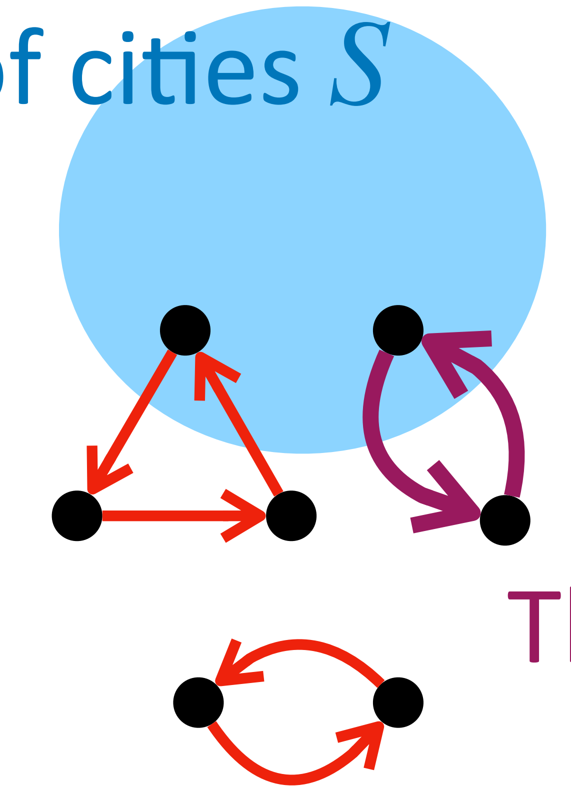
$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$

For every city i, $\Sigma_{j \neq i} x_{ji} = 1$

For every city i, $\Sigma_{j \neq i} x_{ij} = 1$



any non-empty subset of cities S



The tour must be traveling between the cuts

For $S \subsetneq N$ and $S \neq \phi$, $\Sigma_{i \in S} \Sigma_{j \notin S} x_{ij} \geq 1$

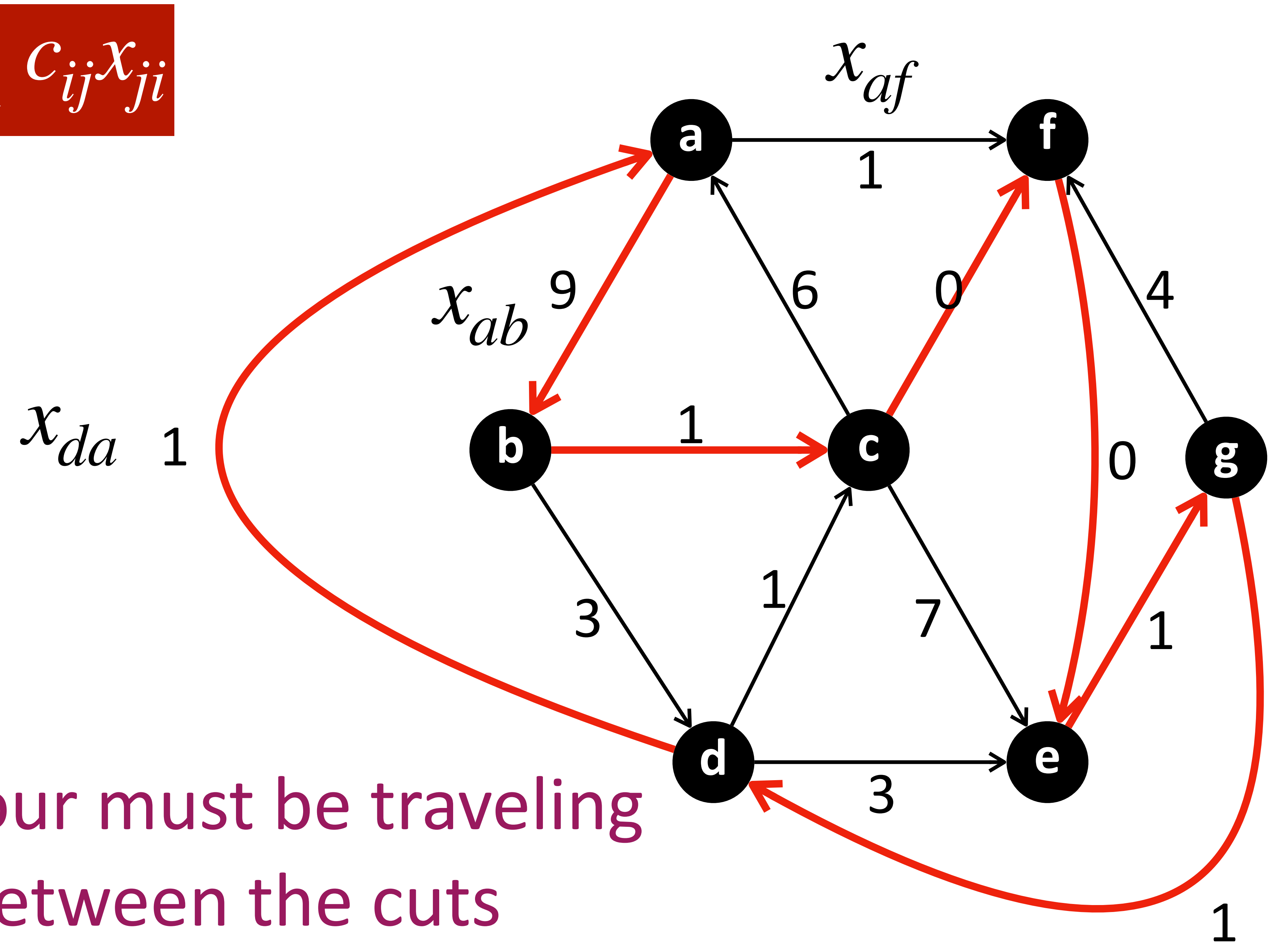
Traveling Salesperson Problem

- A salesperson must visit each of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.

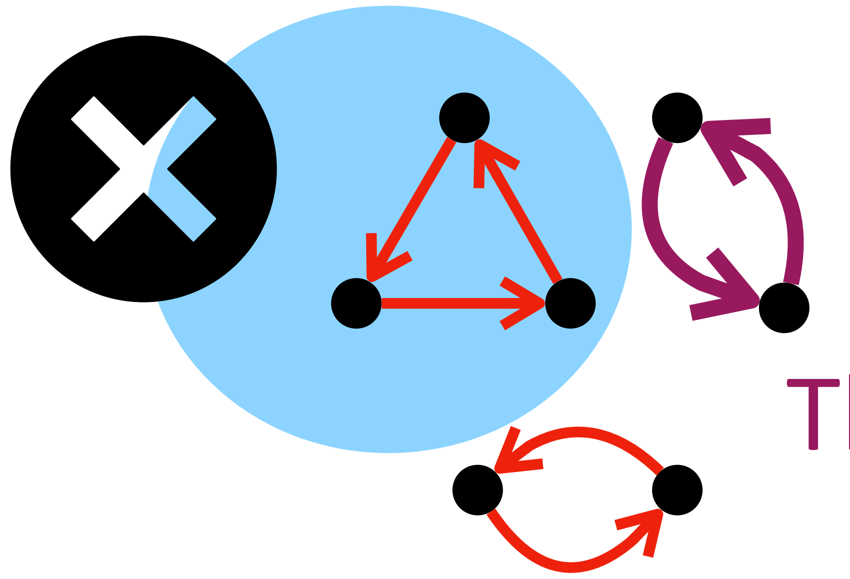
$\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ji}$

For every city i, $\Sigma_{j \neq i} x_{ji} = 1$

For every city i, $\Sigma_{j \neq i} x_{ij} = 1$



any non-empty subset of cities S



The tour must be traveling between the cuts

For $S \subsetneq N$ and $S \neq \phi$, $\Sigma_{i \in S} \Sigma_{j \notin S} x_{ij} \geq 1$

Traveling Salesperson Problem

- A salesperson must visit each of a set N of n cities exactly once and then return to the starting point. The time taken to travel from city i to city j is c_{ij}. Find the order in which the salesperson should make their tour so as to finish as quickly as possible.
  - Decision: which edges to take, and the order of taking the edges
    - $x_{ij} = 1$ if the salesperson goes directly from town i to town j, and $x_{ij} = 0$ otherwise
  - Objective: $\min \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ij}$
  - Constraint: Each city is visited exactly once
    - Leave city i once $\Sigma_{j \neq i} x_{ij} = 1$ for $i = 1, \cdots, n$
    - Arrive city i once $\Sigma_{j \neq i} x_{ji} = 1$ for $i = 1, \cdots, n$
    - For $S \subsetneq N$ and $S \neq \phi$, $\Sigma_{i \in S} \Sigma_{j \notin S} x_{ij} \geq 1$
    - $x_{ij} \in \{0,1\}$ for $i = 1, \cdots, n, j = 1, \cdots, n$

Traveling Salesperson Problem

- Find the ordering is automatically done by “choosing a cycle”

- Sometimes, the naive formulated constraints are only necessary conditions, but not sufficient
  $\Rightarrow$ find alternative formulations to rule out the exceptions

Outline

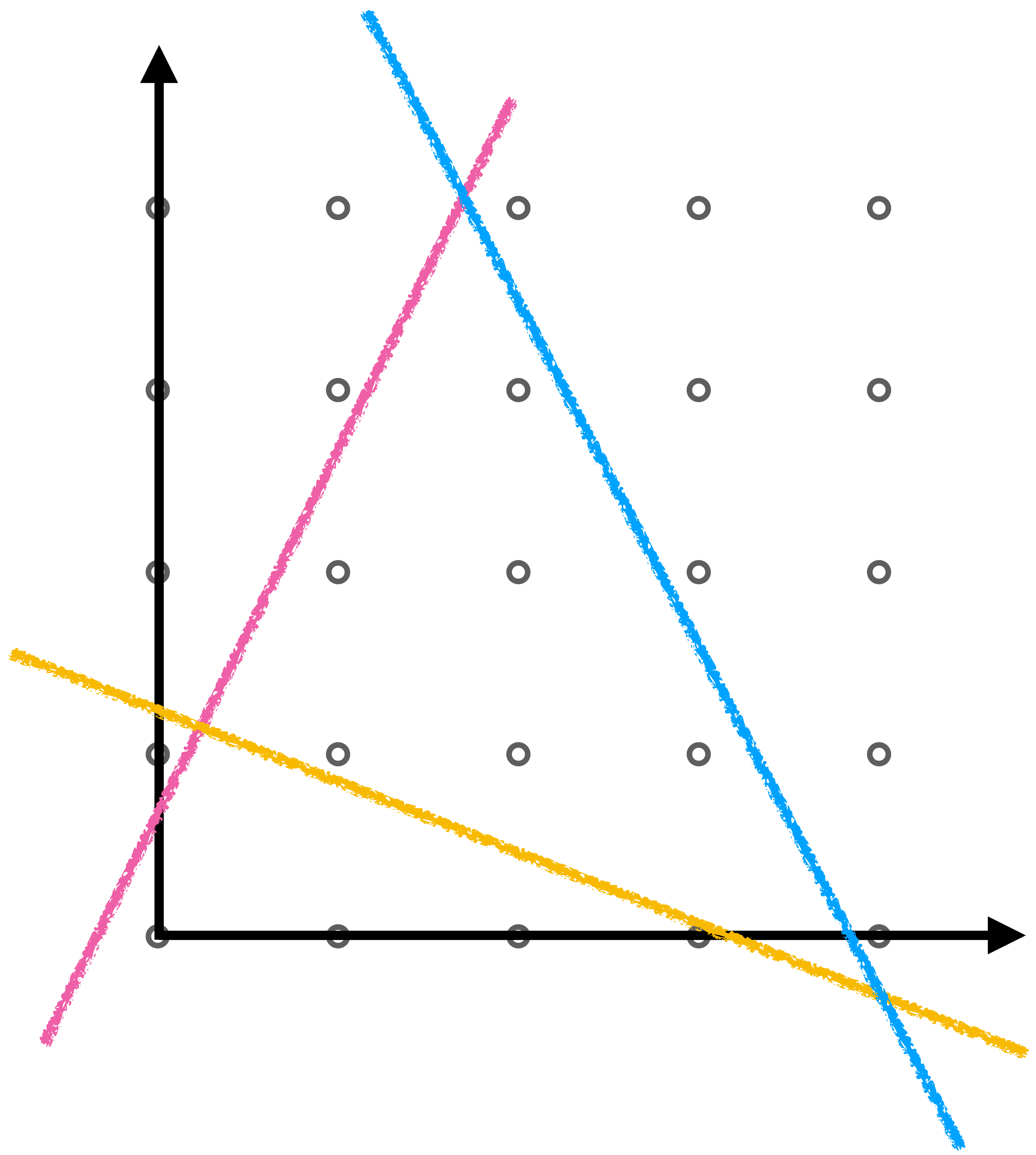
- More modeling optimization problems to (integer) programming problems
  - **Set cover**
  - **Shortest paths**
  - **Traveling Salesperson Problem**
- LP relaxation and upper/lower bound
- Solving ILP: Branch and bound method

Different Linear Programming Problems

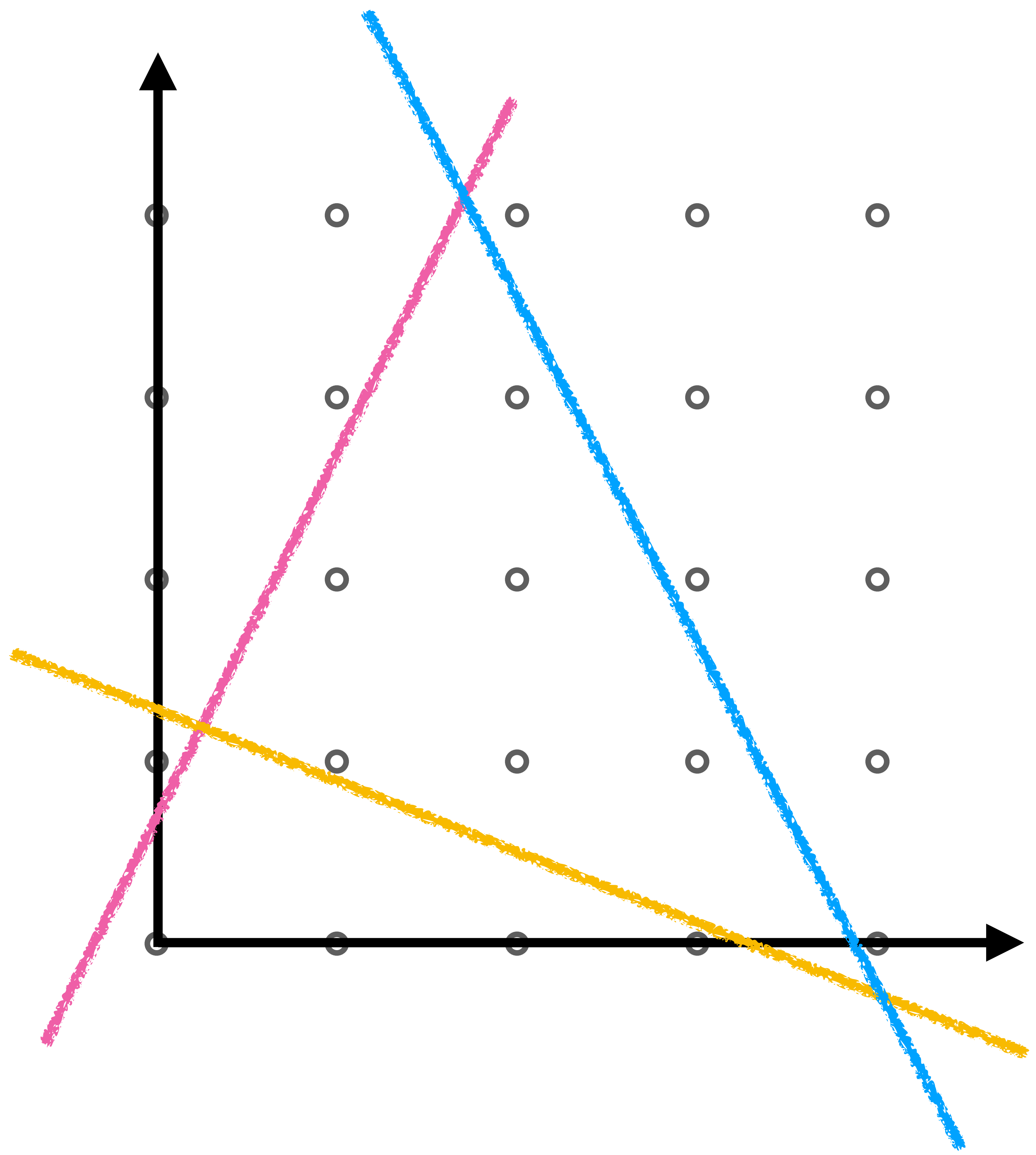
- Linear programming
  - decisions can be real numbers
- Integer Linear programming
  - decisions must be integral

Different Linear Programming Problems

- Linear programming
  - decisions can be real numbers

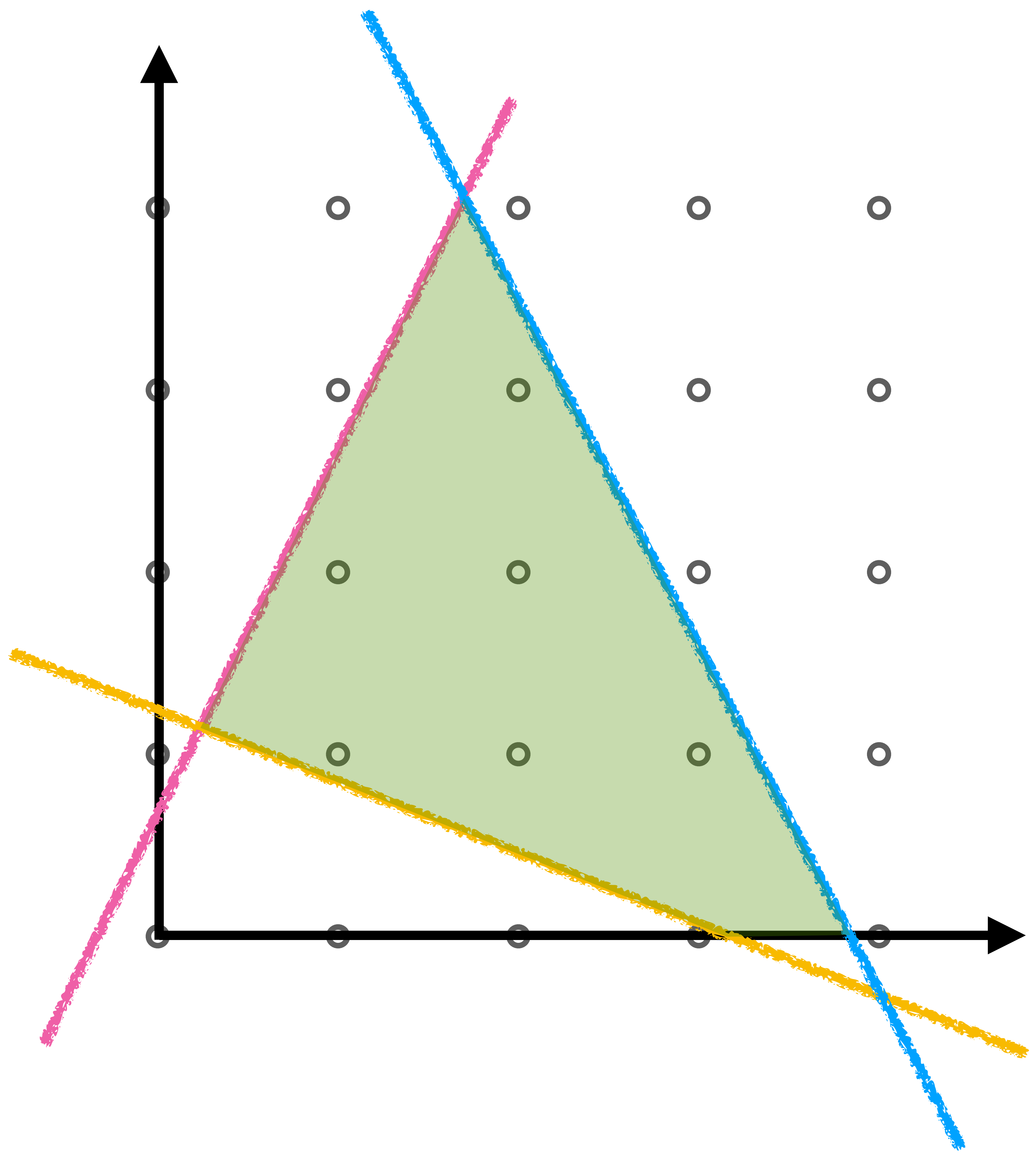


- Integer Linear programming
  - decisions must be integral

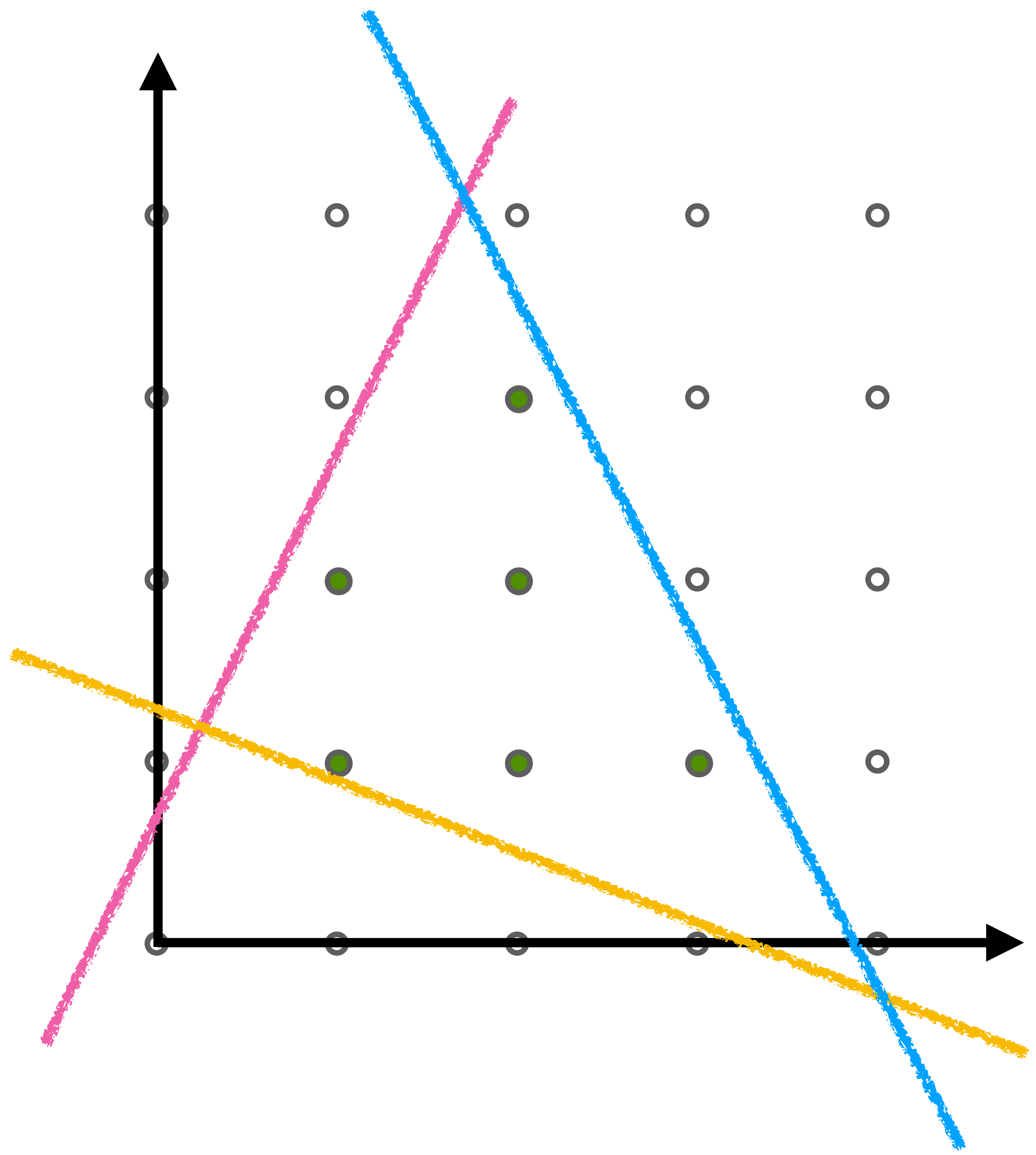


Different Linear Programming Problems

- Linear programming
  - decisions can be real numbers



- Integer Linear programming
  - decisions must be integral



LP relaxation

LP relaxation

- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$$3x - 2y \geq -4$$

$$x, y \in \mathbb{N}$$

LP relaxation

- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \geq 0$

- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \in \mathbb{N}$

LP relaxation

- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \geq 0$

- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \in \mathbb{N}$

Relax the restriction that x and y should be integral

LP relaxation

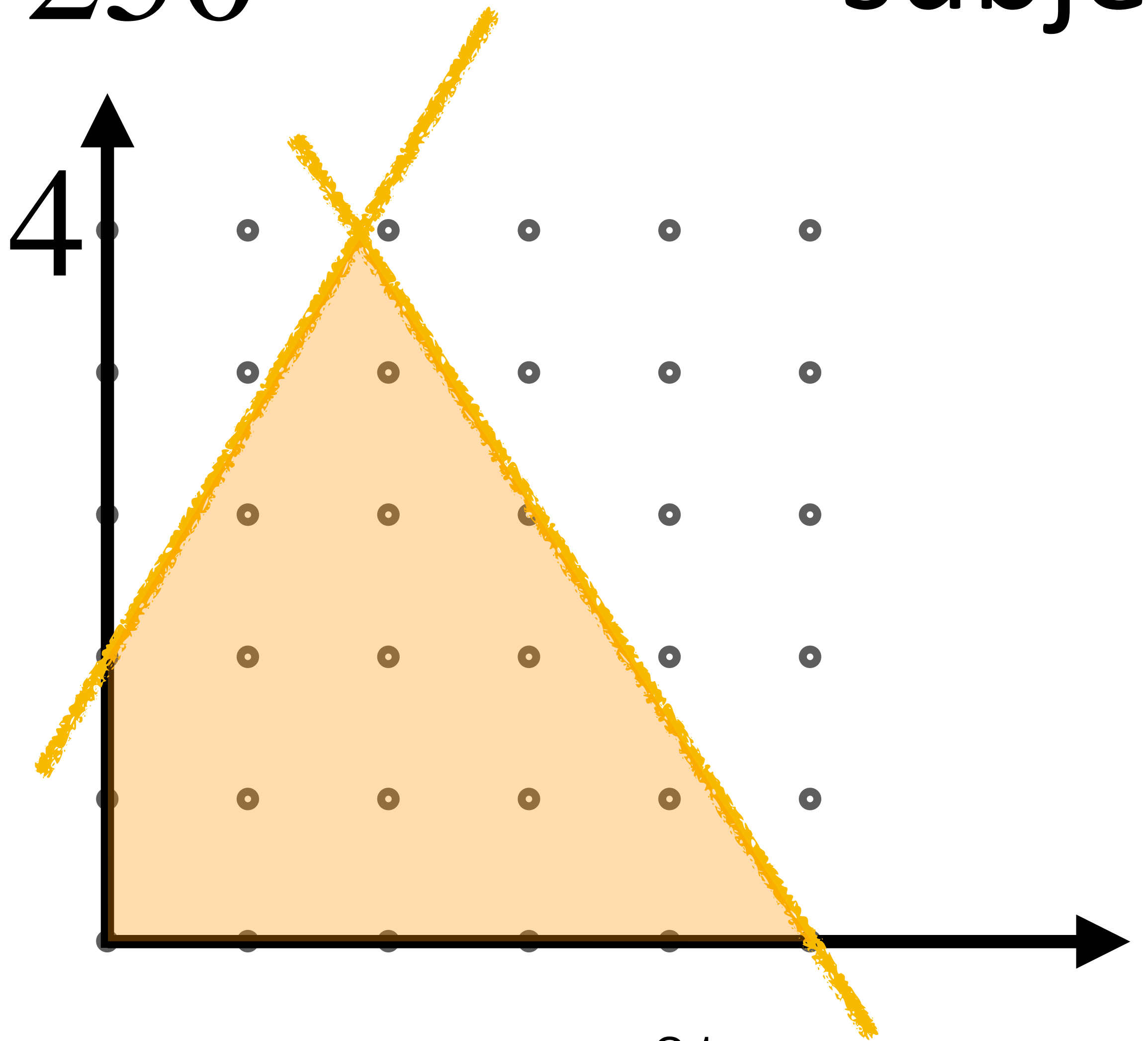
- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \geq 0$



- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \in \mathbb{N}$

LP relaxation

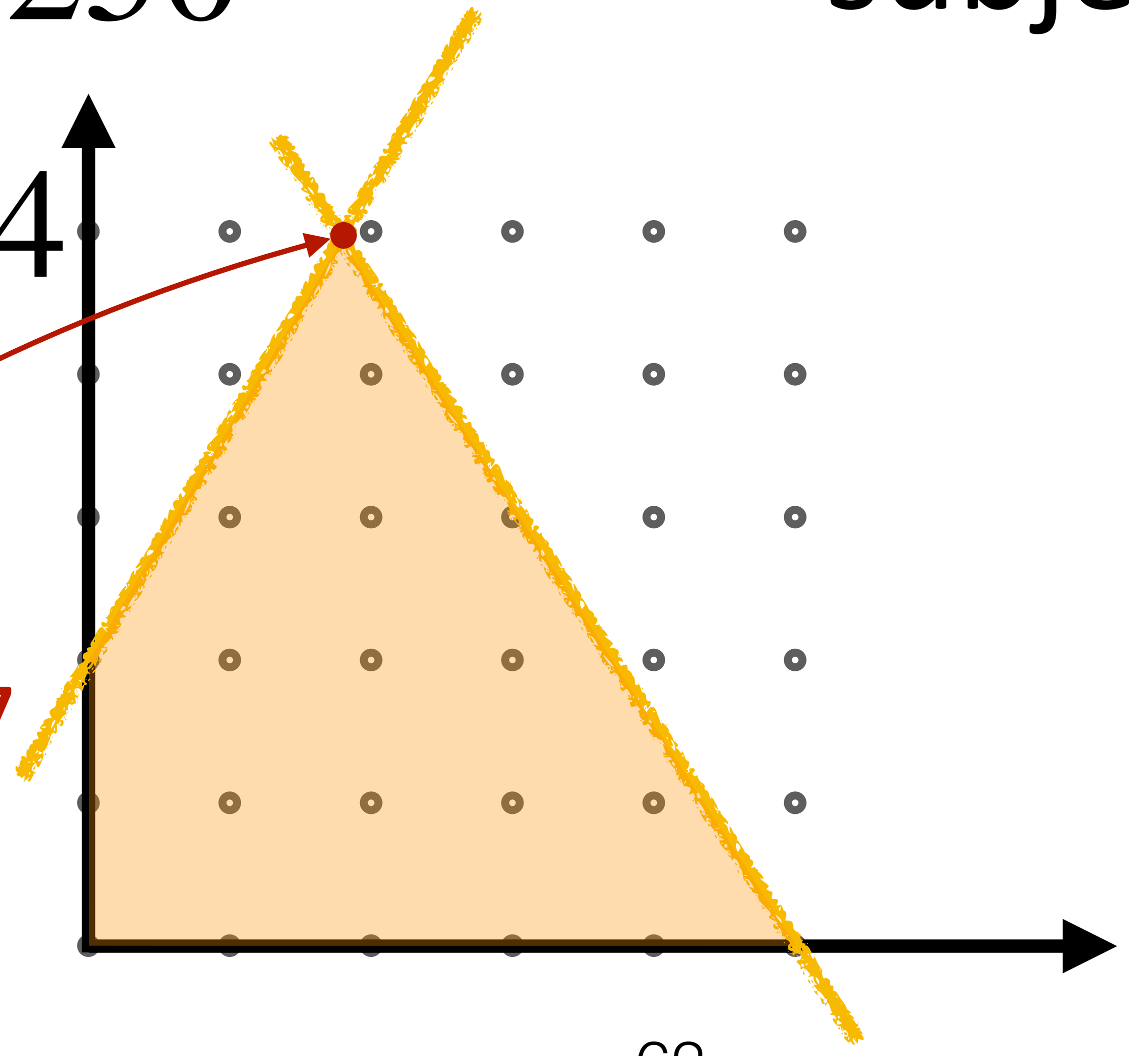
- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$$3x - 2y \geq -4$$

$$x, y \geq 0$$



fractional optimum 254.92227

$$x = \frac{376}{193}, y = \frac{950}{193}$$

- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$$3x - 2y \geq -4$$

$$x, y \in \mathbb{N}$$

LP relaxation

- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

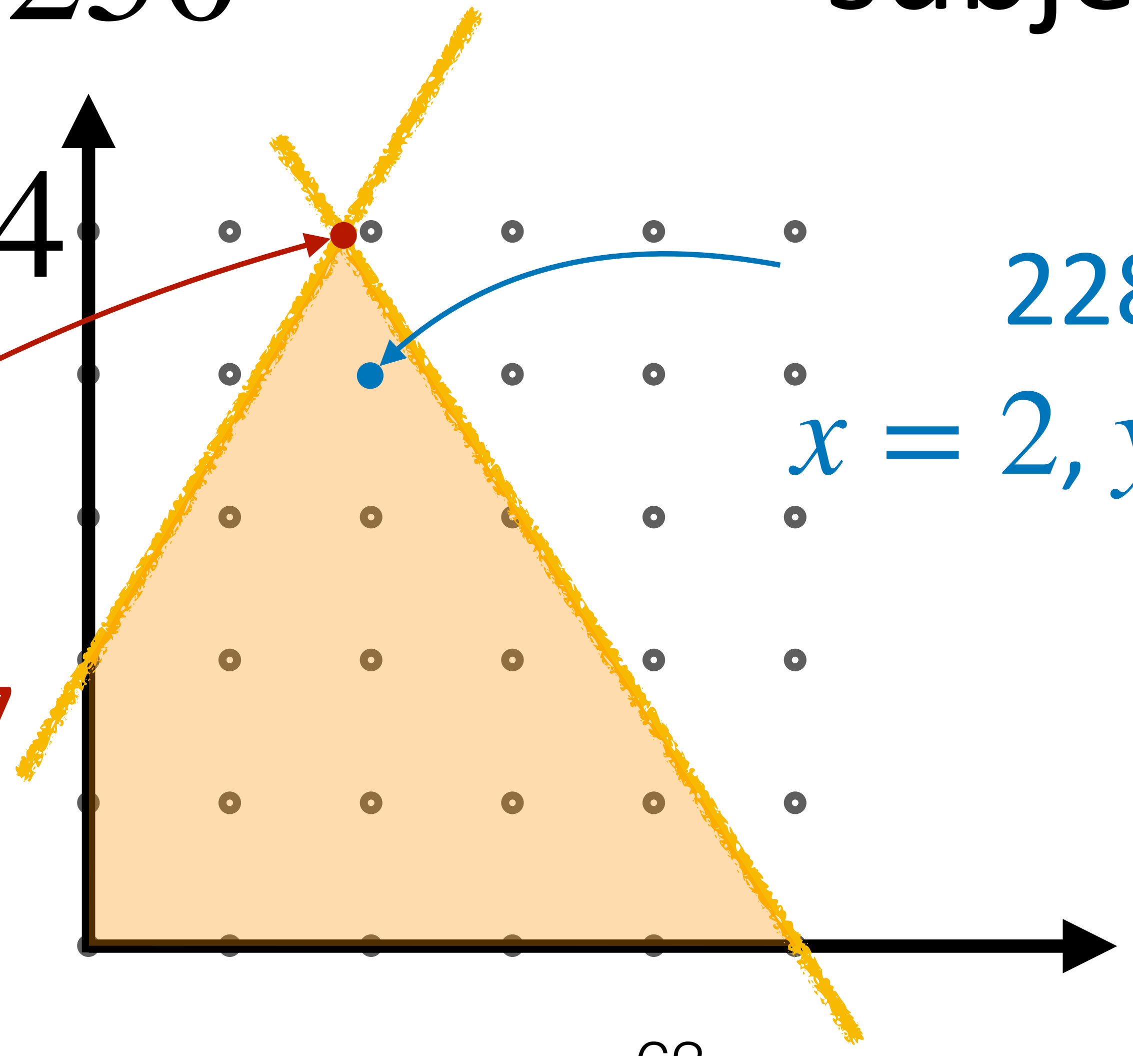
subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \geq 0$

fractional optimum 254.92227

$x = \frac{376}{193}, y = \frac{950}{193}$



228

$x = 2, y = 4$

- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \in \mathbb{N}$

LP relaxation

- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \geq 0$

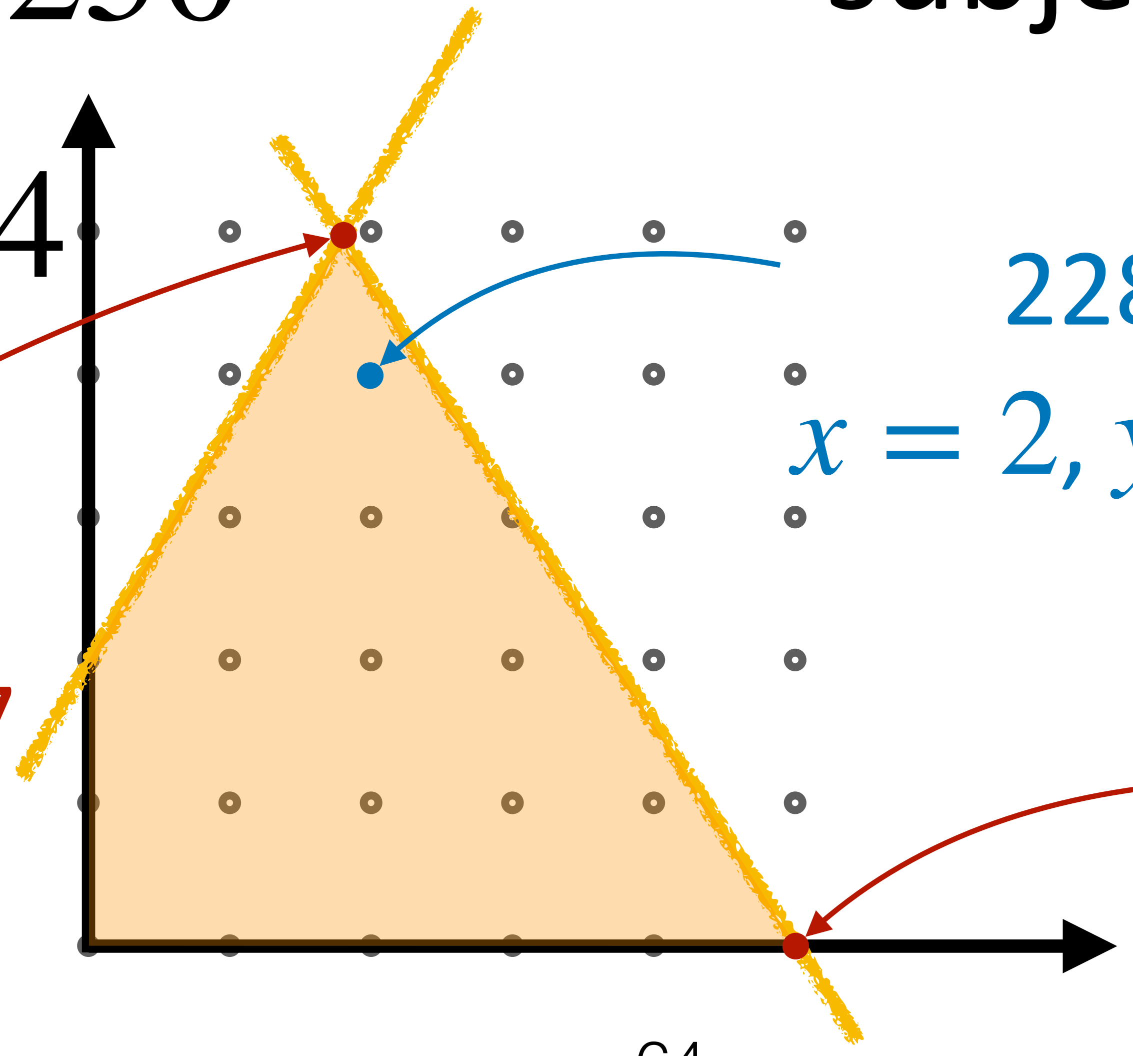
- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$3x - 2y \geq -4$

$x, y \in \mathbb{N}$



fractional optimum 254.92227

$x = \frac{376}{193}, y = \frac{950}{193}$

228

$x = 2, y = 4$

integral optimum 250

$x = 5, y = 0$

LP relaxation

- Linear programming
  - decisions can be real numbers

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

$$3x - 2y \geq -4$$

$$x, y \geq 0$$

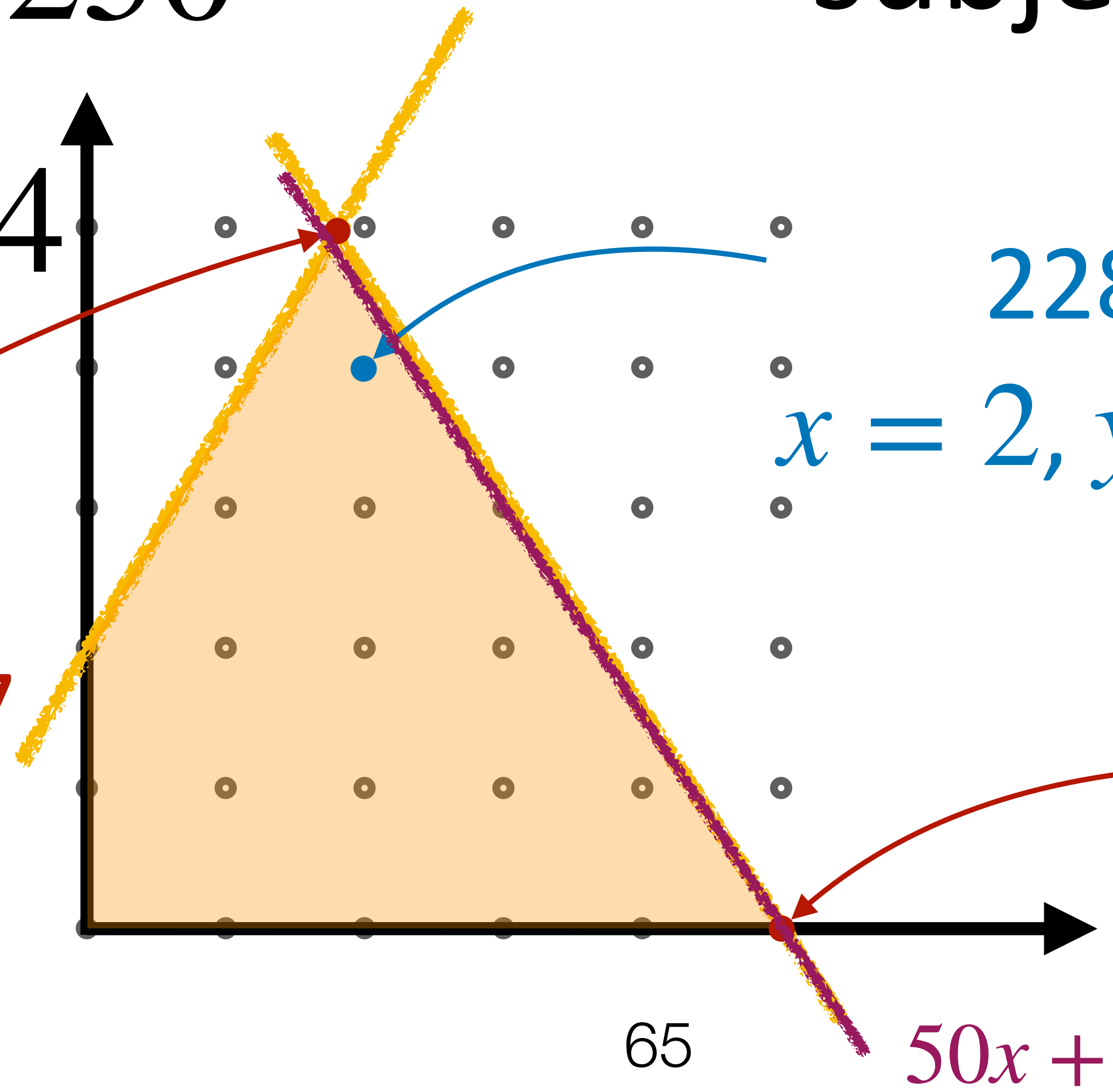
- Integer Linear programming
  - decisions must be integral

maximize $50x + 32y$

subject to $50x + 31y \leq 250$

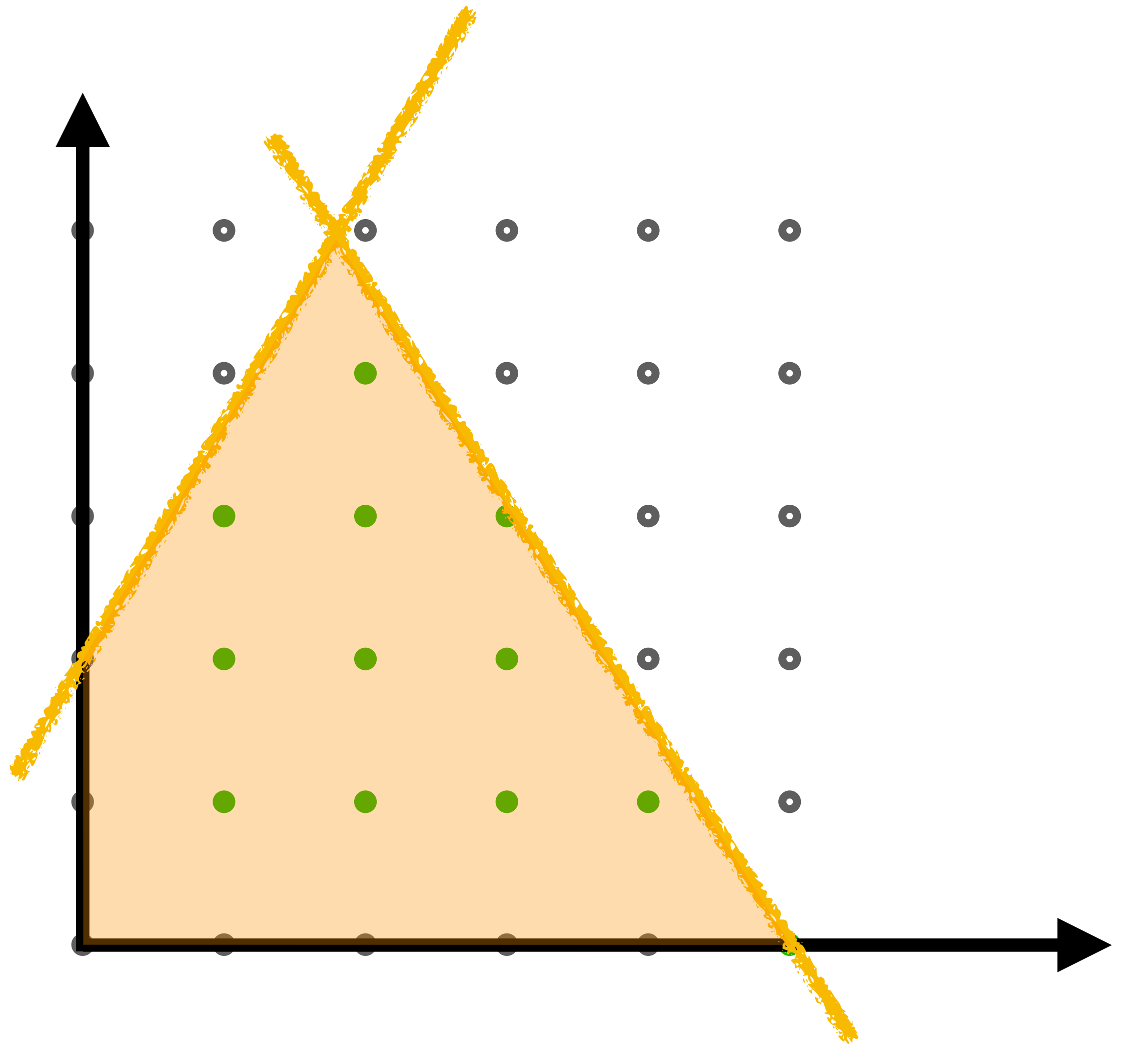
$$3x - 2y \geq -4$$

$$x, y \in \mathbb{N}$$



LP relaxation and upper/lower bound

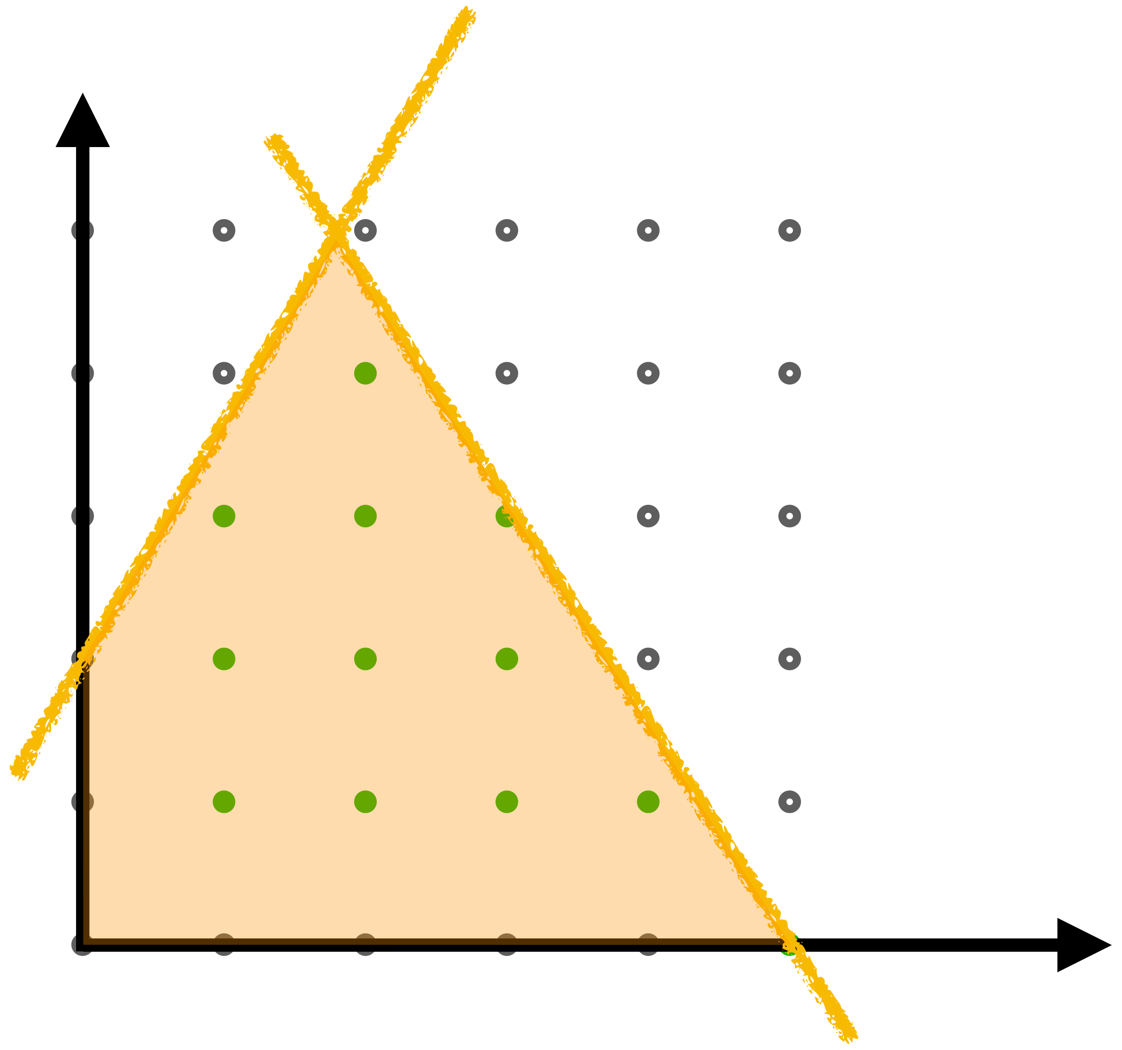
- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value



Any integral solution can be seen as a fractional solution

LP relaxation and upper/lower bound

- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value



Any integral solution can be seen as a fractional solution

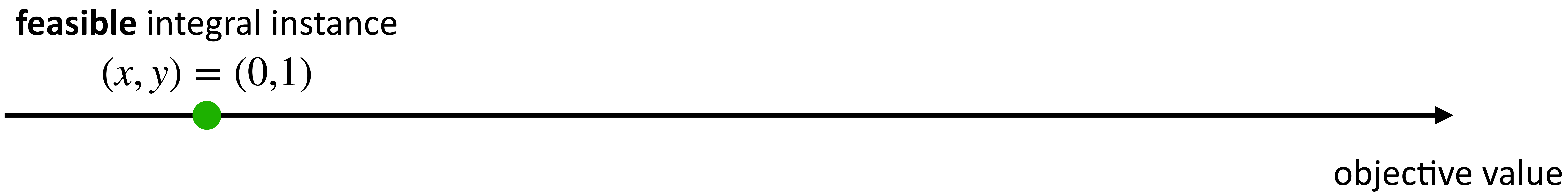
If an optimal fractional solution happens to be integral, it is an optimal integral solution

LP relaxation and upper/lower bound

- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value

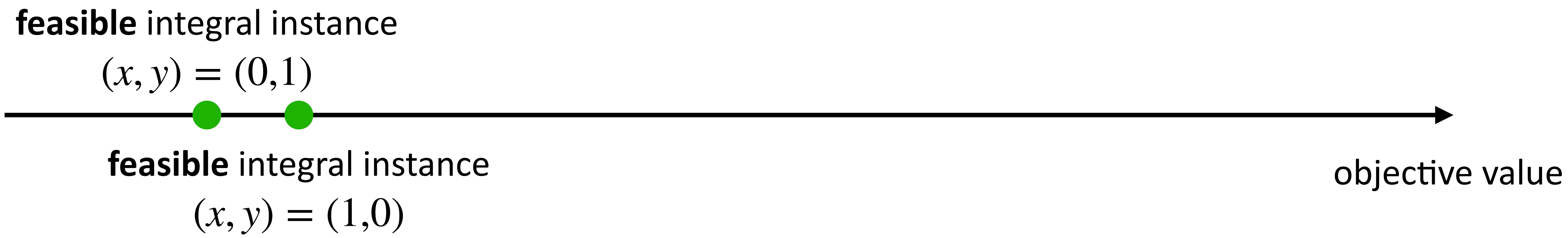
LP relaxation and upper/lower bound

- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value



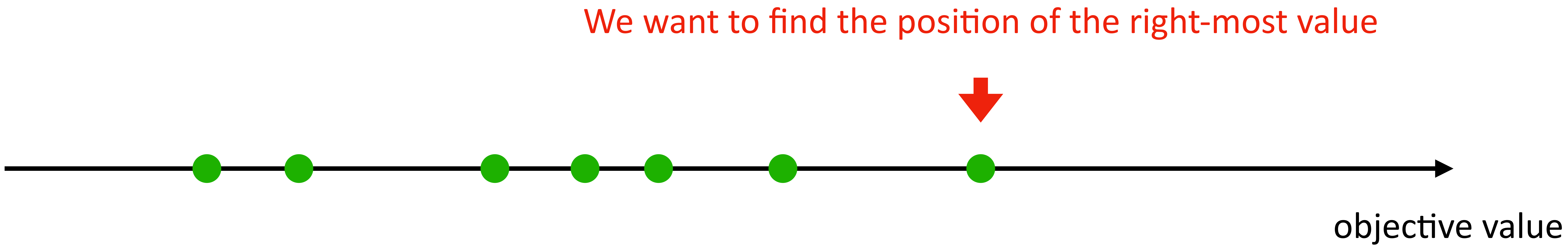
LP relaxation and upper/lower bound

- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value



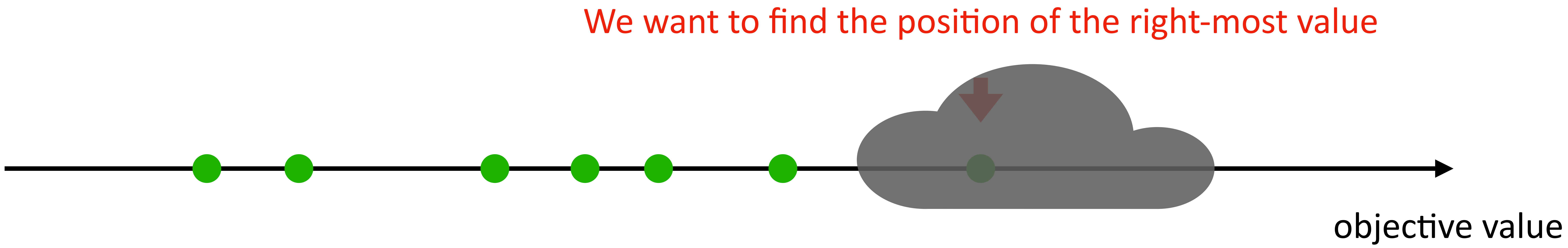
LP relaxation and upper/lower bound

- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value



LP relaxation and upper/lower bound

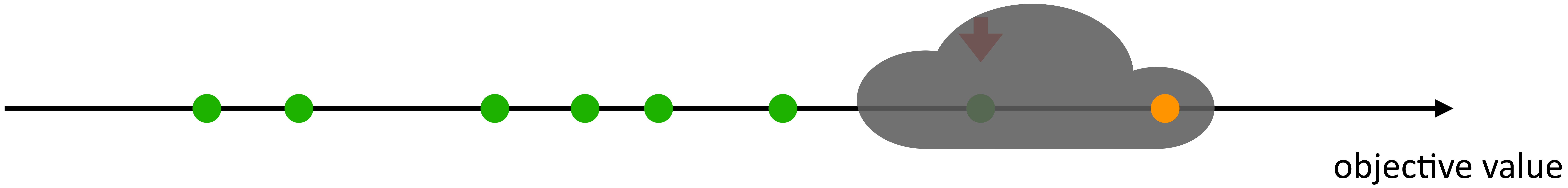
- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value



LP relaxation and upper/lower bound

- For maximization ILP problems, its LP relaxation gives an upper bound of the optimal (integral) value

We want to find the position of the right-most value



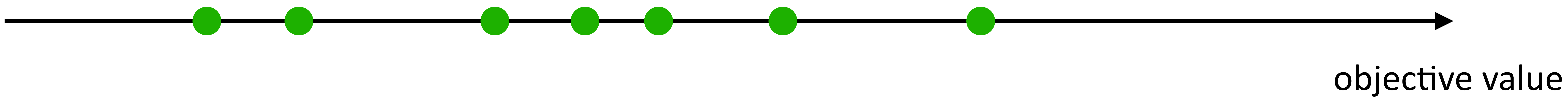
The optimal fractional solution of the LP relaxation
is always an **upper** bound of the optimal integral solution

LP relaxation and upper/lower bound

- For **minimization** ILP problems, its LP relaxation gives an lower bound of the optimal (integral) value

LP relaxation and upper/lower bound

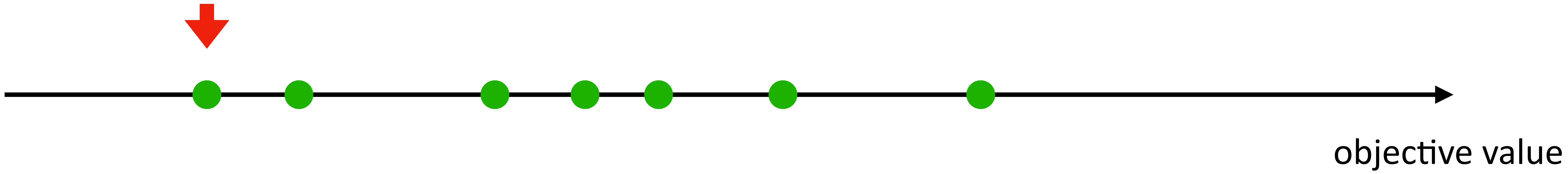
- For **minimization** ILP problems, its LP relaxation gives an lower bound of the optimal (integral) value



LP relaxation and upper/lower bound

- For **minimization** ILP problems, its LP relaxation gives an lower bound of the optimal (integral) value

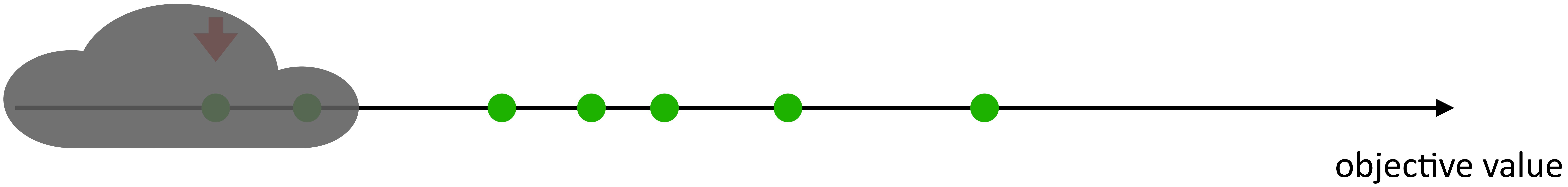
We want to find the position of the left-most value



LP relaxation and upper/lower bound

- For **minimization** ILP problems, its LP relaxation gives an lower bound of the optimal (integral) value

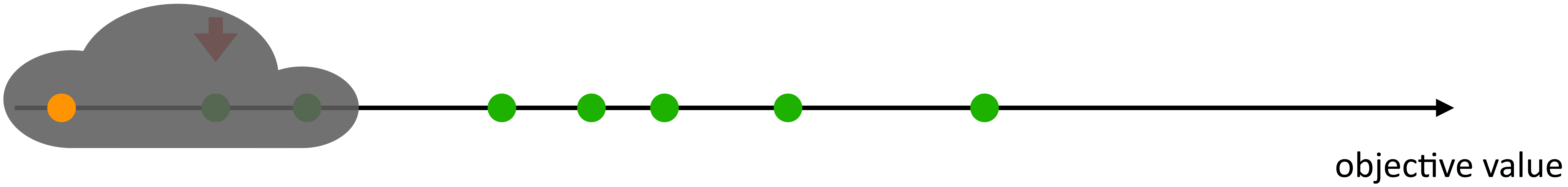
We want to find the position of the left-most value



LP relaxation and upper/lower bound

- For **minimization** ILP problems, its LP relaxation gives an lower bound of the optimal (integral) value

We want to find the position of the left-most value



The optimal fractional solution of the LP relaxation
is always an **lower** bound of the optimal integral solution

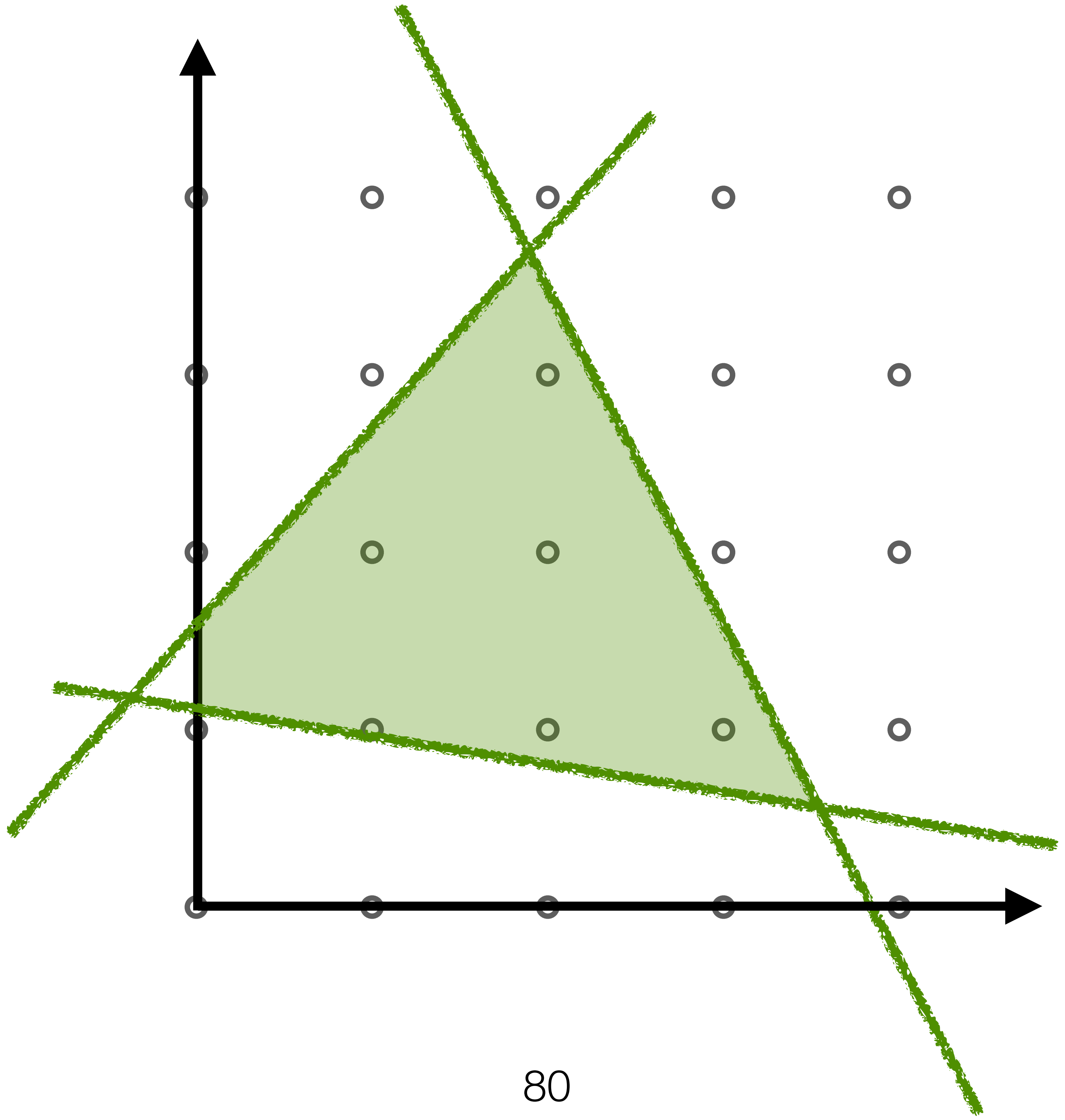
What happened

- It's tricky to find the optimal integral solution, but the optimal fractional solution of the ILP's relaxation provides an upper (lower) bound of the optimal integral solution in the maximization (minimization) problem

Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

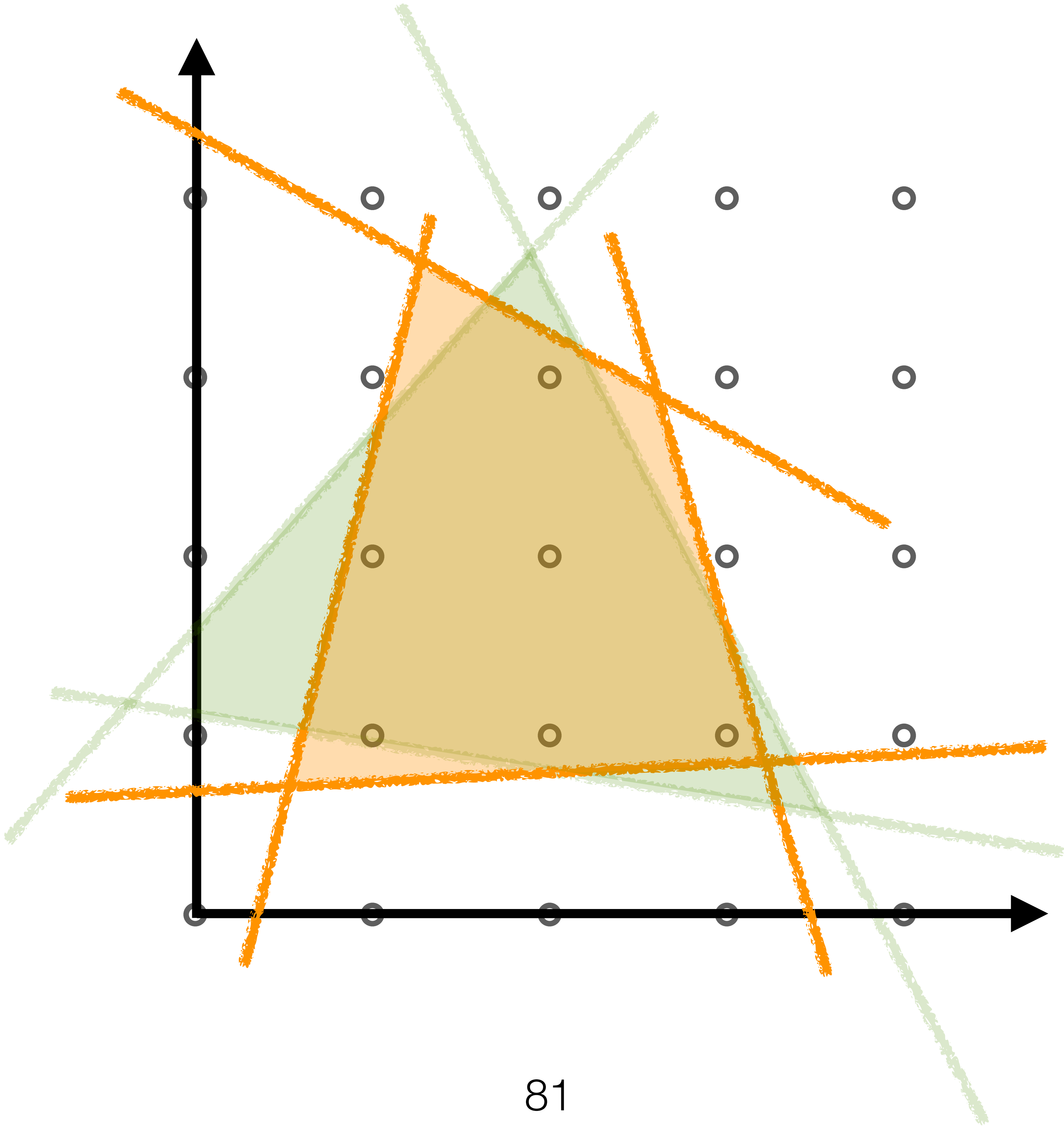


Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

Formulation 2

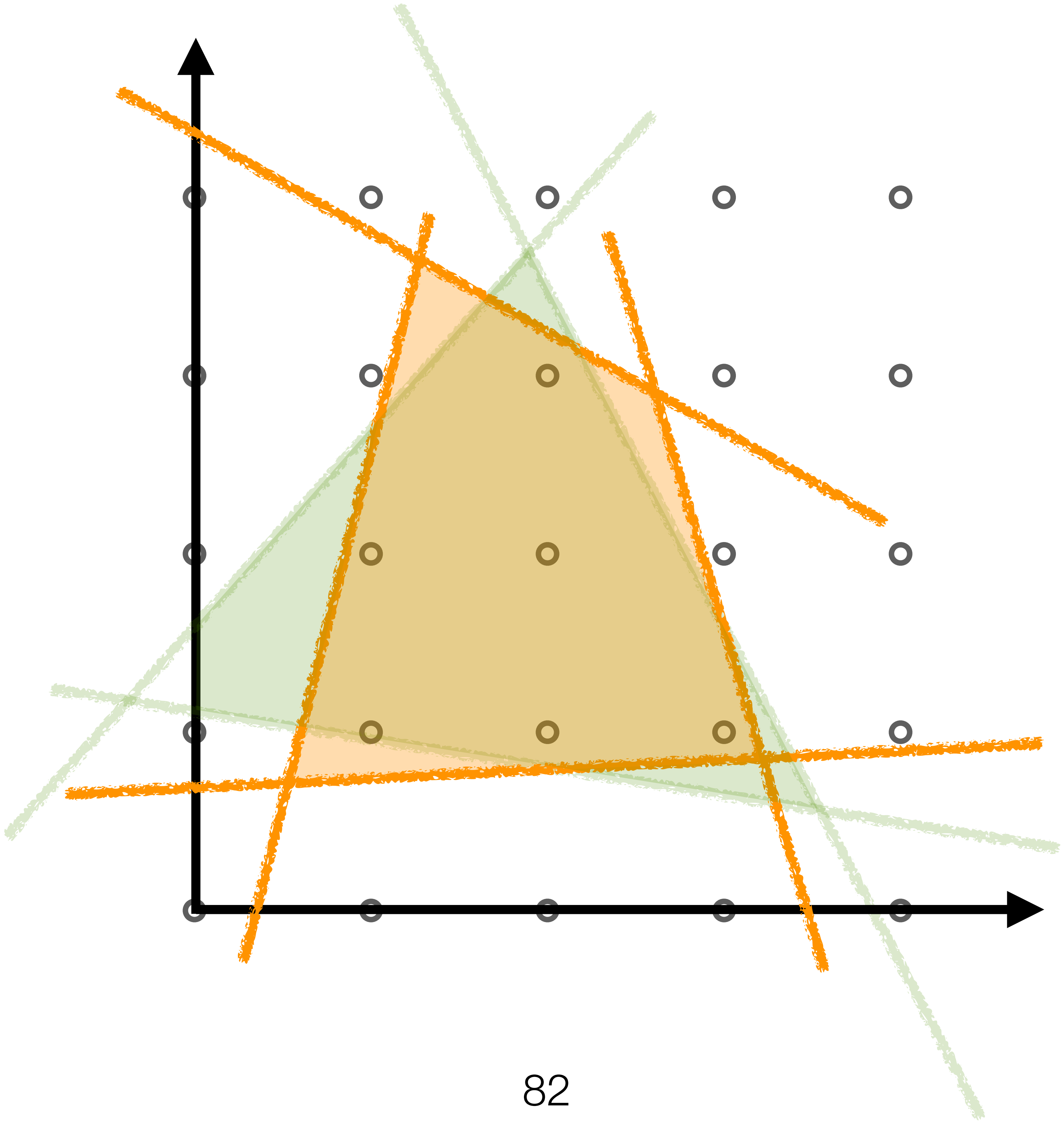


Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

Formulation 2



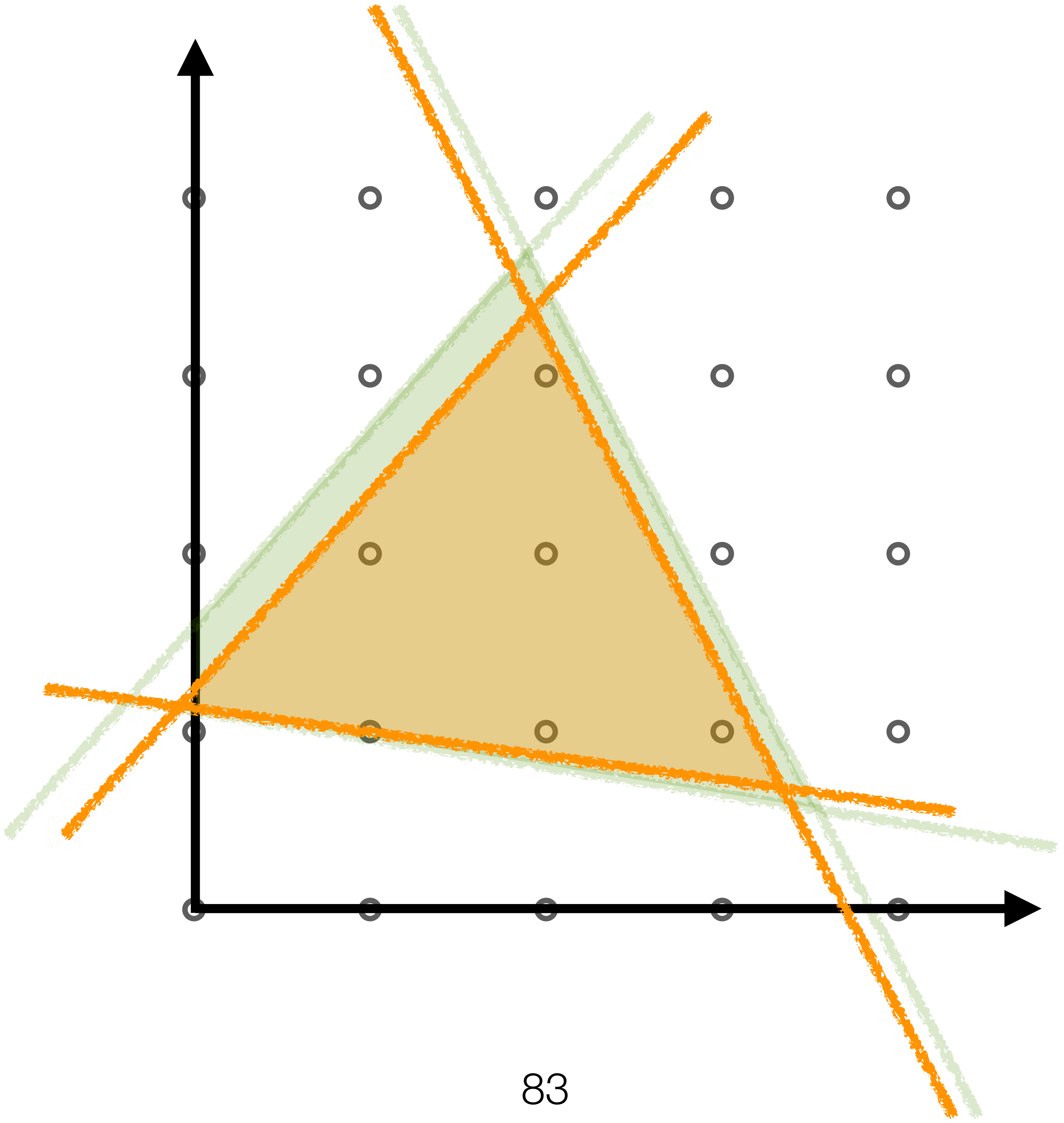
We cannot directly say that Formulation 1 is better or Formulation 2 is better

Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

Formulation 2

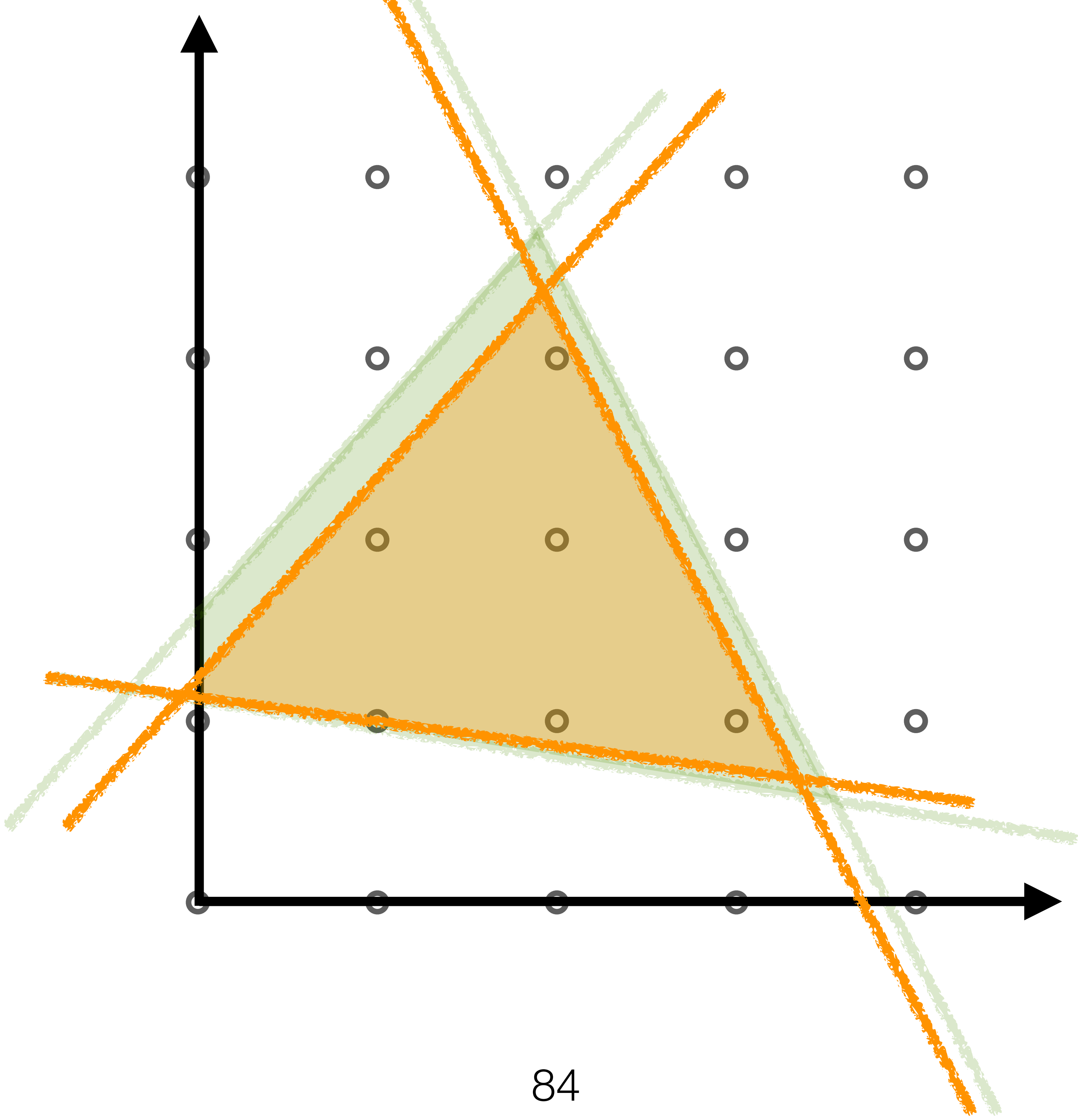


Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

Formulation 2



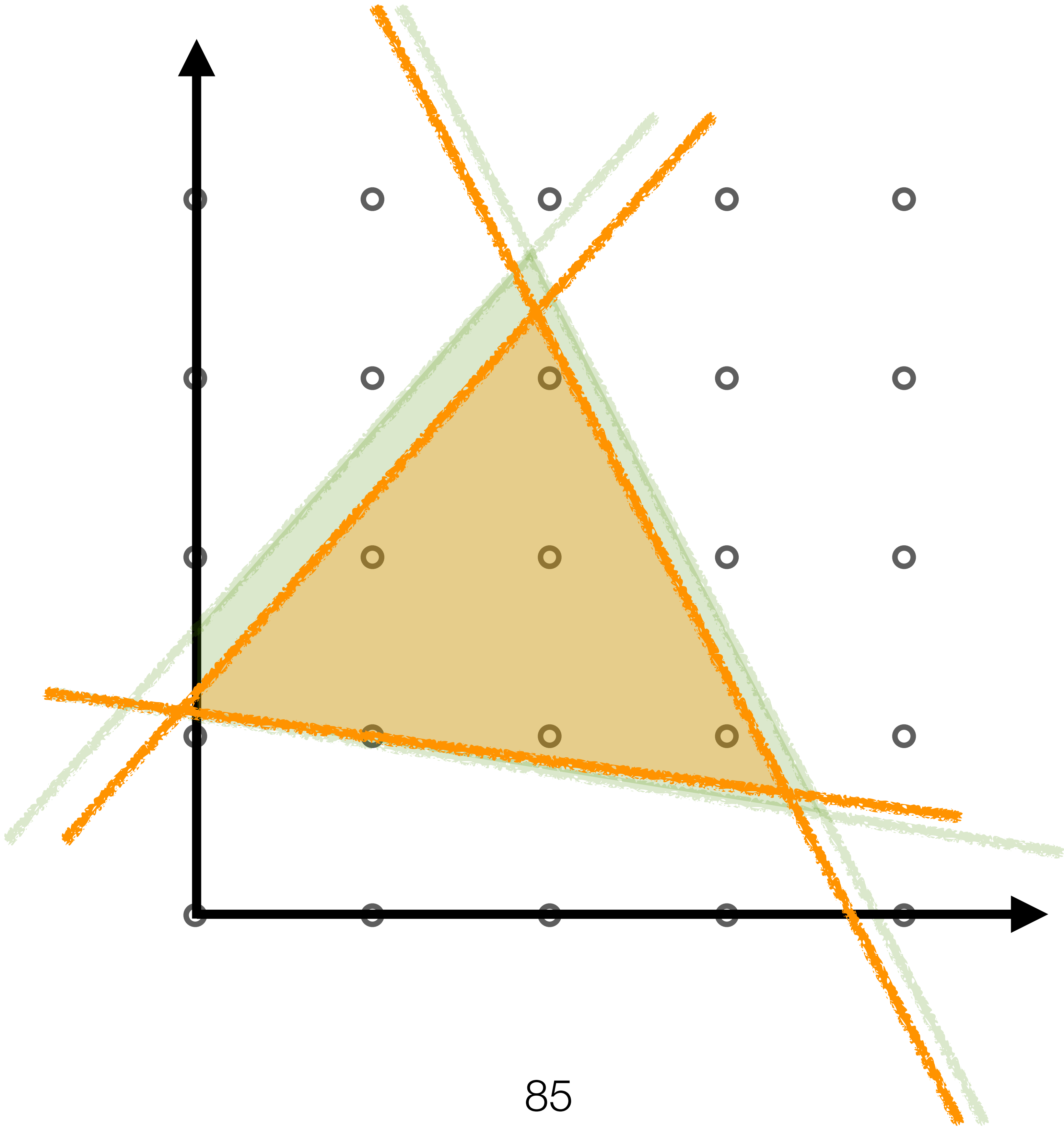
Formulation 2 is better than Formulation 1

Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

Formulation 2



Formulation 2 is better than Formulation 1

(Maximization)
If the feasible region of Form. 2 is fully inside the feasible region of Form. 1, an optimal fractional solution to Form. 2 is always a **feasible** solution to Form. 1
$\Rightarrow OPT_{LP1} \geq OPT_{LP2} \geq OPT_{ILP}$

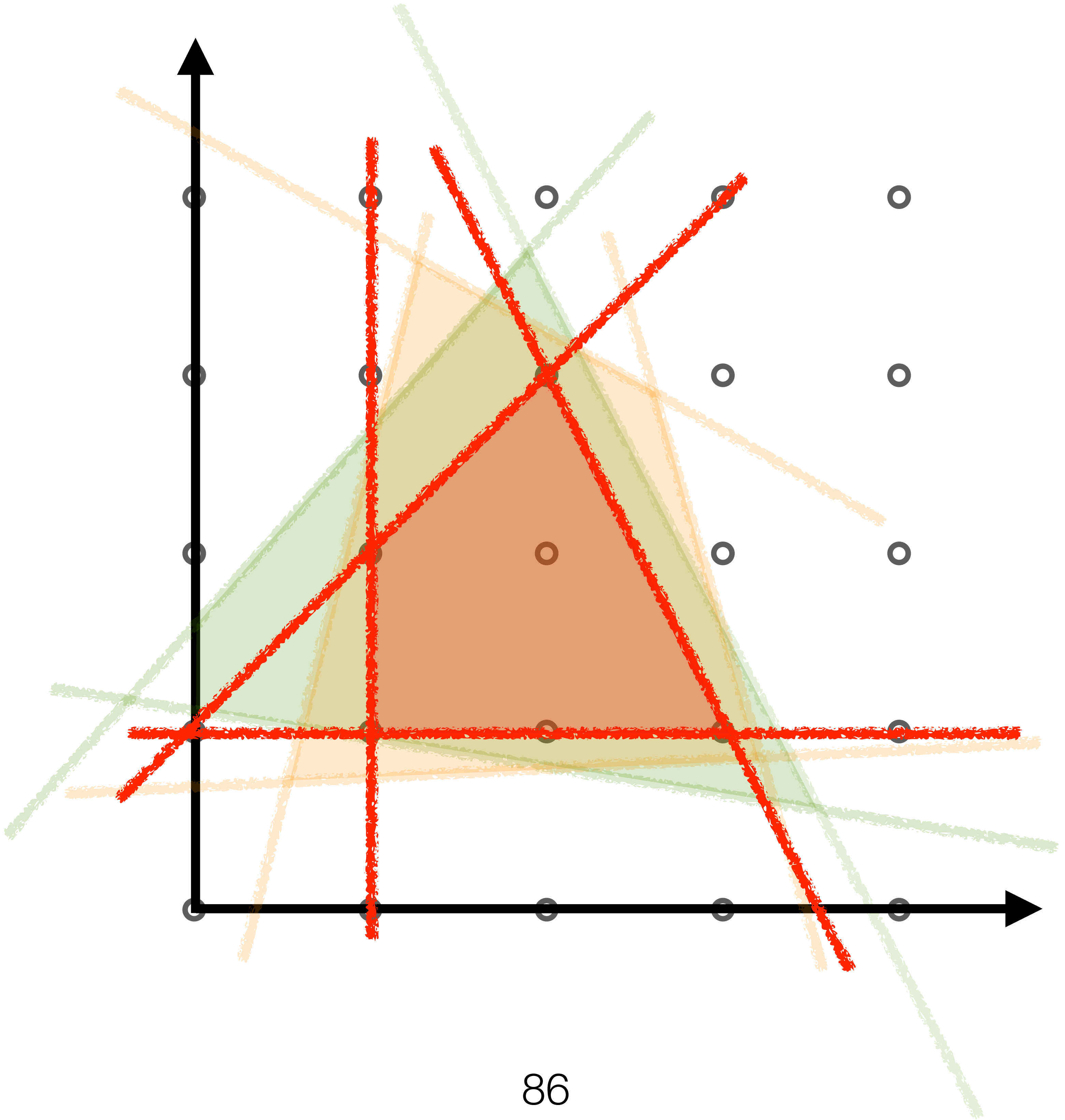
Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

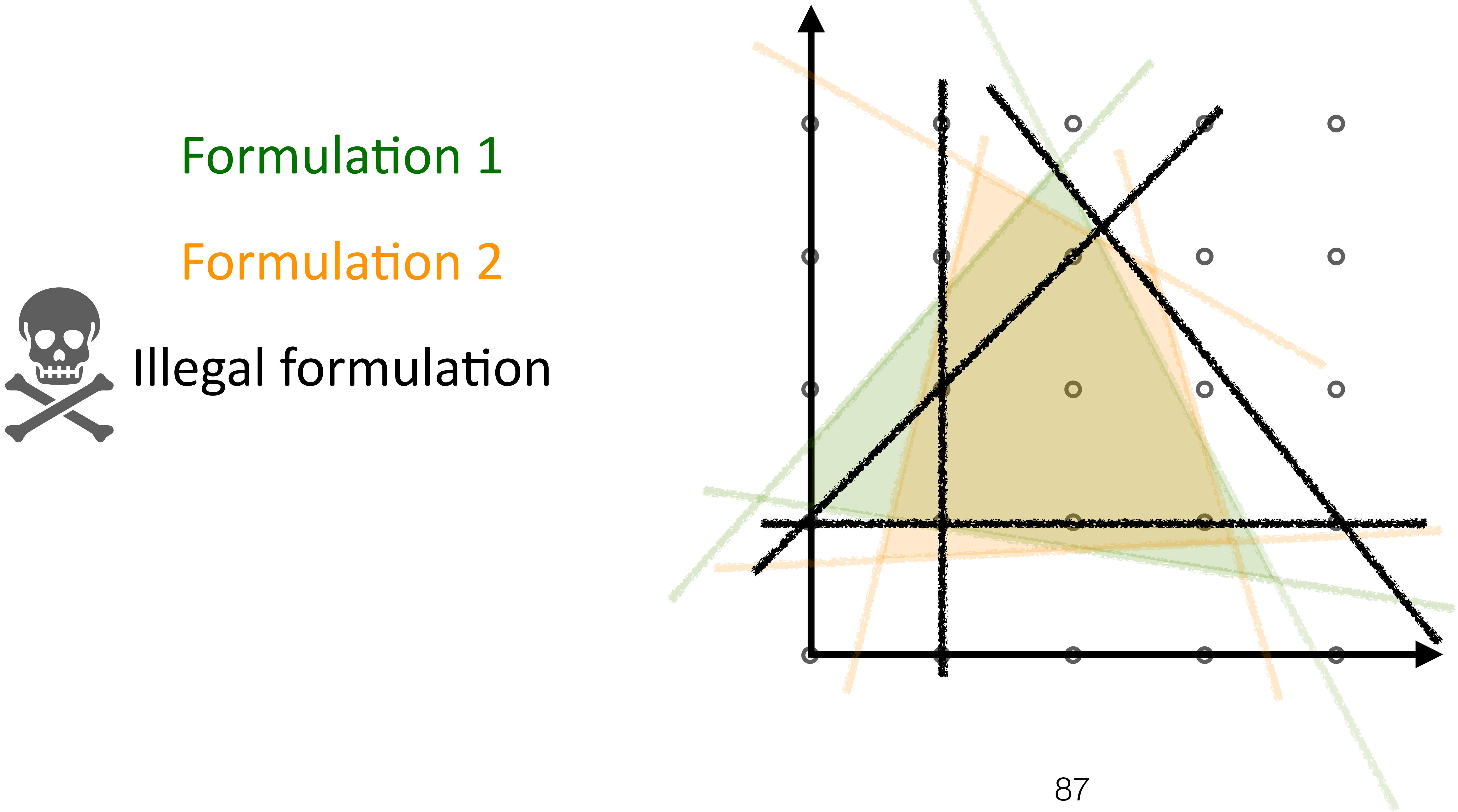
Formulation 2

Ideal formulation



Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
    - How can we choose between them?



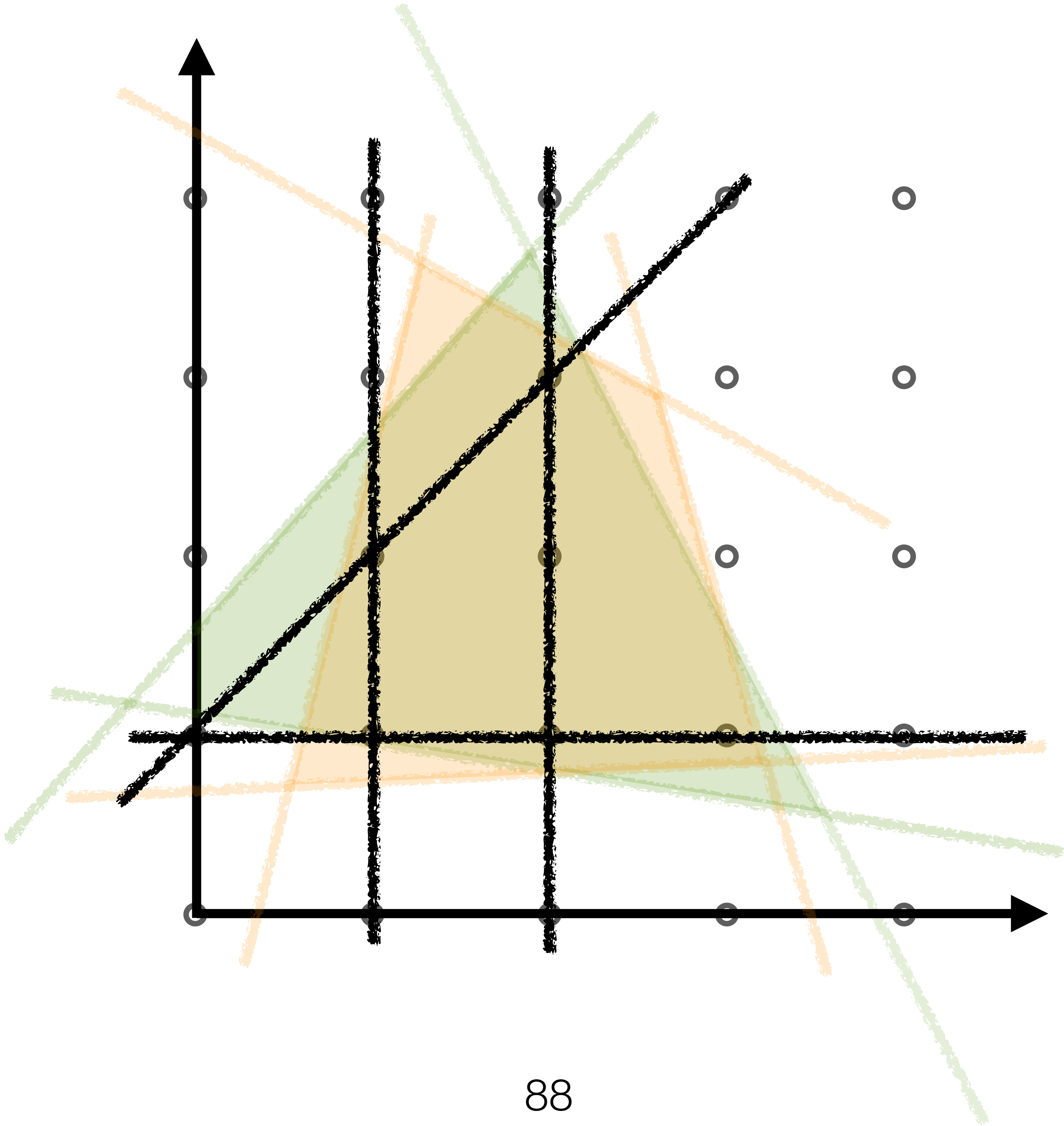
Different formulations of ILP

- Geometrically, we can see that there must be an infinite number of formulations
  - How can we choose between them?

Formulation 1

Formulation 2

☠ Illegal formulation



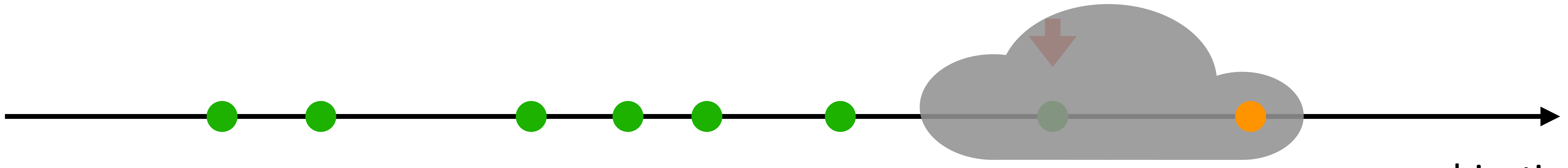
Modeling choice

- There are alternative formulations, and some might be “better” than others
  - That is, it more accurately/efficiently capture the optimal (integral) solution

Modeling choice

- There are alternative formulations, and some might be “better” than others
    - That is, it more accurately/efficiently capture the optimal (integral) solution

We want to find the position of the right-most value

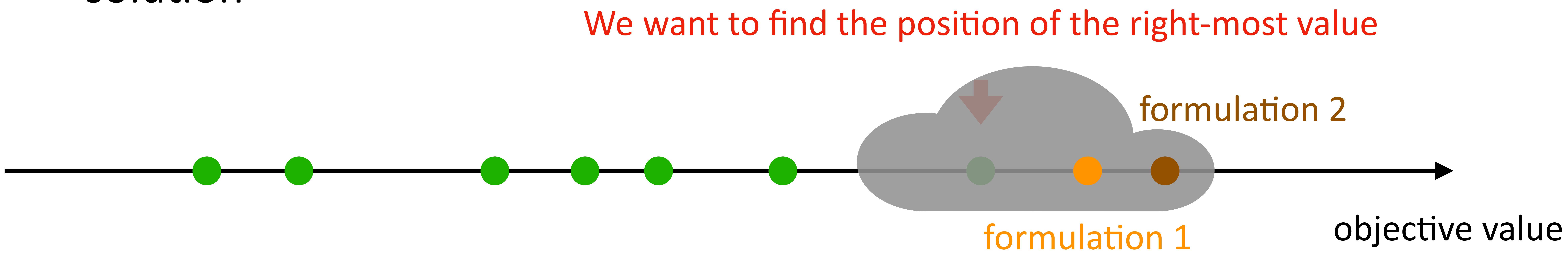


objective value

The optimal fractional solution of the LP relaxation
is always an **upper** bound of the optimal integral solution

Modeling choice

- There are alternative formulations, and some might be “better” than others
  - That is, it more accurately/efficiently capture the optimal (integral) solution

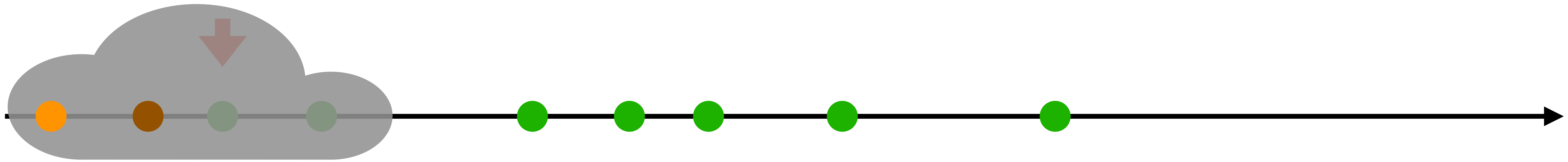


The optimal fractional solution of the LP relaxation
is always an **upper** bound of the optimal integral solution

Modeling choice

- There are alternative formulations, and some might be "better" than others
  - That is, it more accurately/efficiently capture the optimal (integral) solution

We want to find the position of the left-most value



formulation 1 formulation 2

objective value

The optimal fractional solution of the LP relaxation
is always an **upper** bound of the optimal integral solution

What happened

- Different formulation (via different sets of constraints) might provide different optimal fractional solutions
- The different formulations shouldn’t exclude any feasible integral solution or include any infeasible integral solution

Outline

- More modeling optimization problems to (integer) programming problems
  - **Set cover**
  - **Shortest paths**
  - **Traveling Salesperson Problem**

- LP relaxation and upper/lower bound

- Solving ILP: Branch and bound method

Branch-and-Bound

B&B

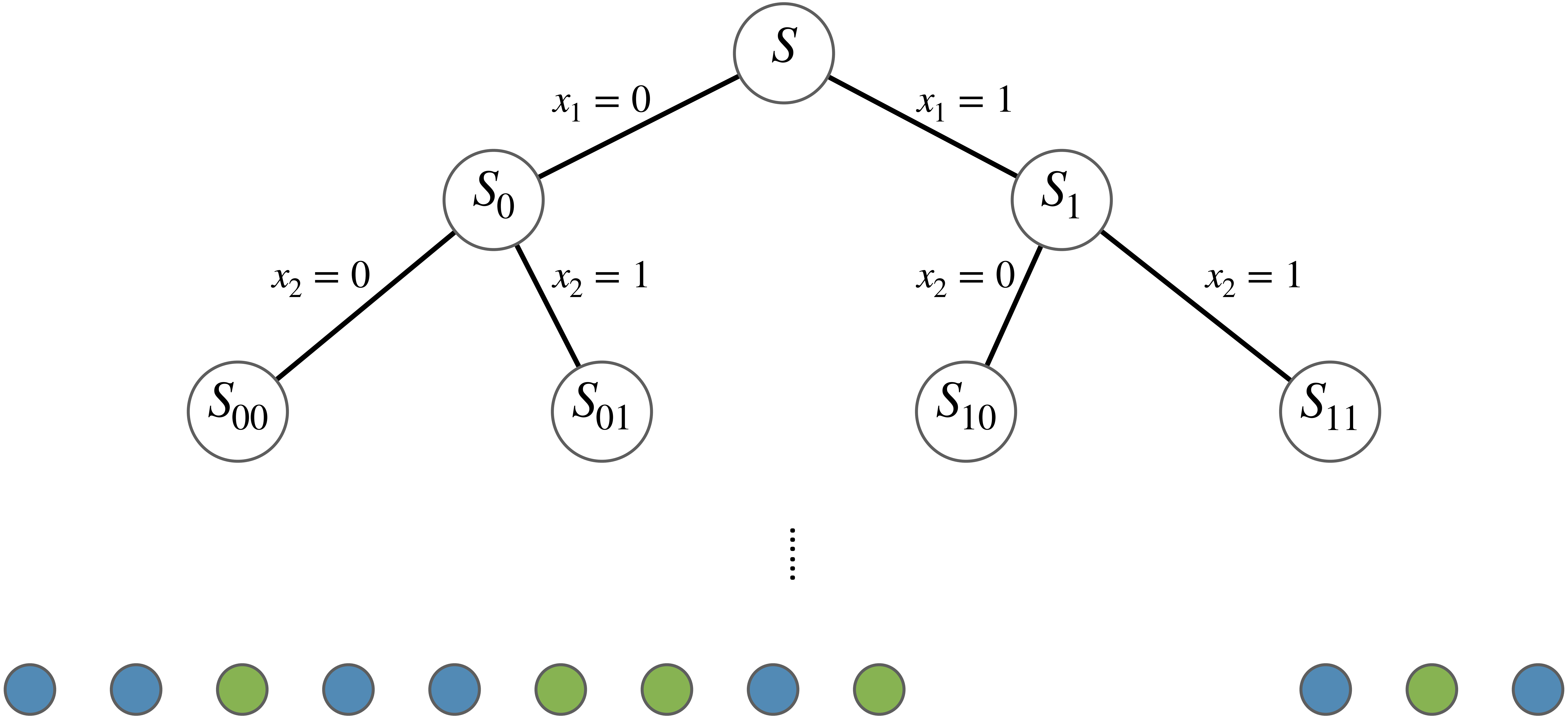
HOTELS

Branch-and-Bound

- Solve integer programming problems
  - Listing every feasible solution $(x_1, x_2, \cdots, x_n) = (0,1,\cdots,0)$ solves the problem (not efficiently)

- Idea: Use divide and conquer via an enumeration tree
  - Divide the solution set into subsets
  - Find the upper bound and lower bound of the optimal solution within each subset
  - "Cut" the branch if the bounds provide enough information

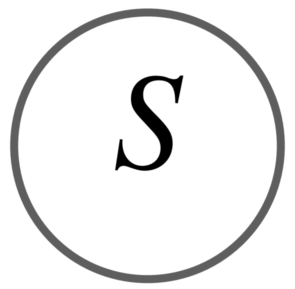
Branch-and-Bound

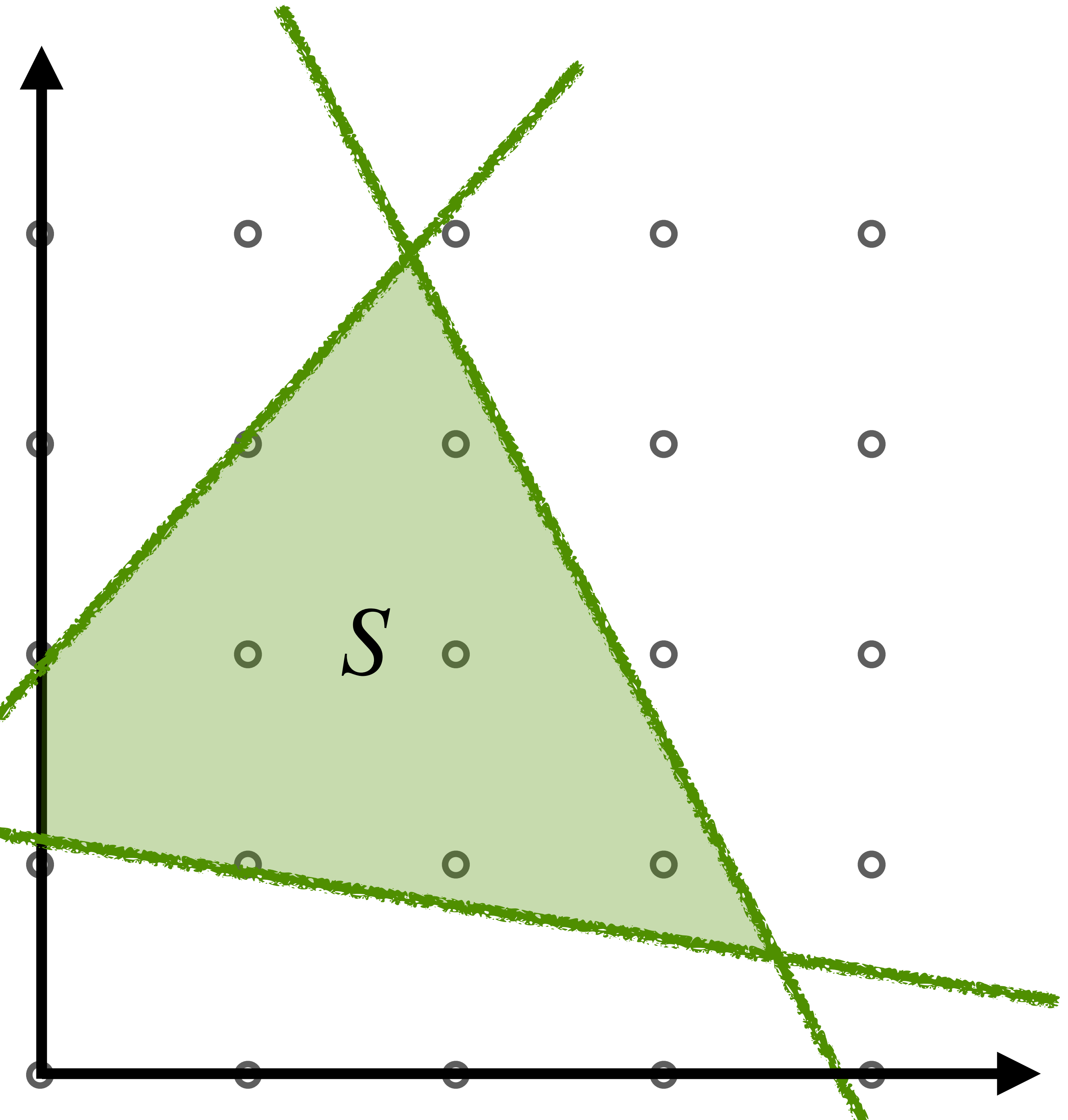


all possible solutions (feasible or infeasible)

Branch-and-Bound

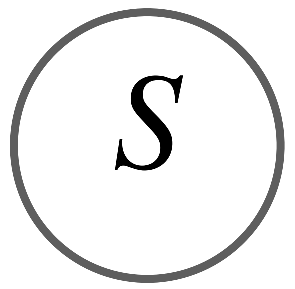
Maximization





Branch-and-Bound

Maximization



Branch-and-Bound

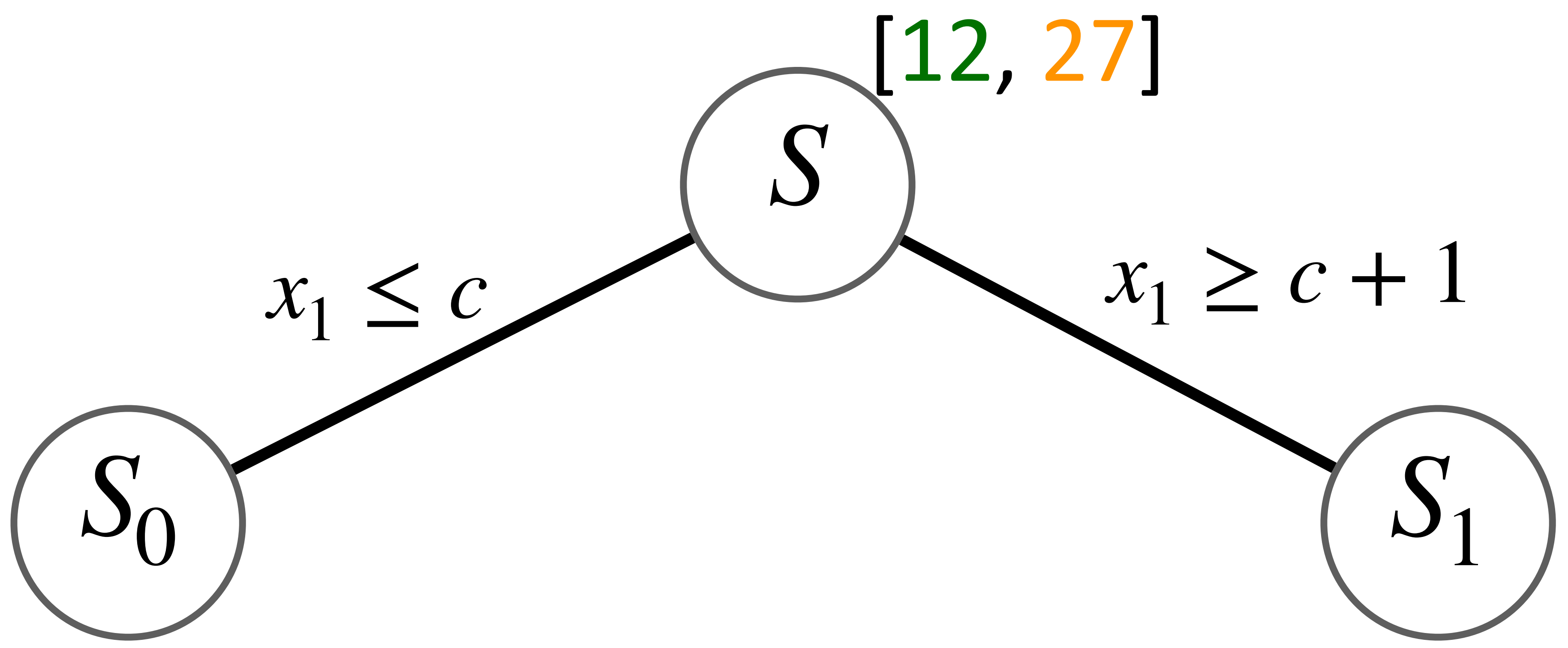
Maximization

Branch-and-Bound

Maximization

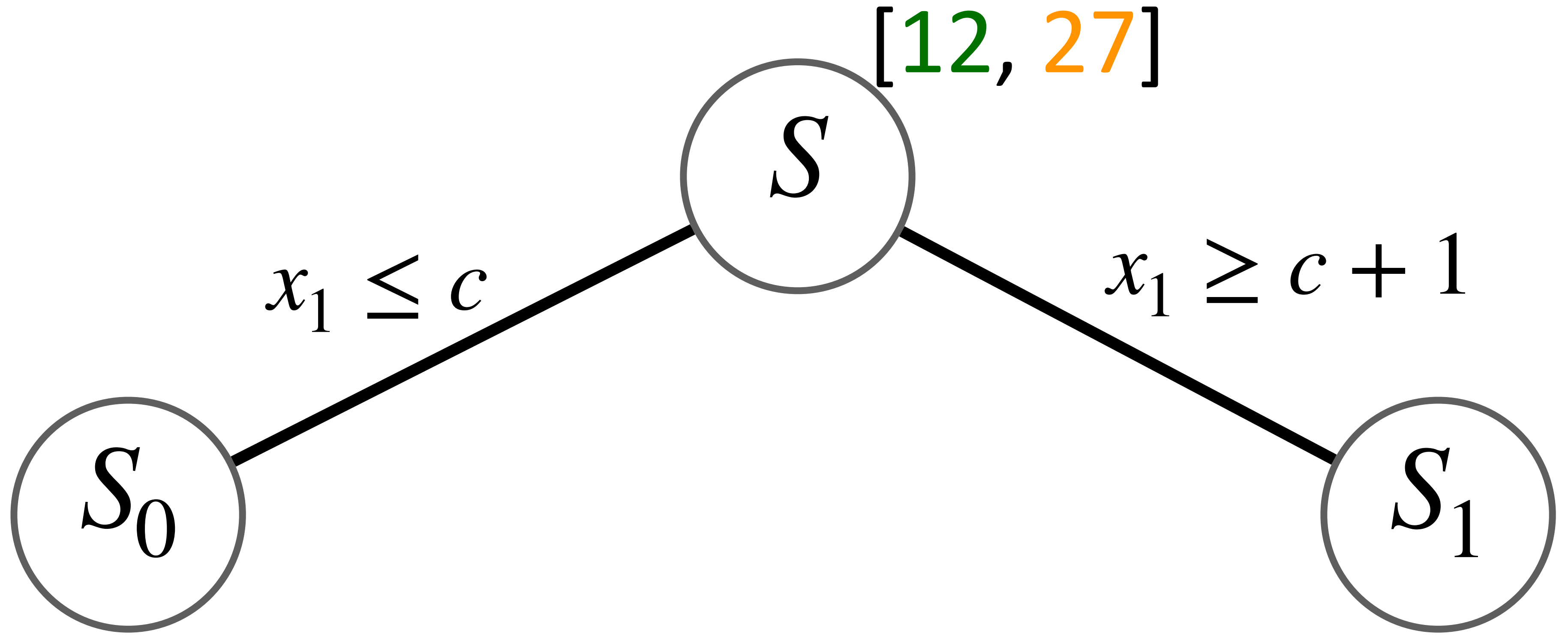
Branch-and-Bound

Maximization



Branch-and-Bound

Maximization

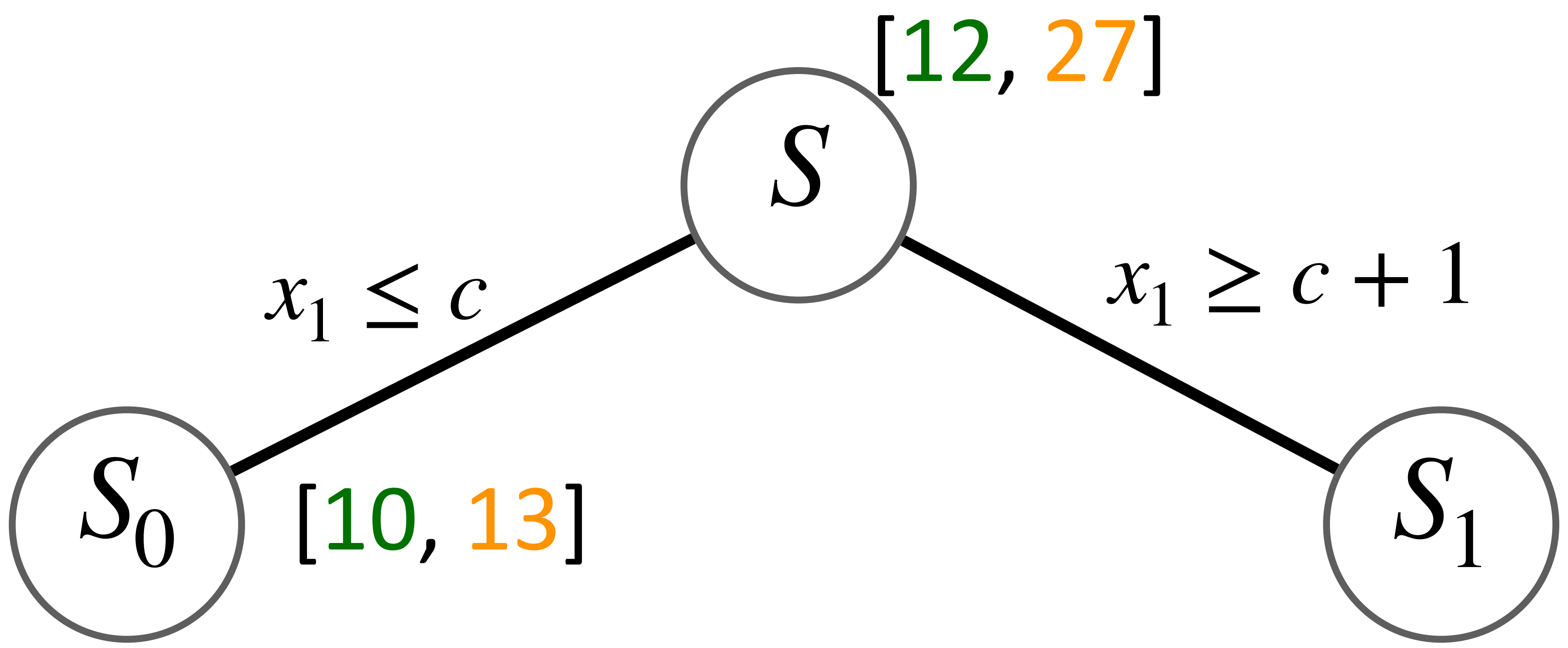


Branch-and-Bound

Maximization

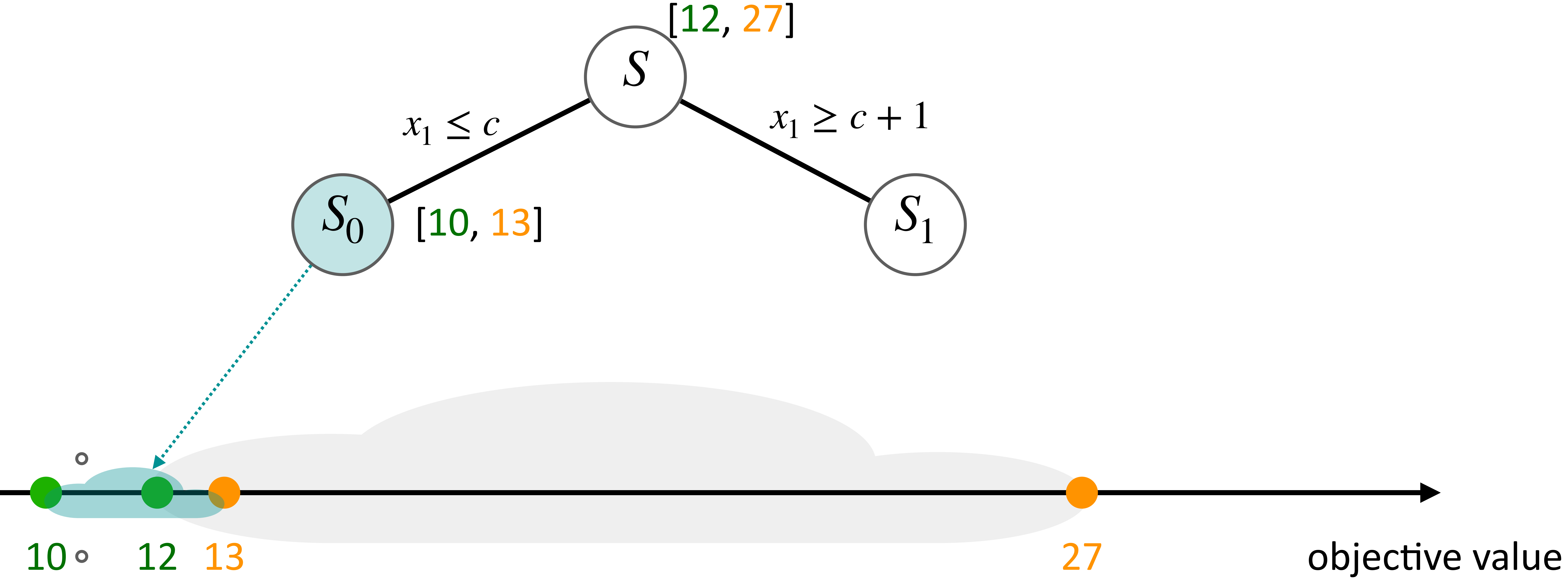
Branch-and-Bound

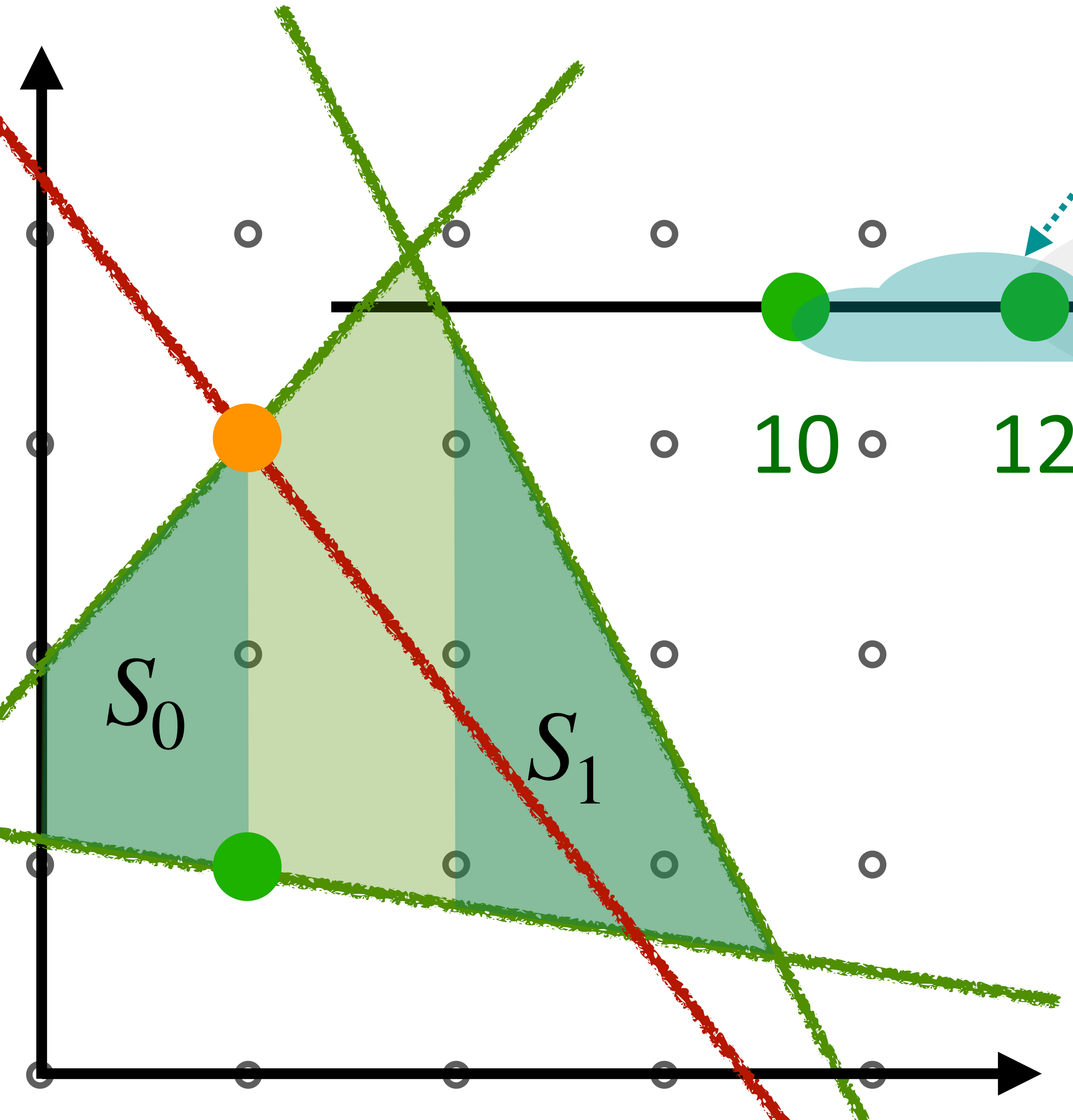
Maximization



Branch-and-Bound

Maximization





Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Branch-and-Bound

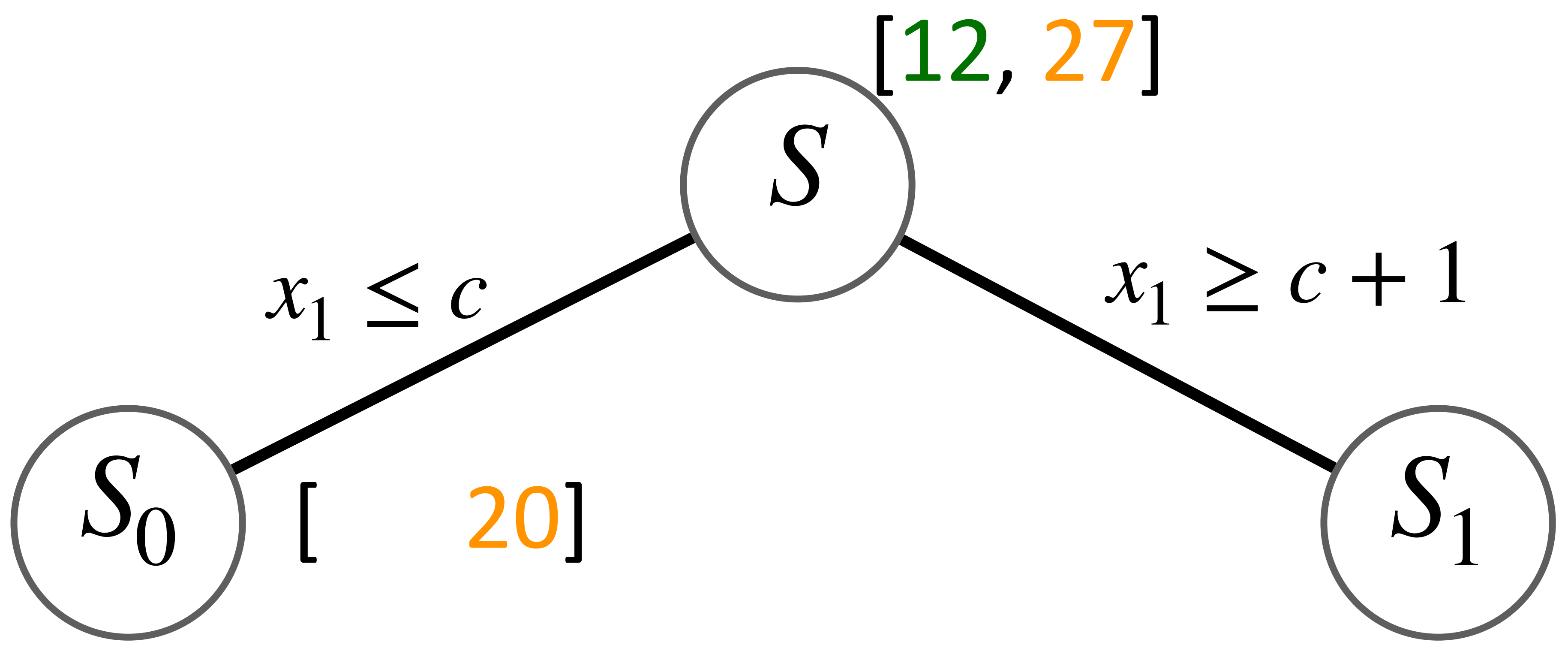
Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Maximization



Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

Branch-and-Bound

Maximization

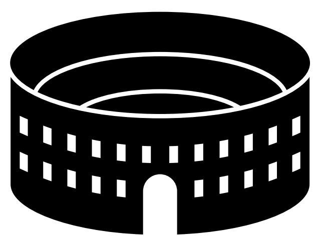
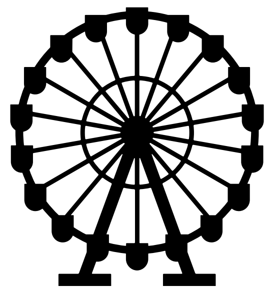
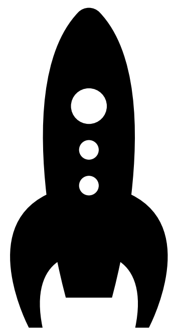
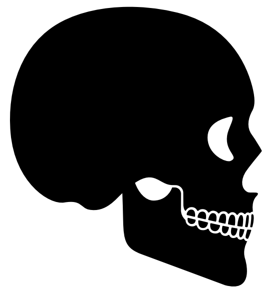
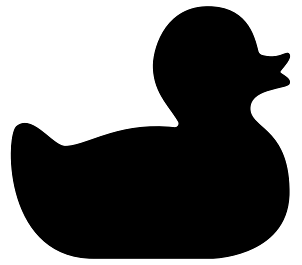
Branch-and-Bound

Maximization

Branch-and-Bound

- Branch-and-bound method solves ILPs by gradually narrowing down the range of optimal solutions
  - Branching: divide the solution space via choices of specific variables
  - Bound: improve the range of the optimal value via pruning a branch or merging the bounds from different branches

Use Branch-and-Bound to solve Knapsack

project	1	2	3	j	n	
						budget
outlay	a_1	a_2	a_3	a_4	a_5	b
expected return	c_1	c_2	c_3	c_4	c_5	

- There is a budget b available for investment in projects during the coming year, and n projects are under consideration, where a_j is the outlay for project j, and c_j is its expected return. The goal is to choose a set of projects so that the budget is not exceeded and the expected return is maximized

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay	4	8	3	6	5

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay	4	8	3	6	5
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6	5
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 0

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

- Optimal fractional solution can be found by **greedily selecting the item with the highest outlay/return value** without exceeding the budget

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 1	5
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 6

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

- Optimal fractional solution can be found by **greedily selecting the item with the highest outlay/return value** without exceeding the budget

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 1	5 1
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 11

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

- Optimal fractional solution can be found by **greedily selecting the item with the highest outlay/return value** without exceeding the budget

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 1	6 1	5 1
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 14

$$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$$

- Optimal fractional solution can be found by **greedily selecting the item with the highest outlay/return value** without exceeding the budget

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8	3 1	6 1	5 1
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 15

$$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$$

- Optimal fractional solution can be found by **greedily selecting the item with the highest outlay/return value** without exceeding the budget

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8 0	3 1	6 1	5 1
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 15

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

- Optimal fractional solution can be found by **greedily selecting the item with the highest outlay/return value** without exceeding the budget

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8 0	3 1	6 1	5 1
outlay/return	2	1.5	2.333	2.5	2.4

maximize $8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$

subject to $4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15$ total outlay: 15

$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$

- Optimal fractional solution can be found by greedily selecting the item with the highest outlay/return value without exceeding the budget
  - OPT$_f = [8,12,7,15,12] \cdot [\frac{1}{4},0,1,1,1]^T = 36$

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 1 1	6 1 1	5 1 1
outlay/return	2	1.5	2.333	2.5	2.4

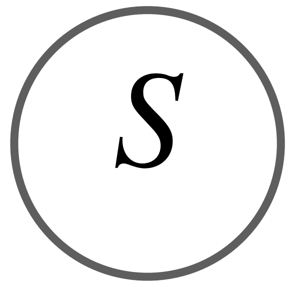
$$\text{maximize } 8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5$$
$$\text{subject to } 4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \le 15$$
$$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$$

- Optimal fractional solution can be found by greedily selecting the item with the highest outlay/return value without exceeding the budget
  - $\text{OPT}_f = [8,12,7,15,12] \cdot [\frac{1}{4},0,1,1,1]^T = 36$, and there is a feasible integral solution $[8,12,7,15,12] \cdot [0,0,1,1,1]^T = 34$

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 1 1	6 1 1	5 1 1
outlay/return	2	1.5	2.333	2.5	2.4

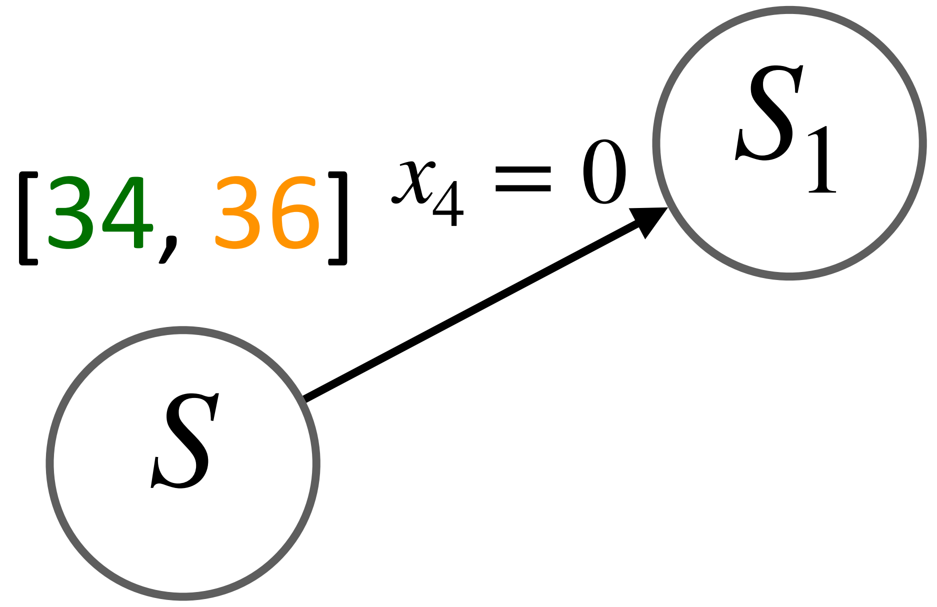
[34, 36]



- OPT$_f$ $= [8,12,7,15,12]\cdot[\frac{1}{4},0,1,1,1]^T = 36$, and there is a feasible integral solution $[8,12,7,15,12]\cdot[0,0,1,1,1]^T = 34$

Use Branch-and-Bound to solve Knapsack

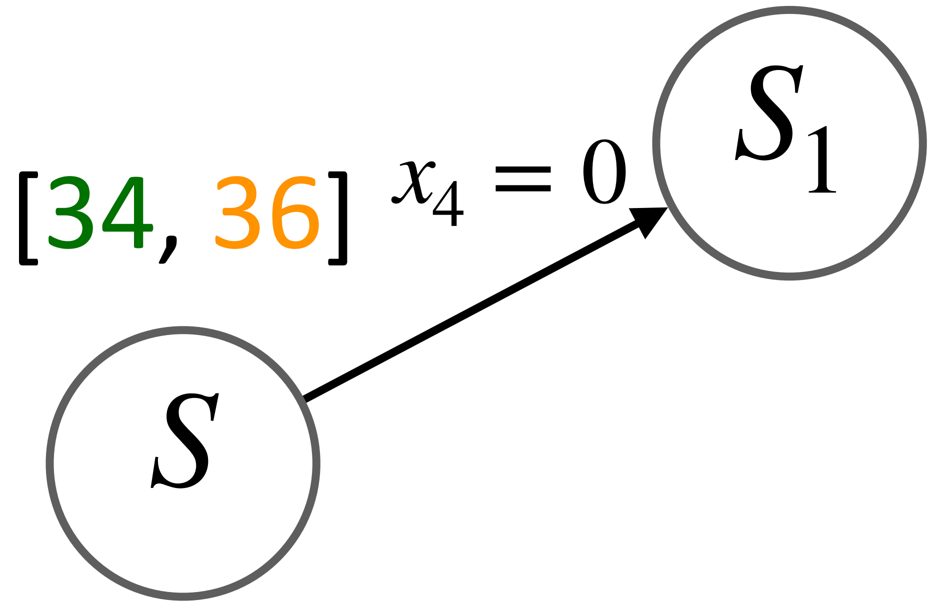
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **0** **0**	5
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 0

Use Branch-and-Bound to solve Knapsack

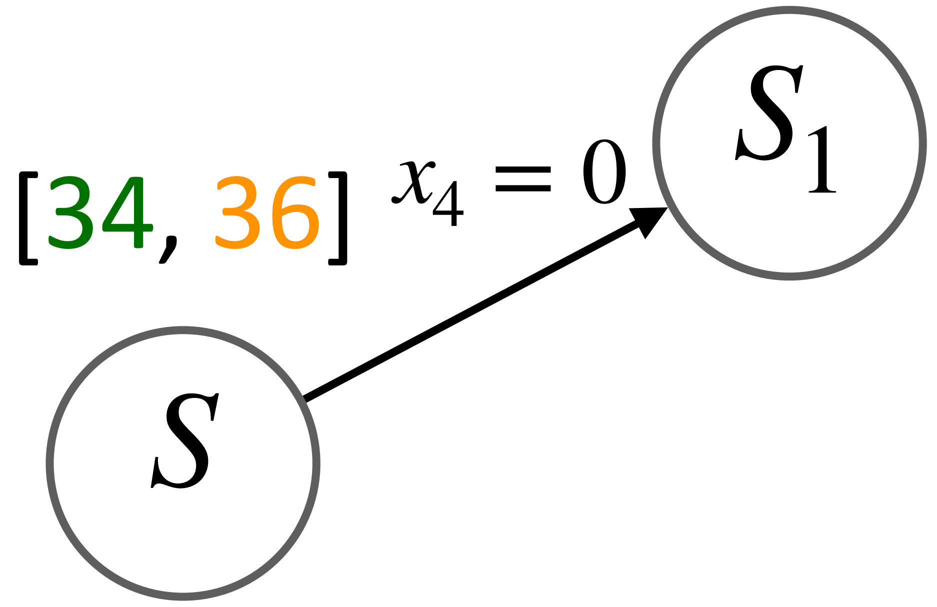
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **0** **0**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 5

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 1	6 **0** **0**	5 1
outlay/return	2	1.5	2.333	2.5	2.4

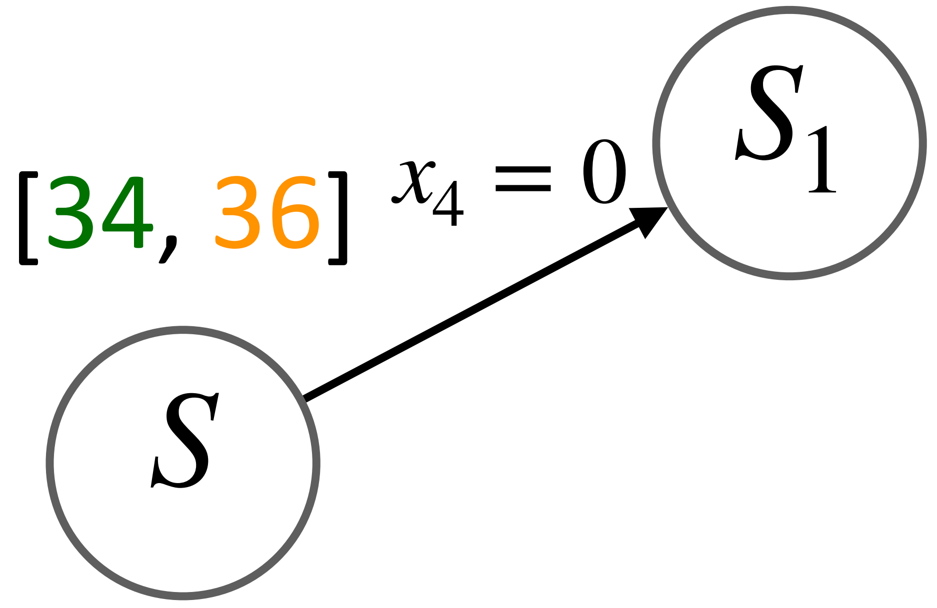


total outlay: 8

Use Branch-and-Bound to solve Knapsack

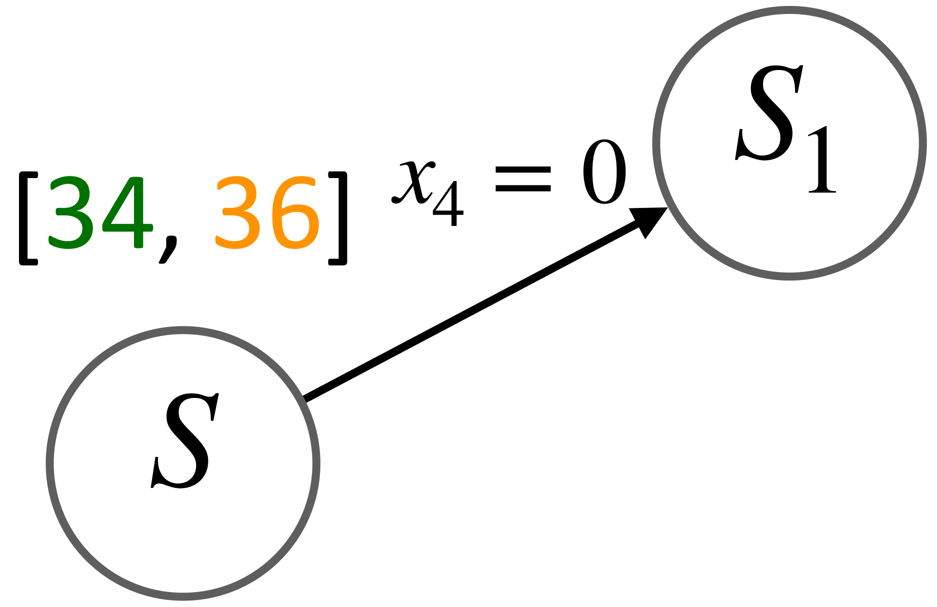
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8	3 1	6 **0** **0**	5 1
outlay/return	2	1.5	2.333	2.5	2.4

total outlay: 12



Use Branch-and-Bound to solve Knapsack

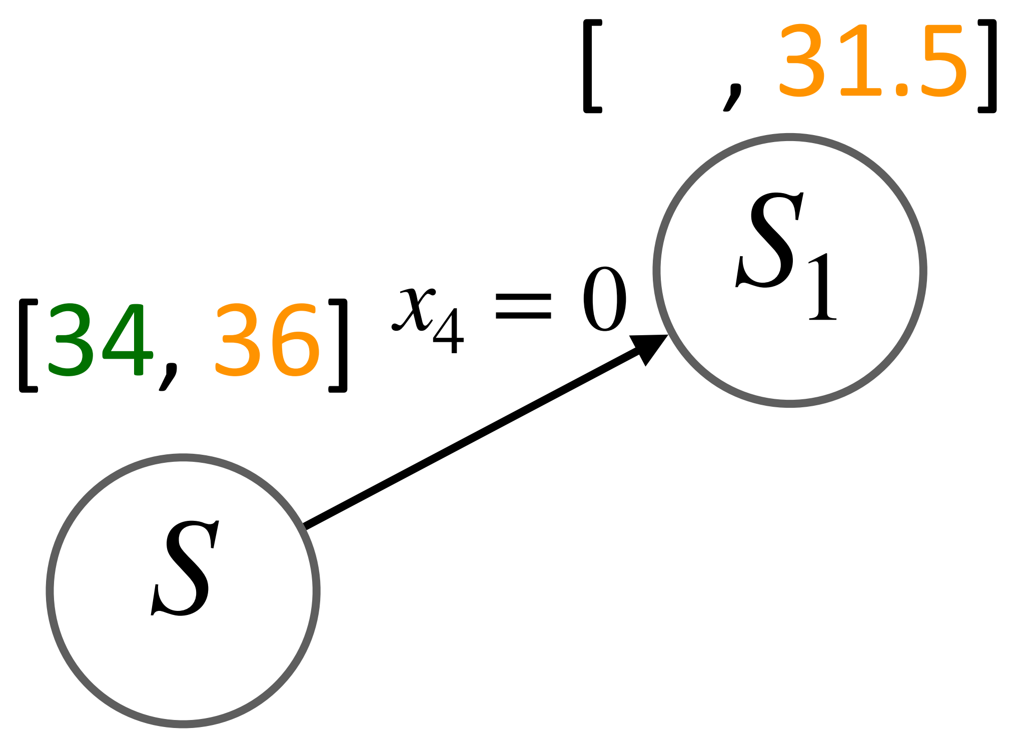
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8 $\frac{3}{8}$	3 1	6 **0** **0**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

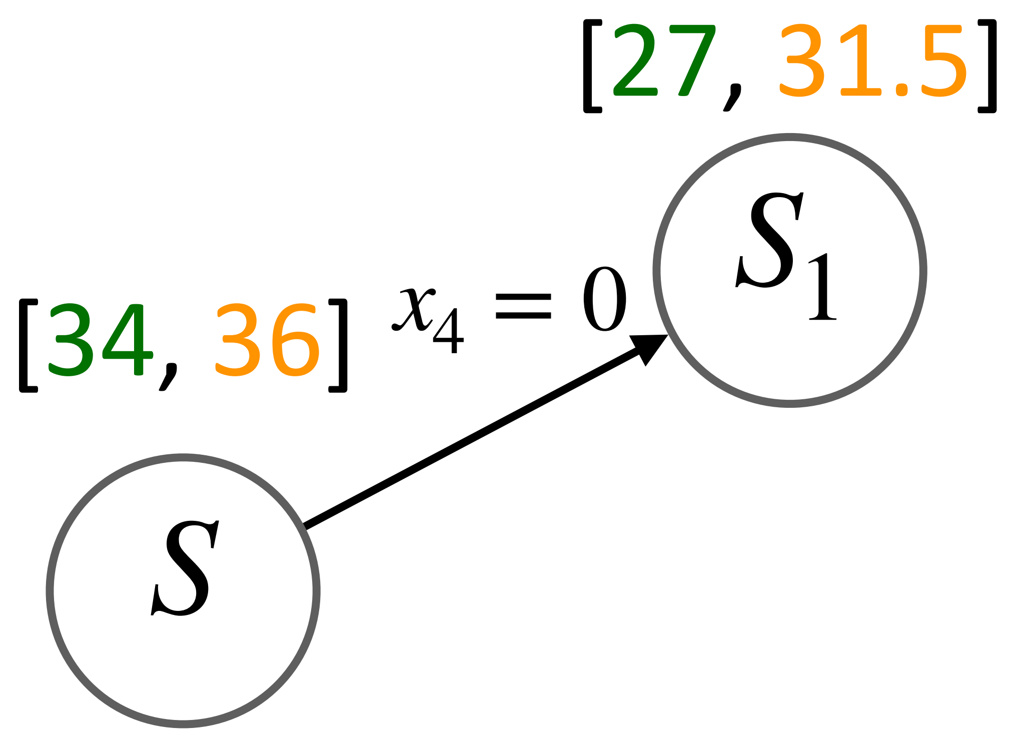
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8 $\frac{3}{8}$	3 1	6 **0** **0**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

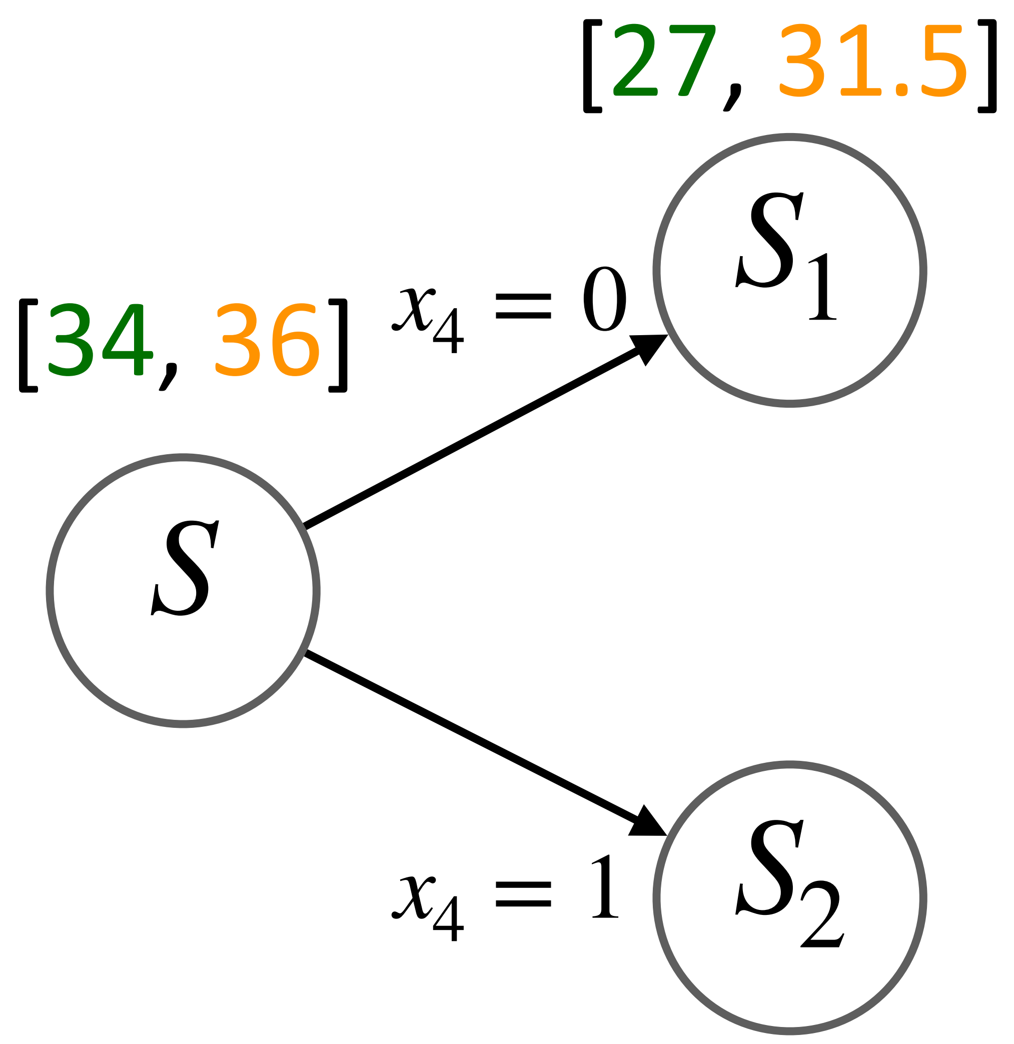
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1 1	8 $\frac{3}{8}$ 0	3 1 1	6 **0** **0**	5 1 1
outlay/return	2	1.5	2.333	2.5	2.4



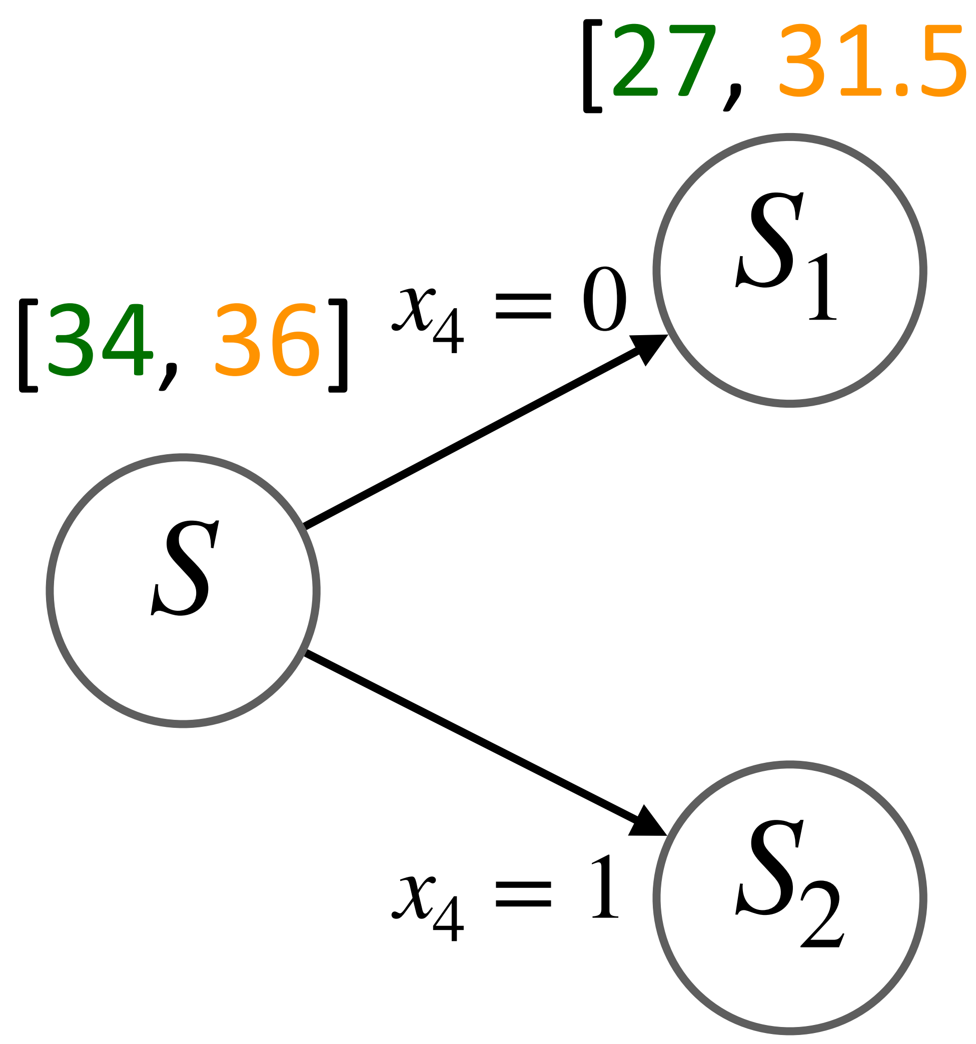
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **1** **1**	5
outlay/return	2	1.5	2.333	2.5	2.4



Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **1** **1**	5
outlay/return	2	1.5	2.333	2.5	2.4

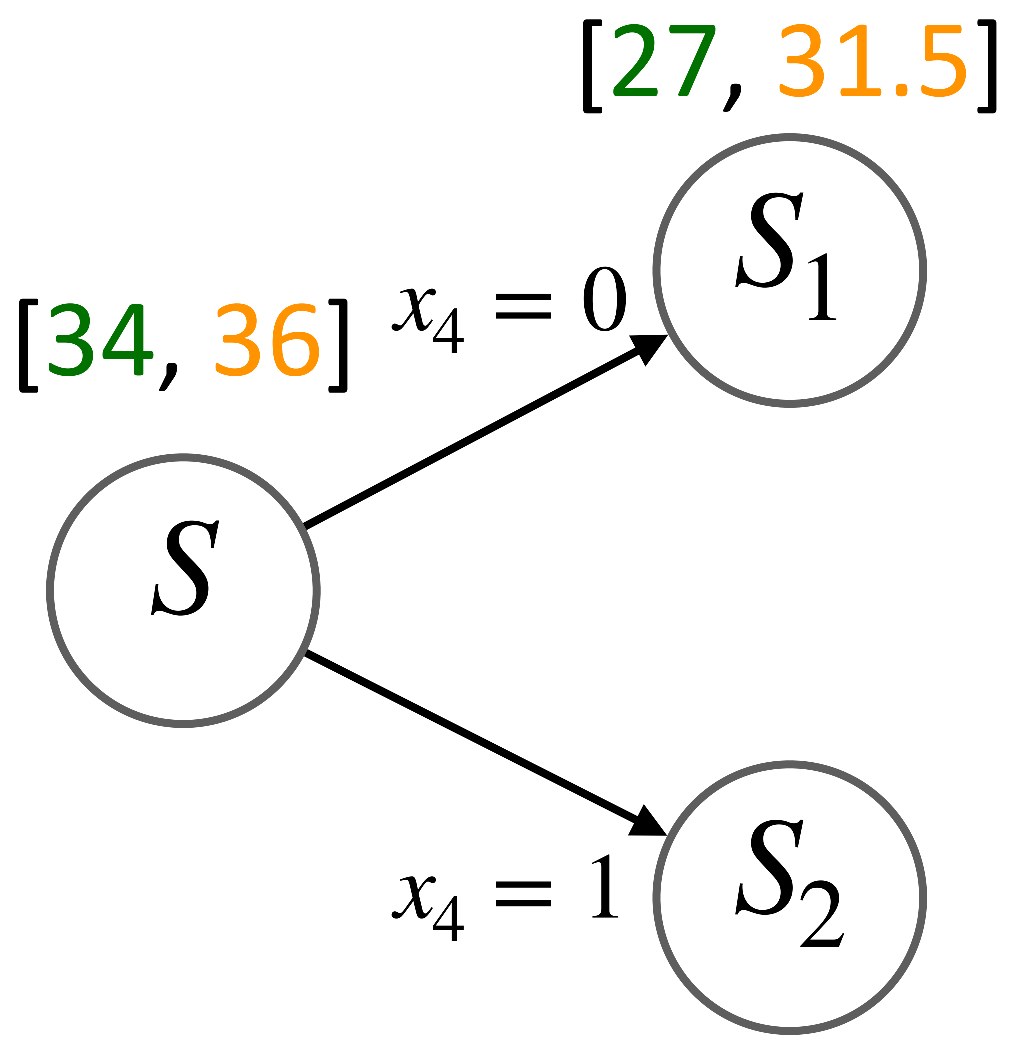


total outlay: 6

Use Branch-and-Bound to solve Knapsack

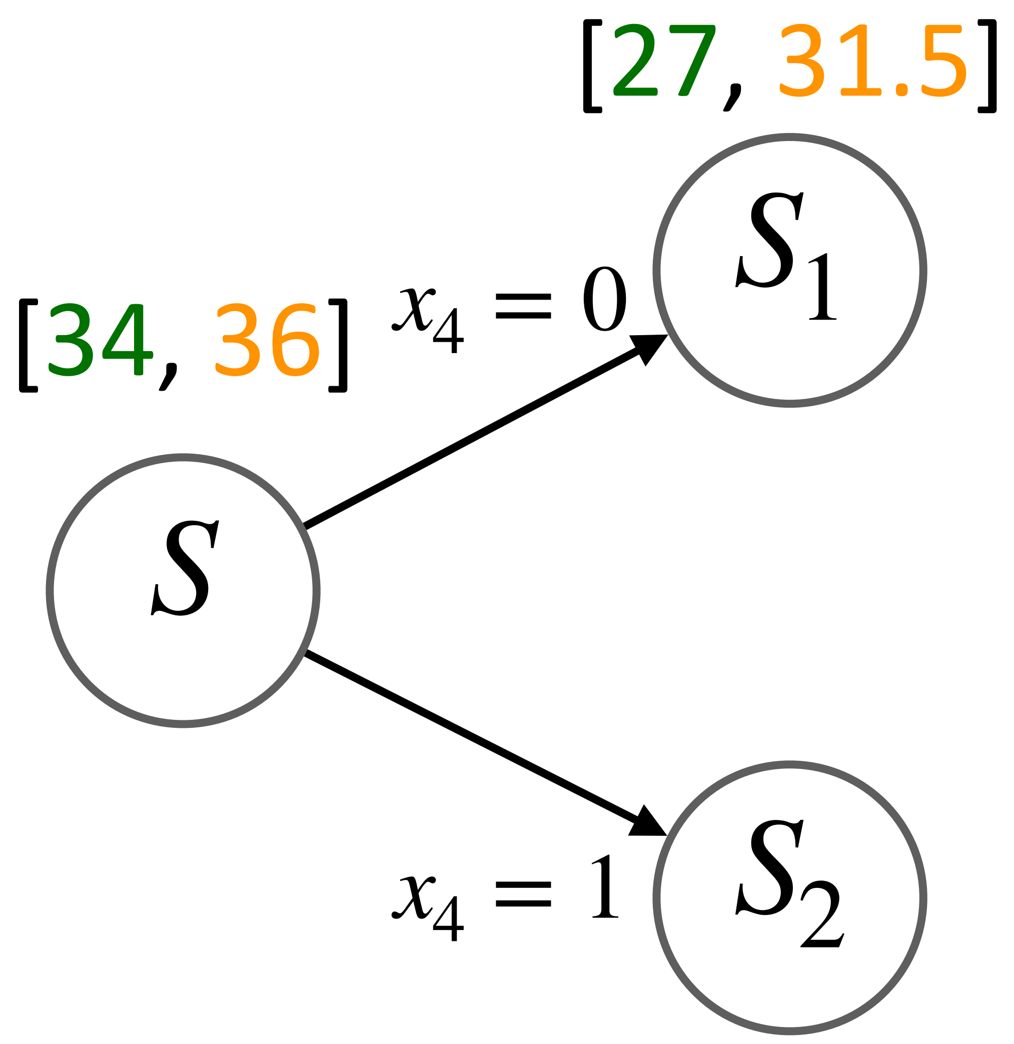
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **1** **1**	5 1
outlay/return	2	1.5	2.333	2.5	2.4

total outlay: 11



Use Branch-and-Bound to solve Knapsack

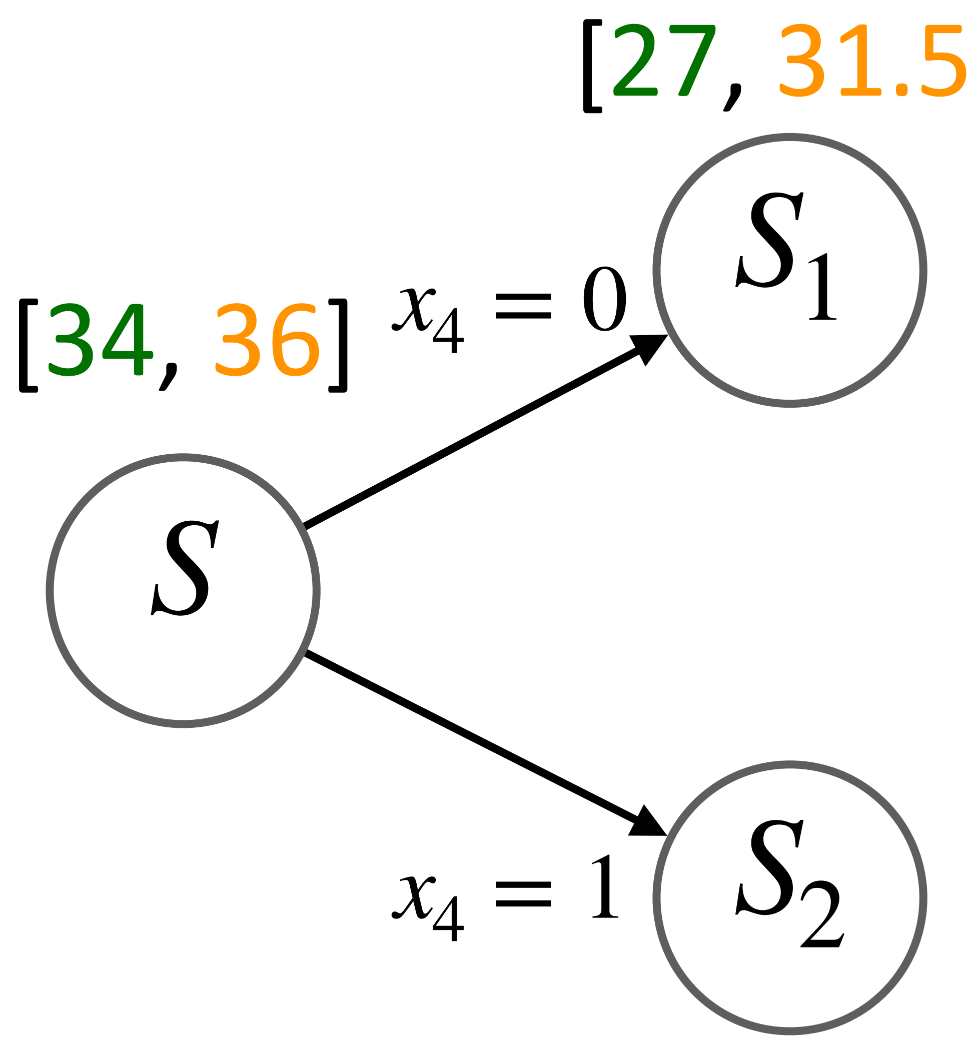
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 1	6 **1** **1**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 14

Use Branch-and-Bound to solve Knapsack

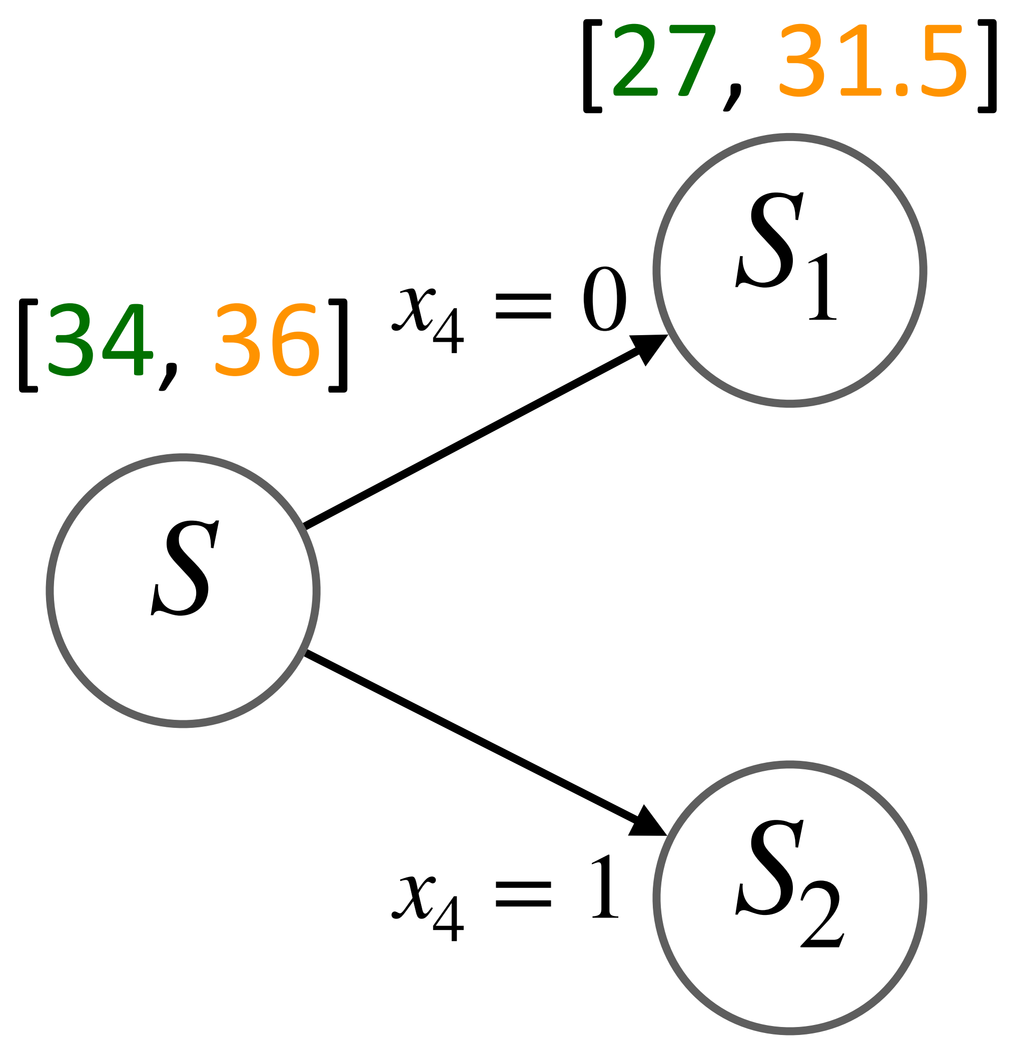
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8	3 1	6 **1** **1**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

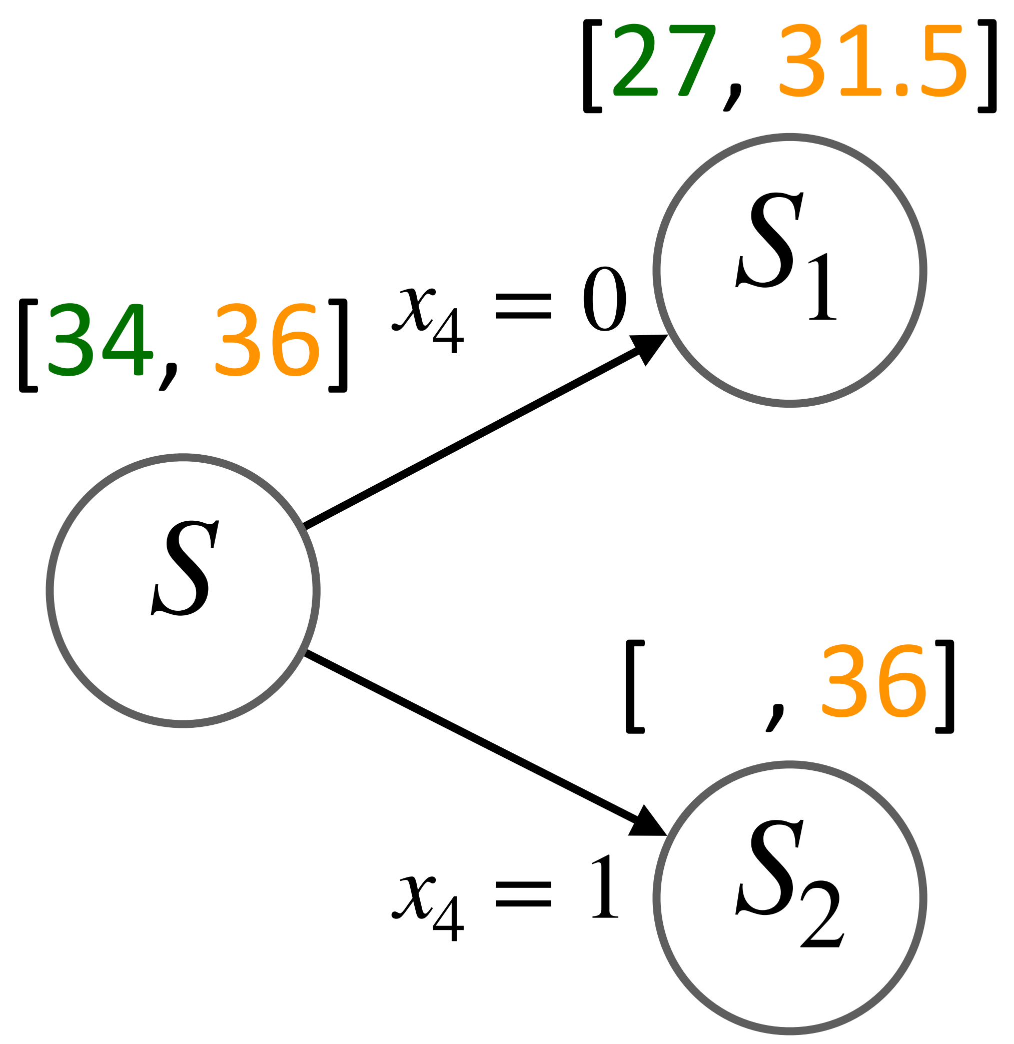
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8 0	3 1	6 **1** **1**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

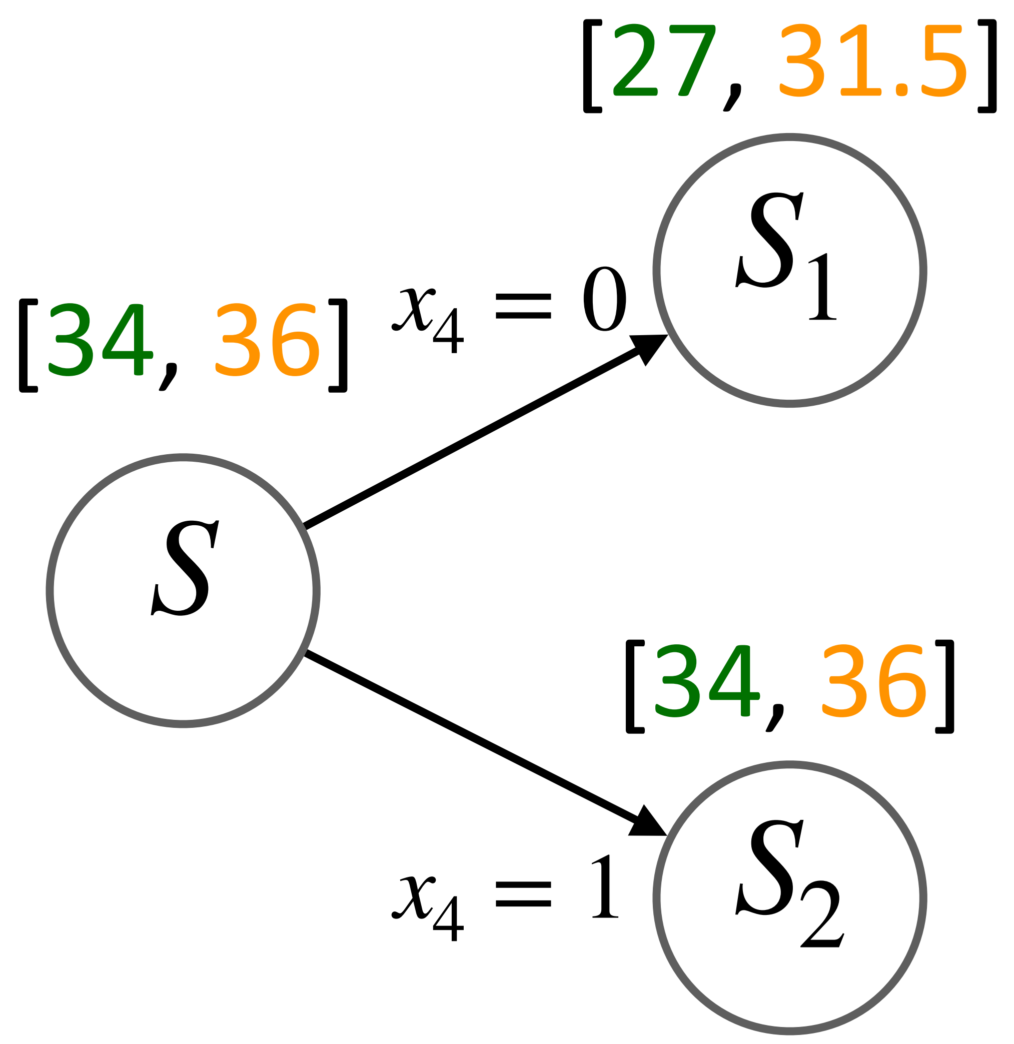
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8 0	3 1	6 **1** **1**	5 1
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

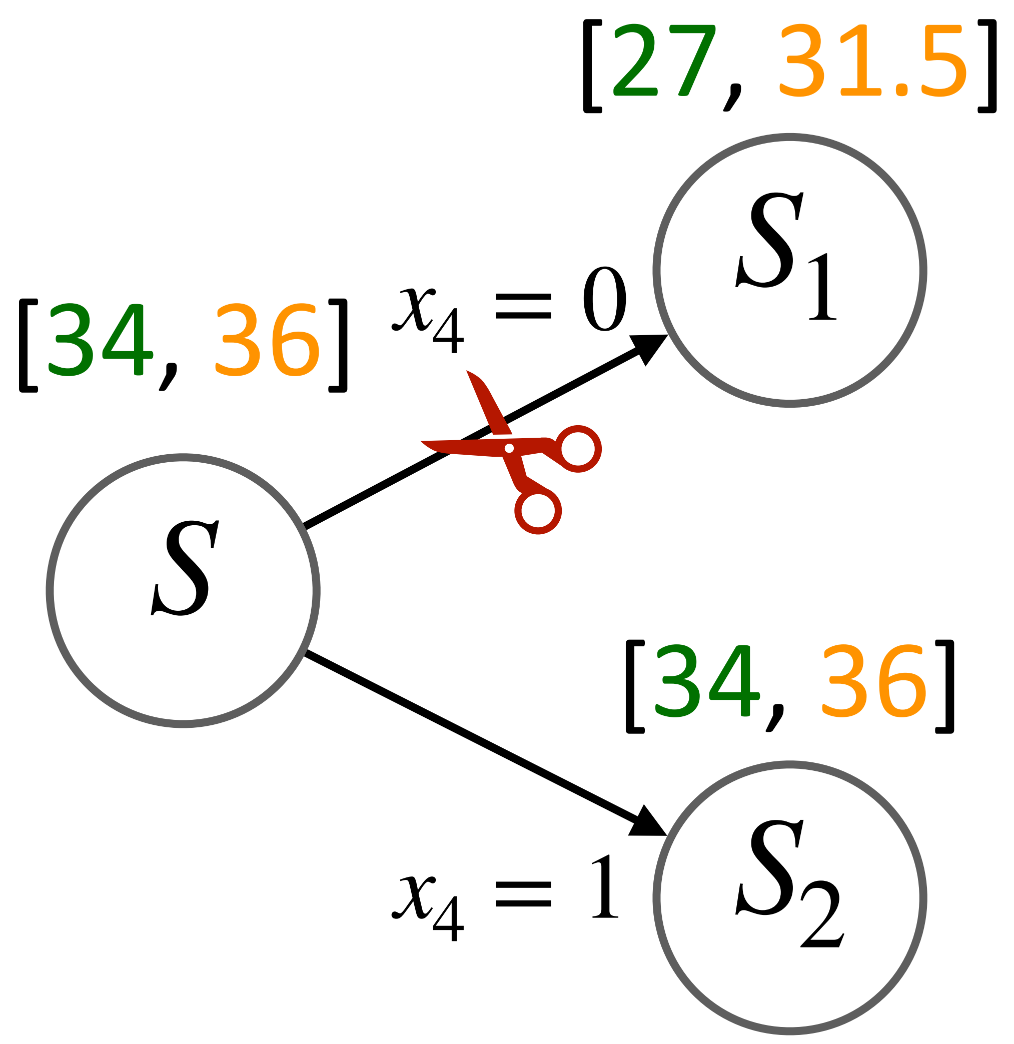
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 1 1	6 **1** **1**	5 1 1
outlay/return	2	1.5	2.333	2.5	2.4



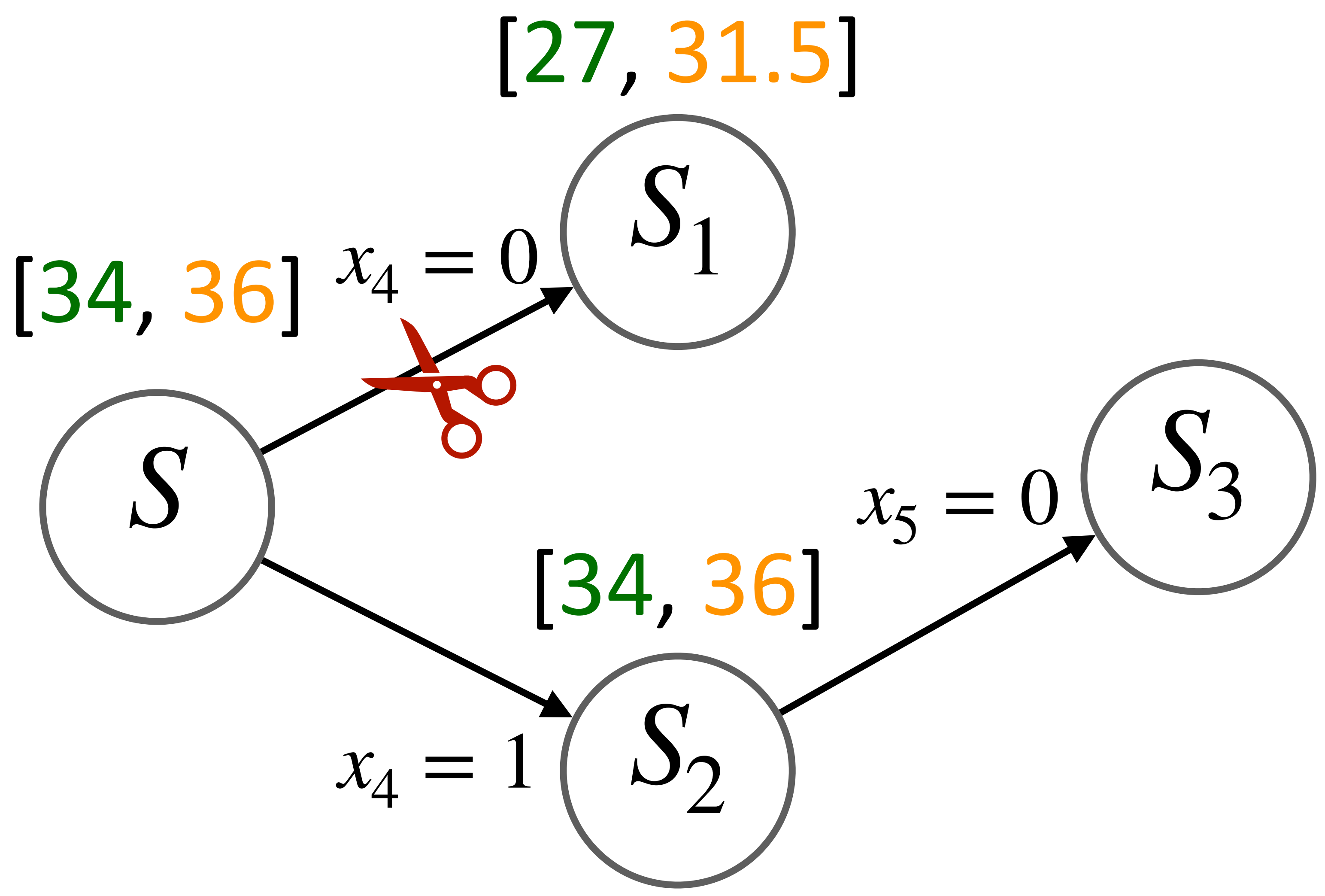
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 1 1	6 **1** **1**	5 1 1
outlay/return	2	1.5	2.333	2.5	2.4



Use Branch-and-Bound to solve Knapsack

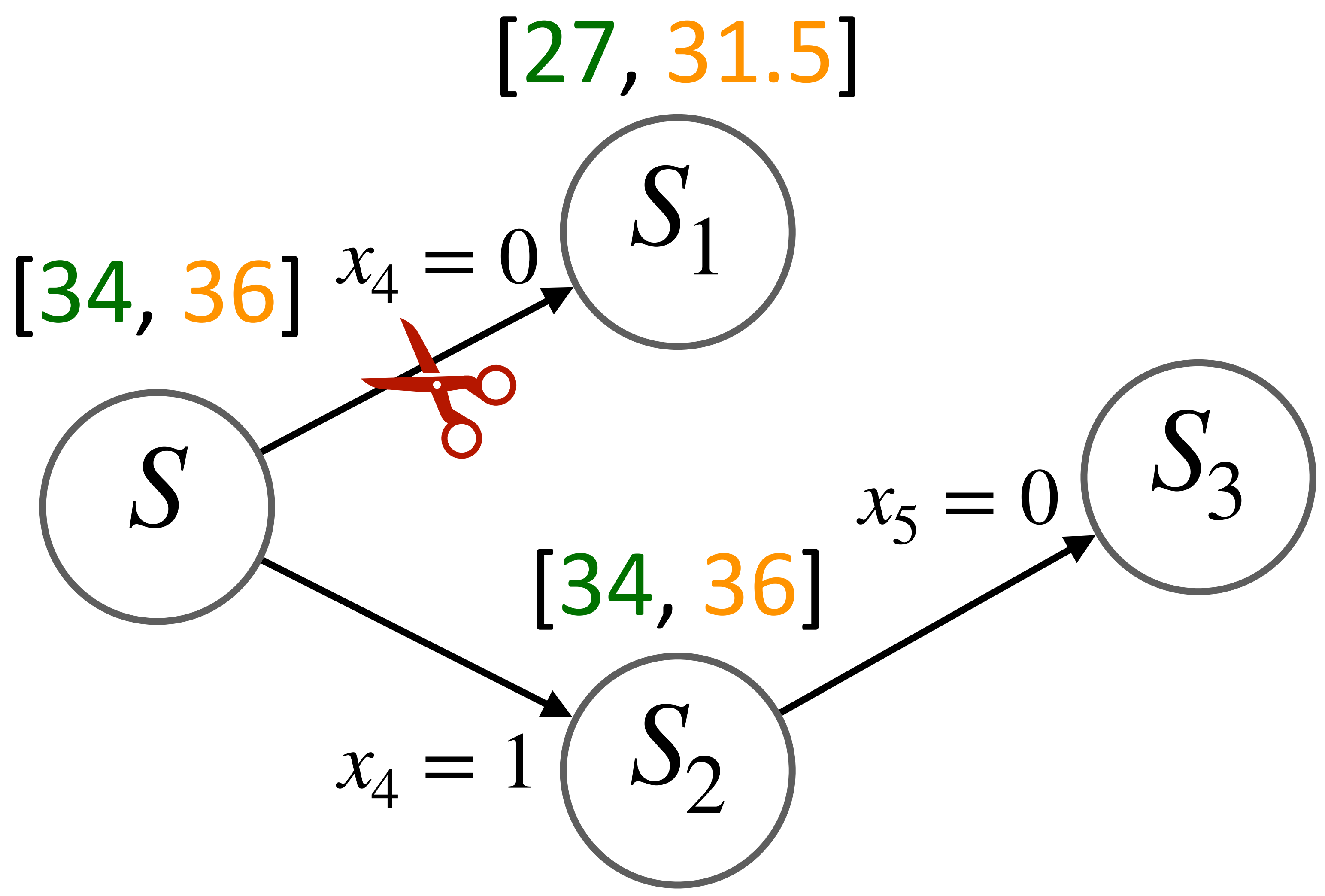
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **1** **1**	5 **0** **0**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 6

Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 **1**	6 **1** **1**	5 **0** **0**
outlay/return	2	1.5	2.333	2.5	2.4

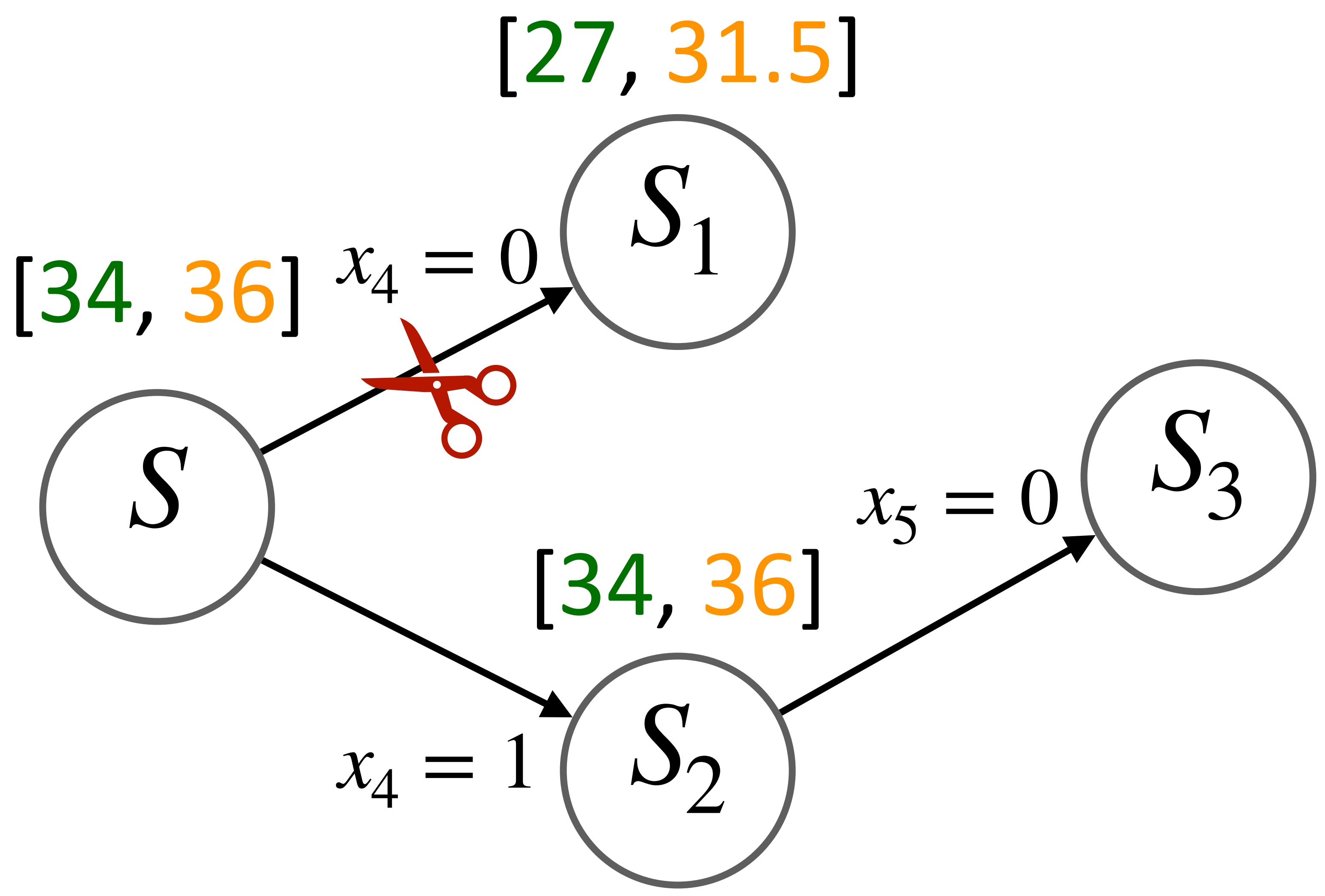


total outlay: 9

Use Branch-and-Bound to solve Knapsack

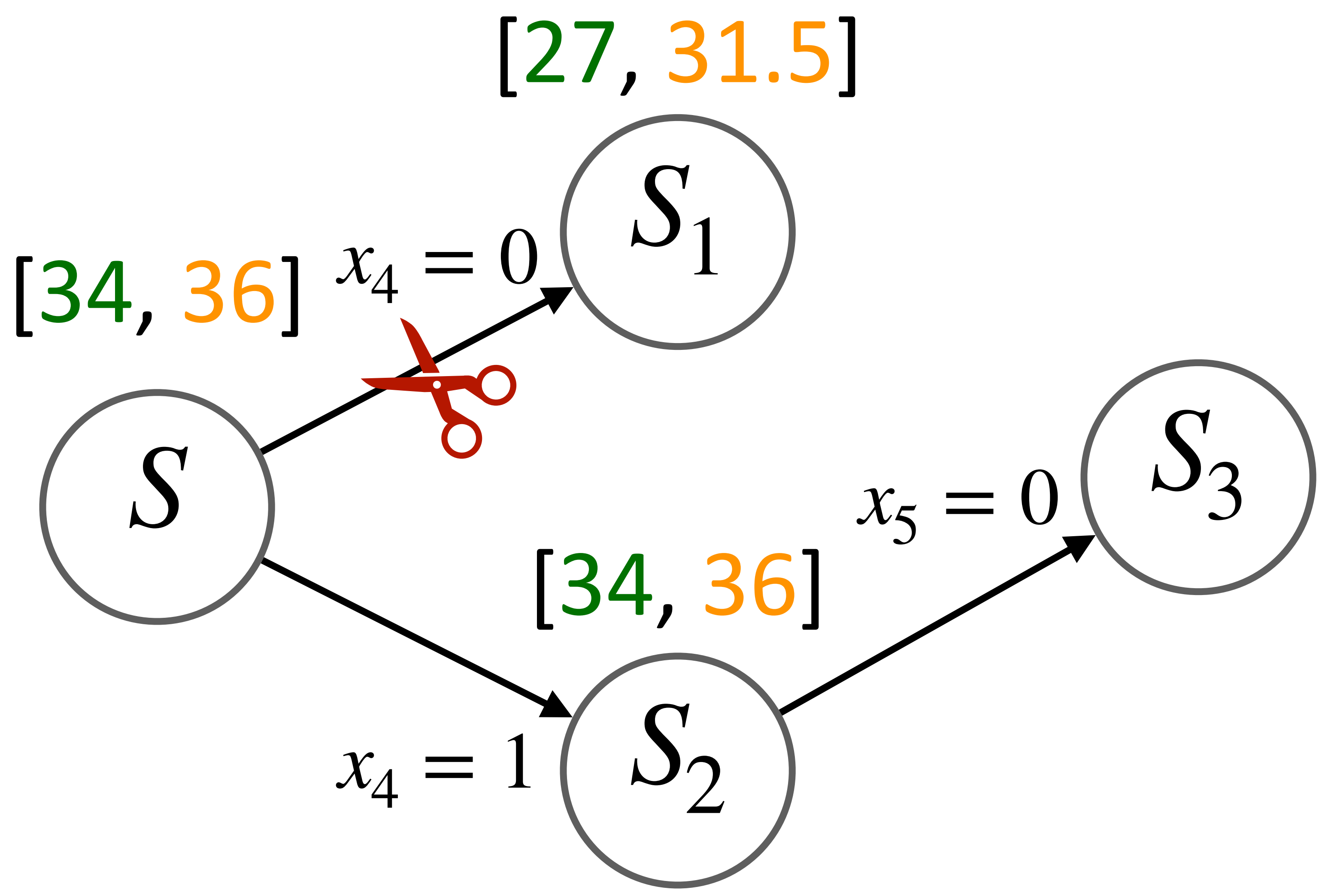
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8	3 1	6 **1** **1**	5 **0** **0**
outlay/return	2	1.5	2.333	2.5	2.4

total outlay: 13



Use Branch-and-Bound to solve Knapsack

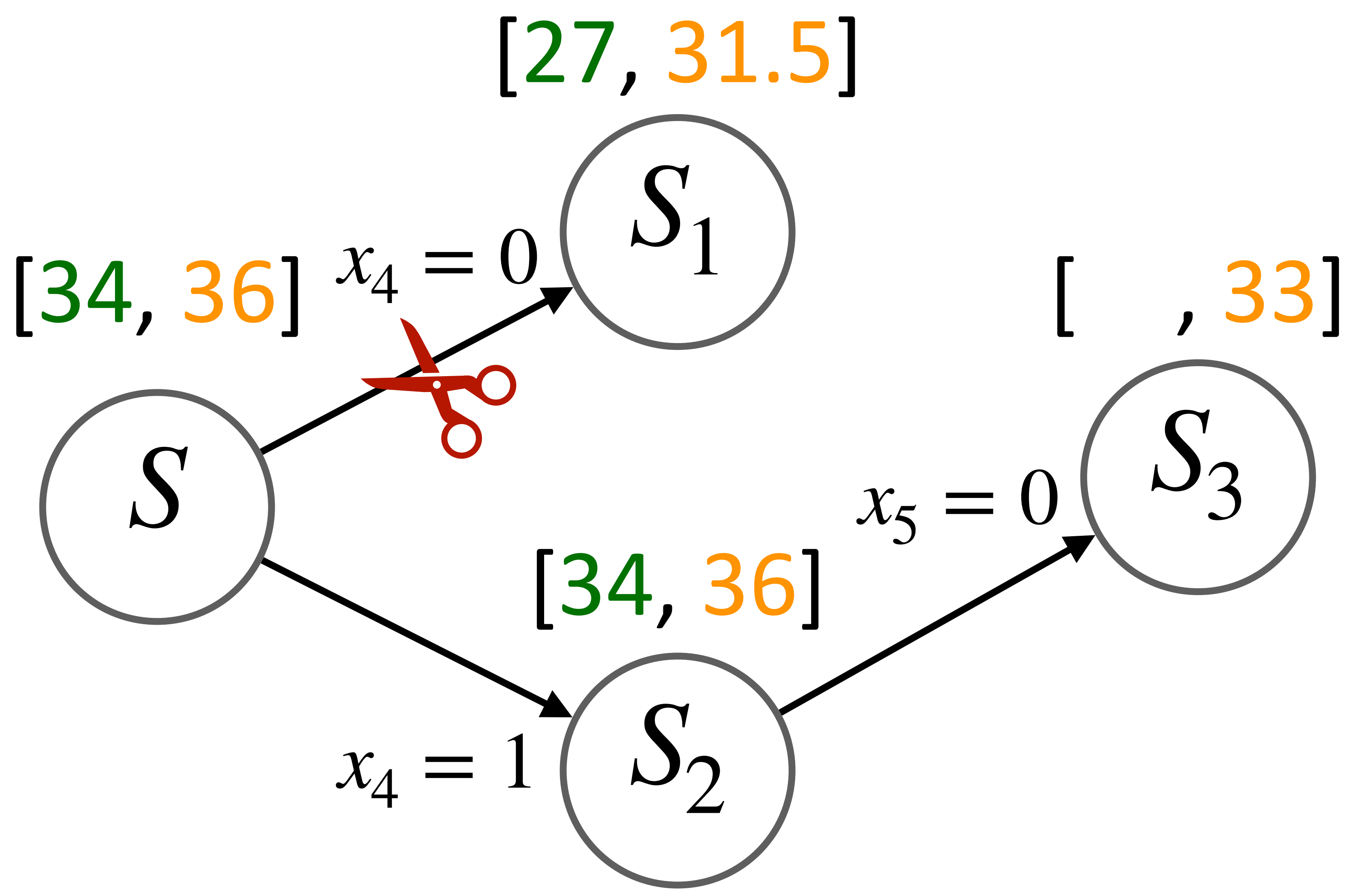
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8 $\frac{1}{4}$	3 1	6 **1** **1**	5 **0** **0**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

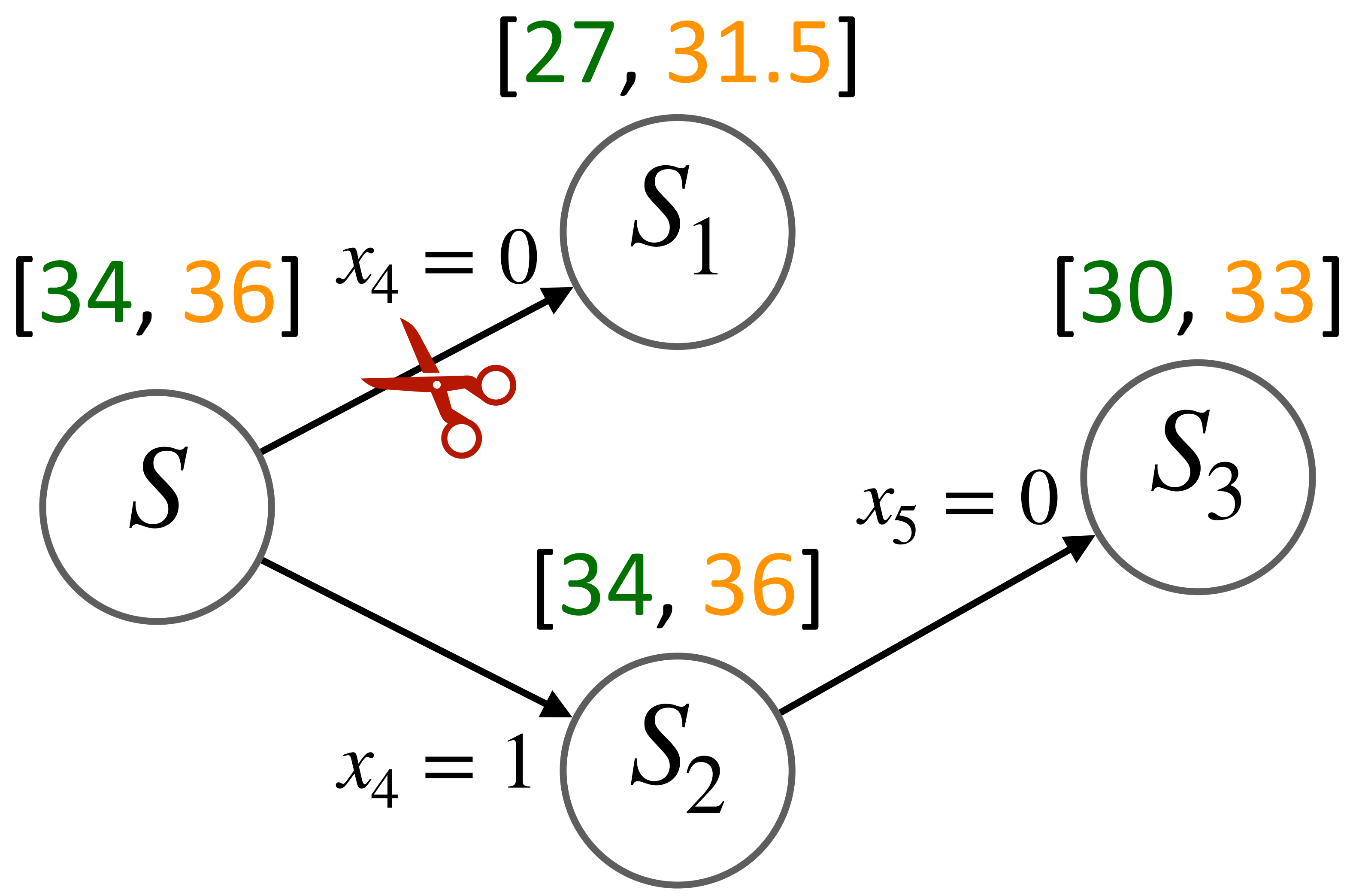
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8 $\frac{1}{4}$	3 1	6 **1** **1**	5 **0** **0**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

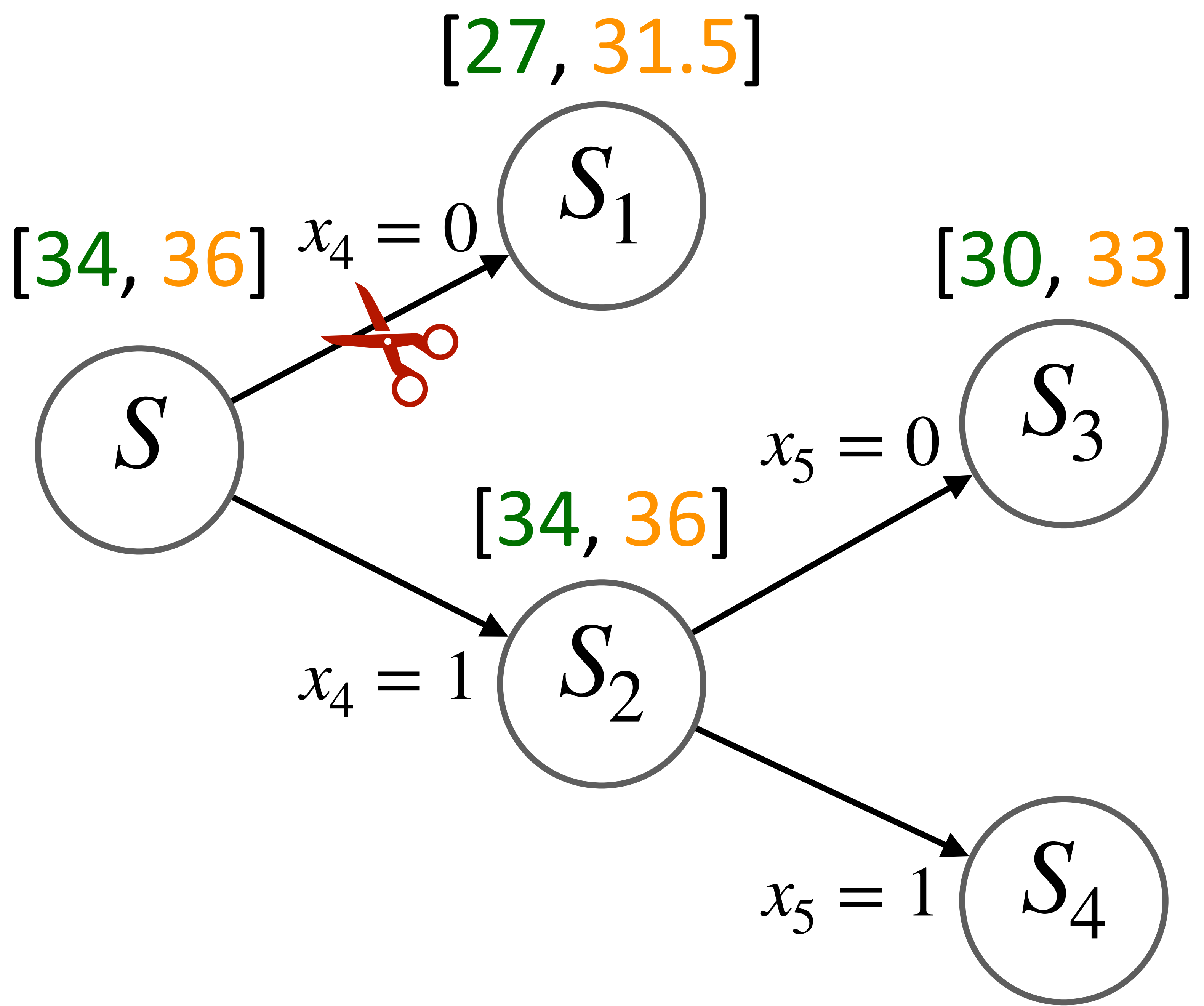
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1 1	8 $\frac{1}{4}$ 0	3 1 1	6 **1** **1**	5 **0** **0**
outlay/return	2	1.5	2.333	2.5	2.4



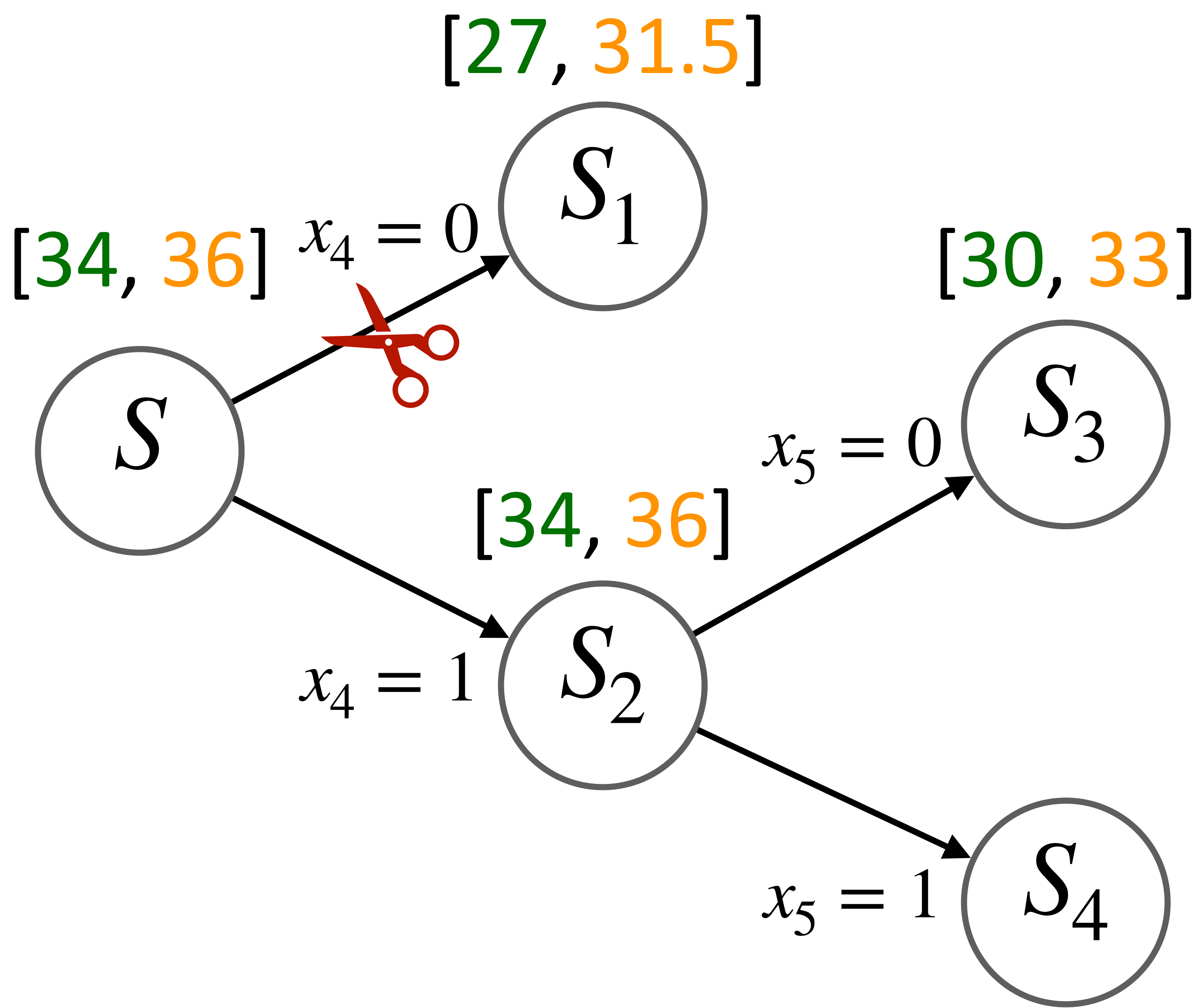
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



Use Branch-and-Bound to solve Knapsack

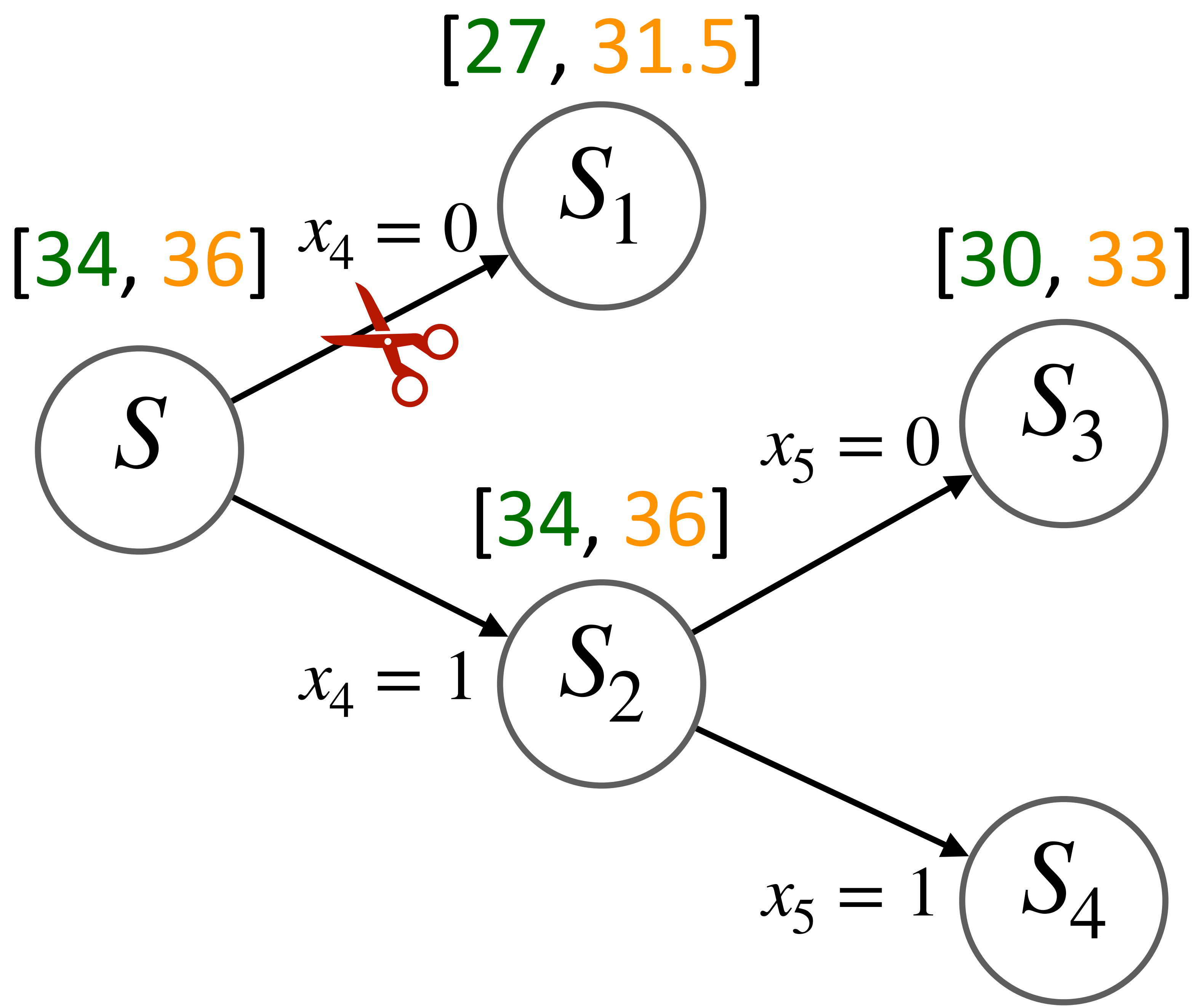
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 11

Use Branch-and-Bound to solve Knapsack

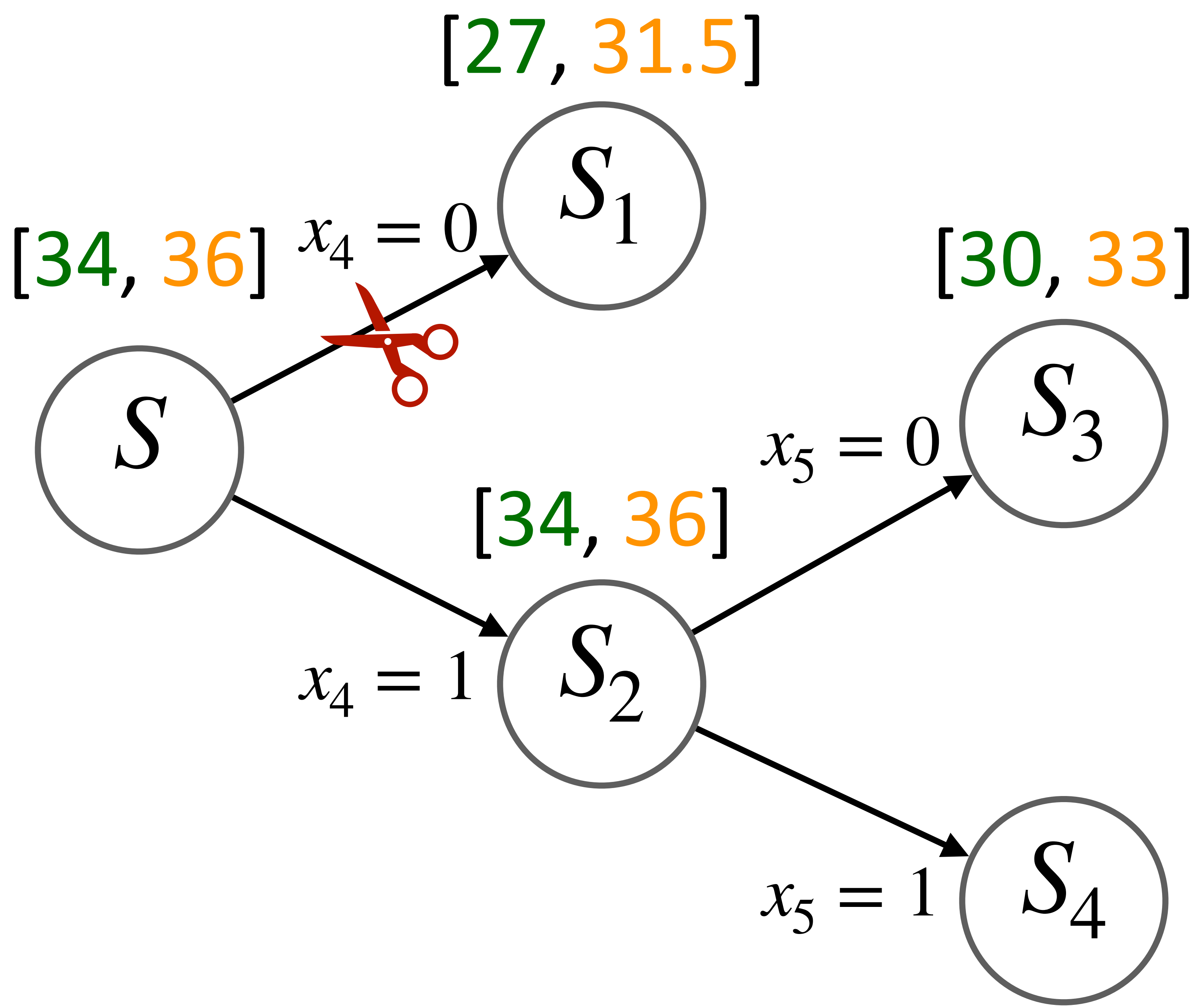
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 1	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 14

Use Branch-and-Bound to solve Knapsack

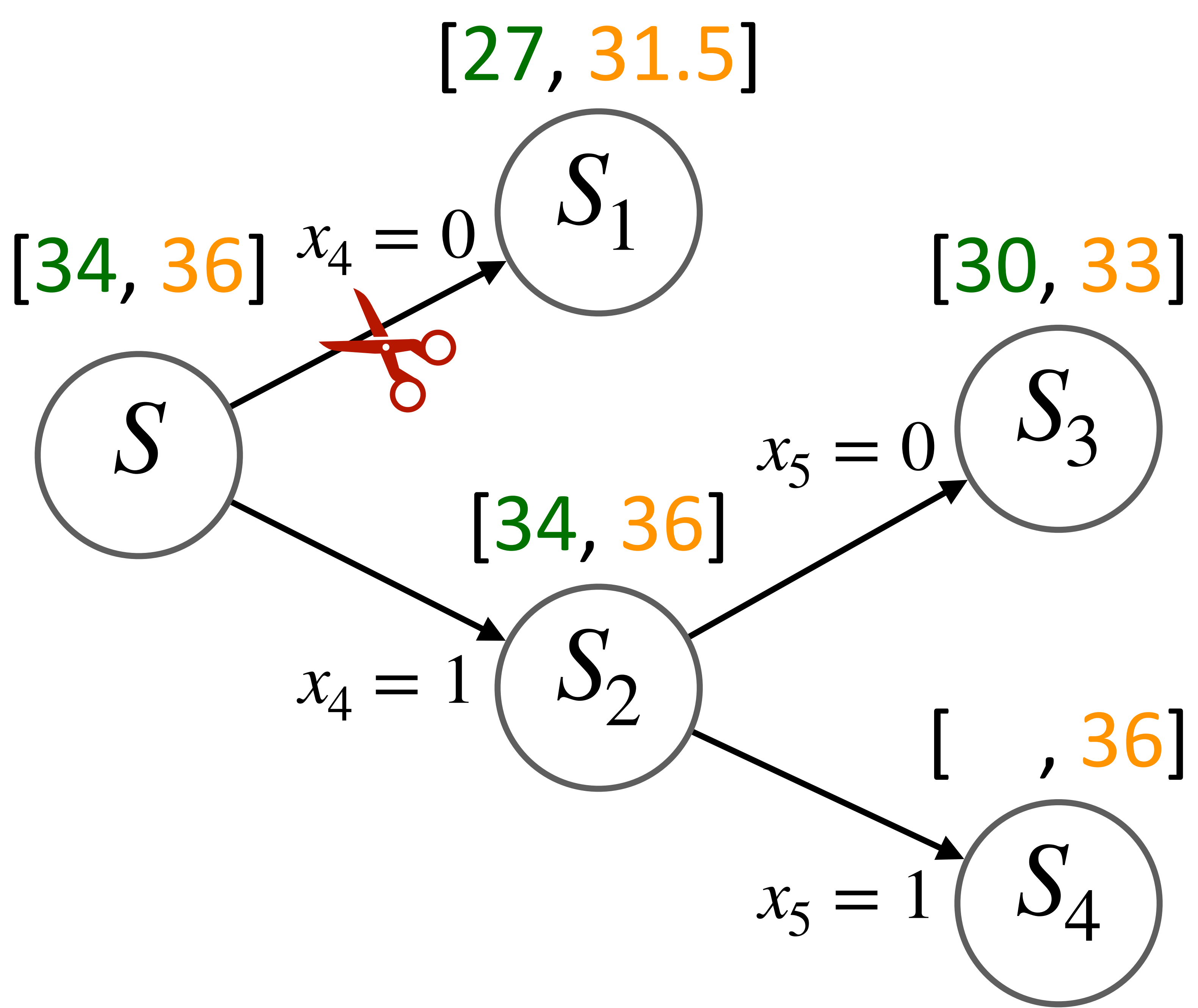
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8	3 1	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

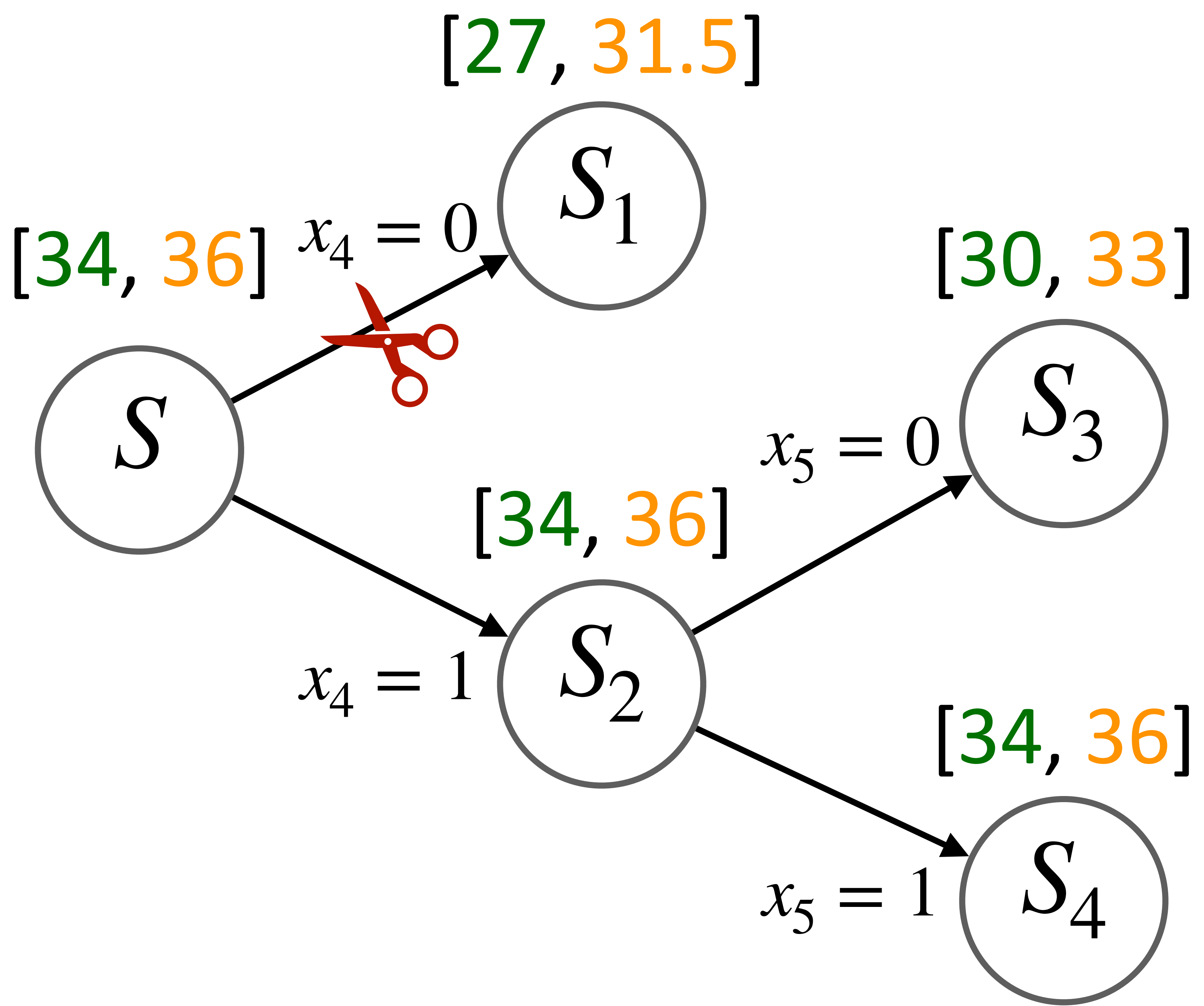
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8 0	3 1	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

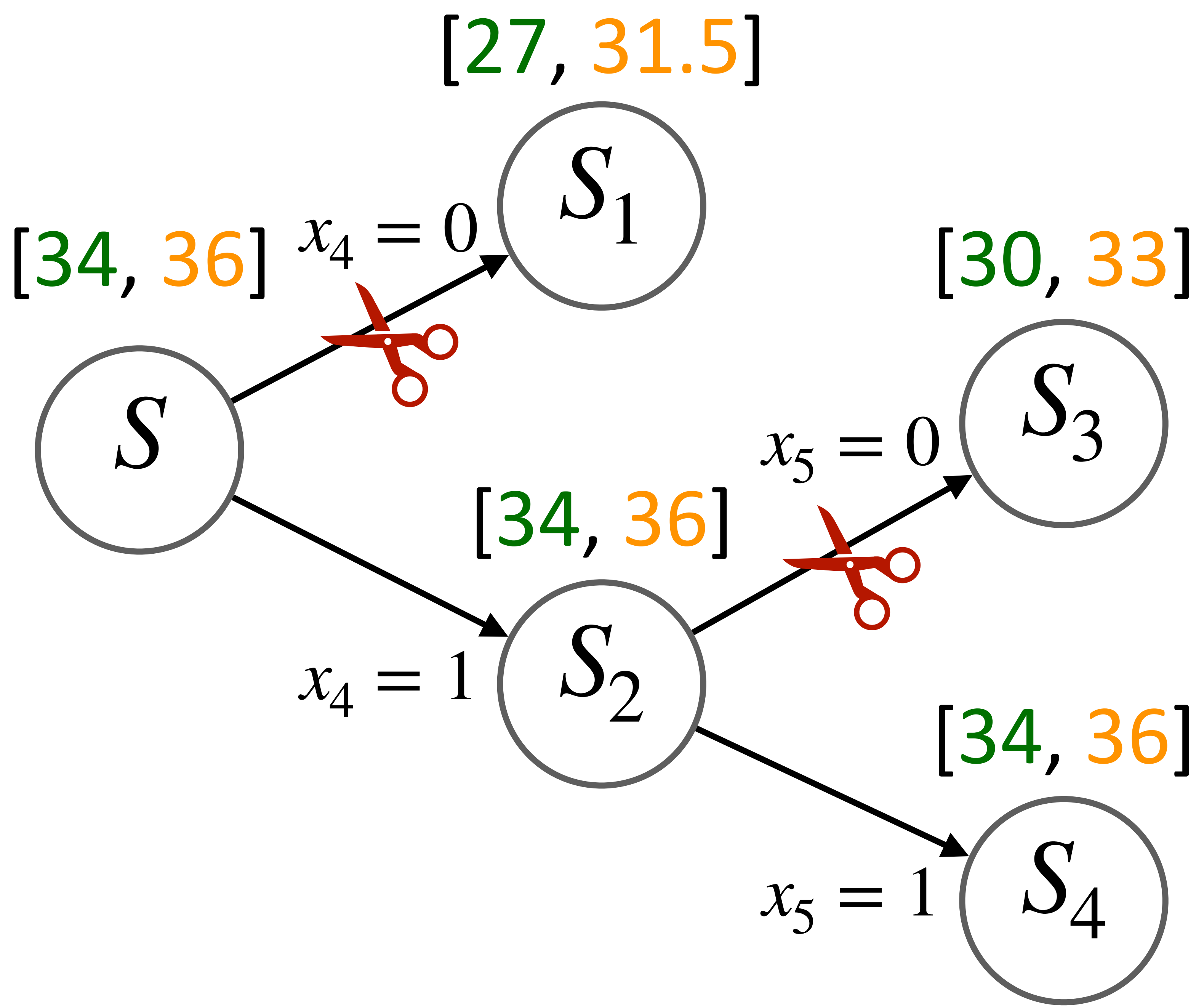
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 1 1	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

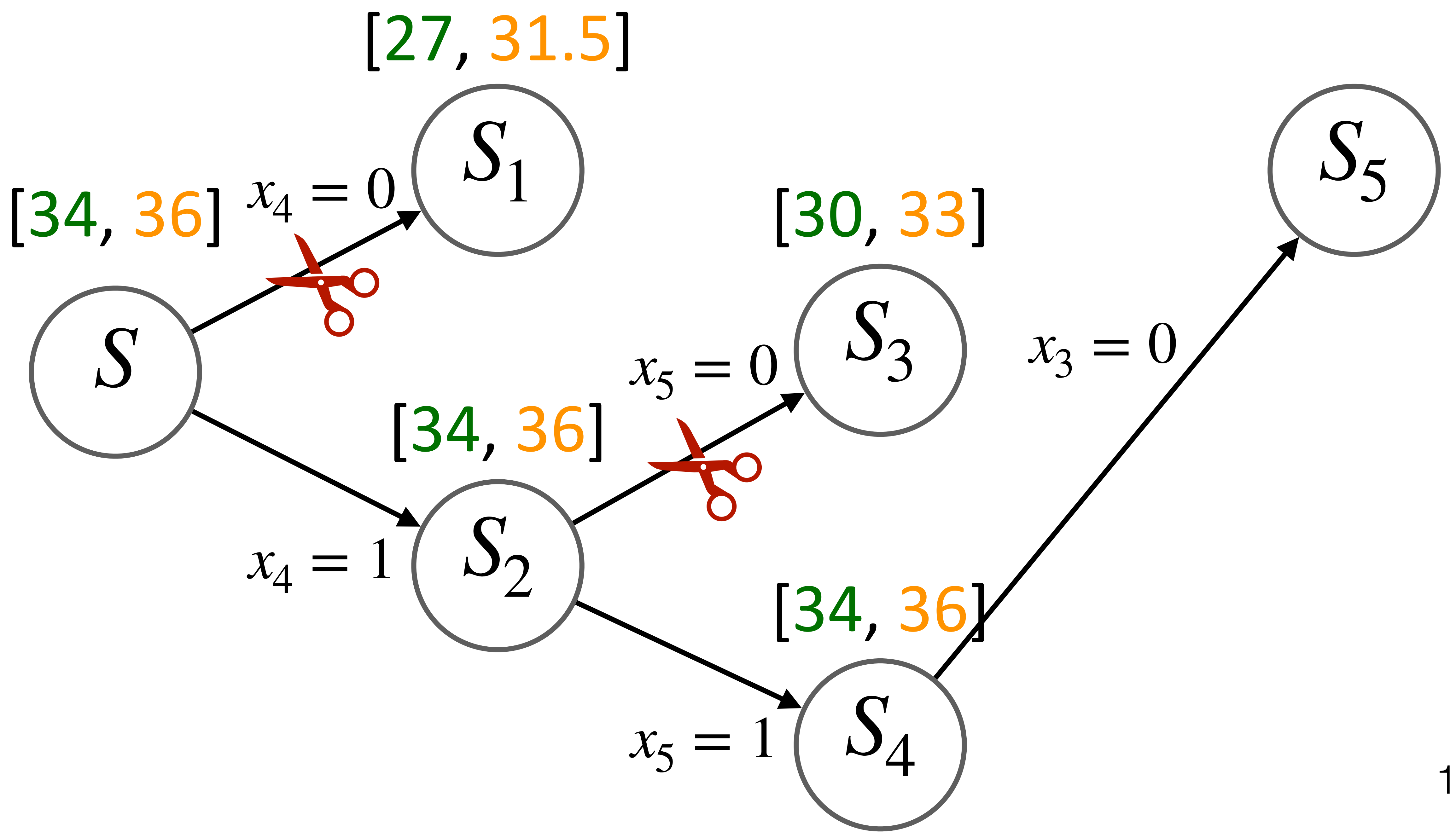
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 1 1	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

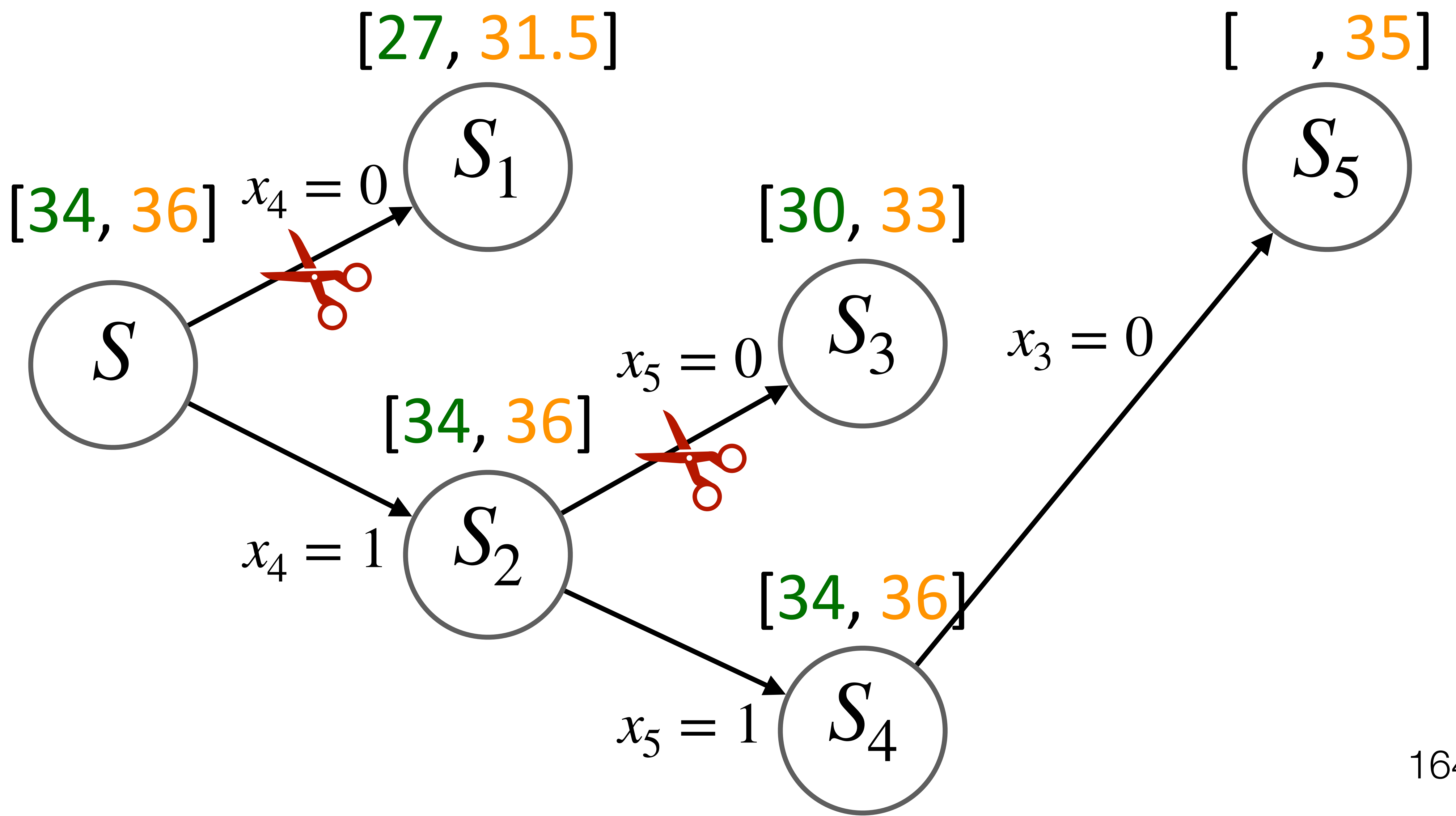
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 **0** **0**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



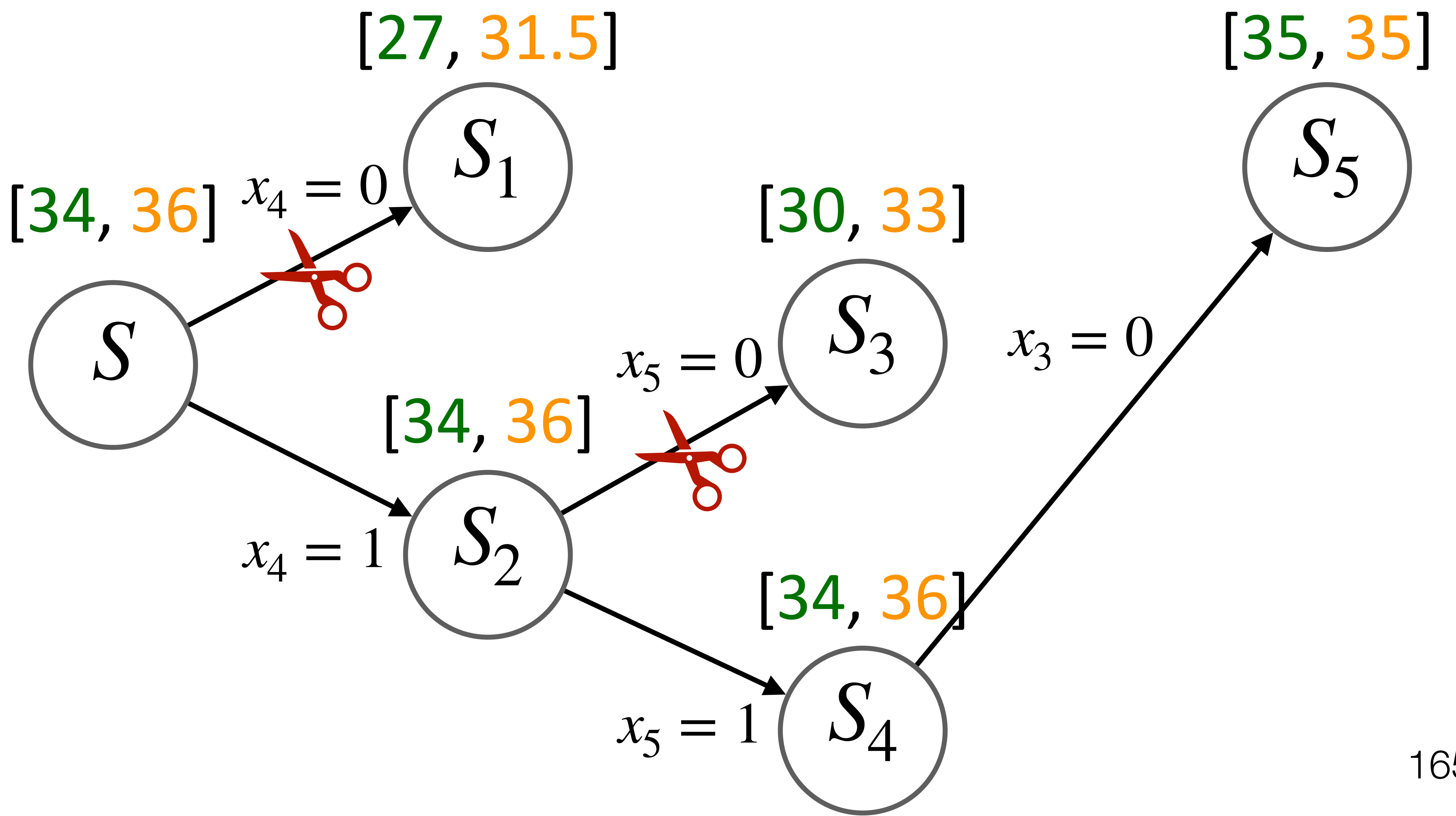
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1	8 0	3 **0** **0**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



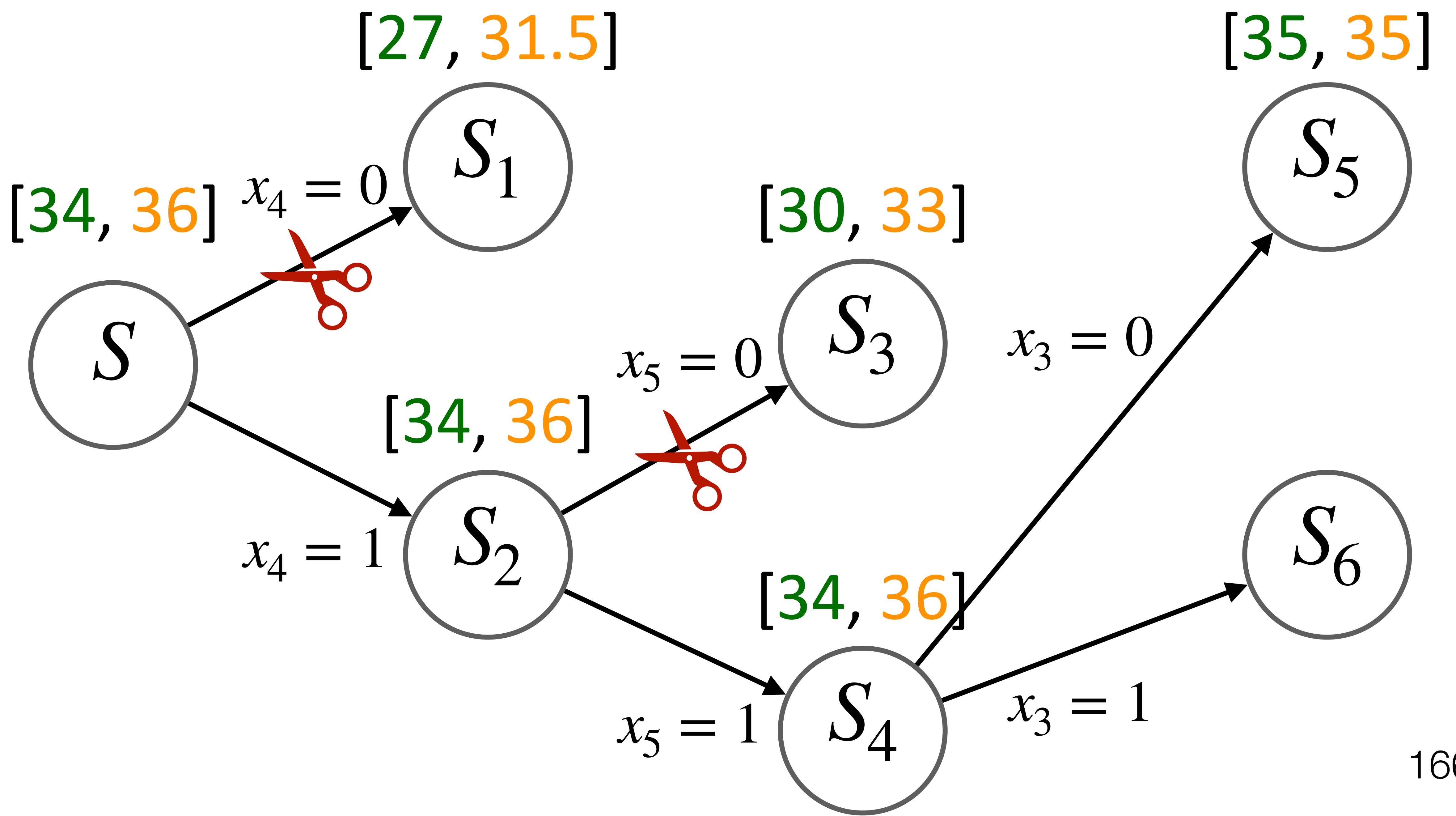
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1 1	8 0 0	3 **0** **0**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



Use Branch-and-Bound to solve Knapsack

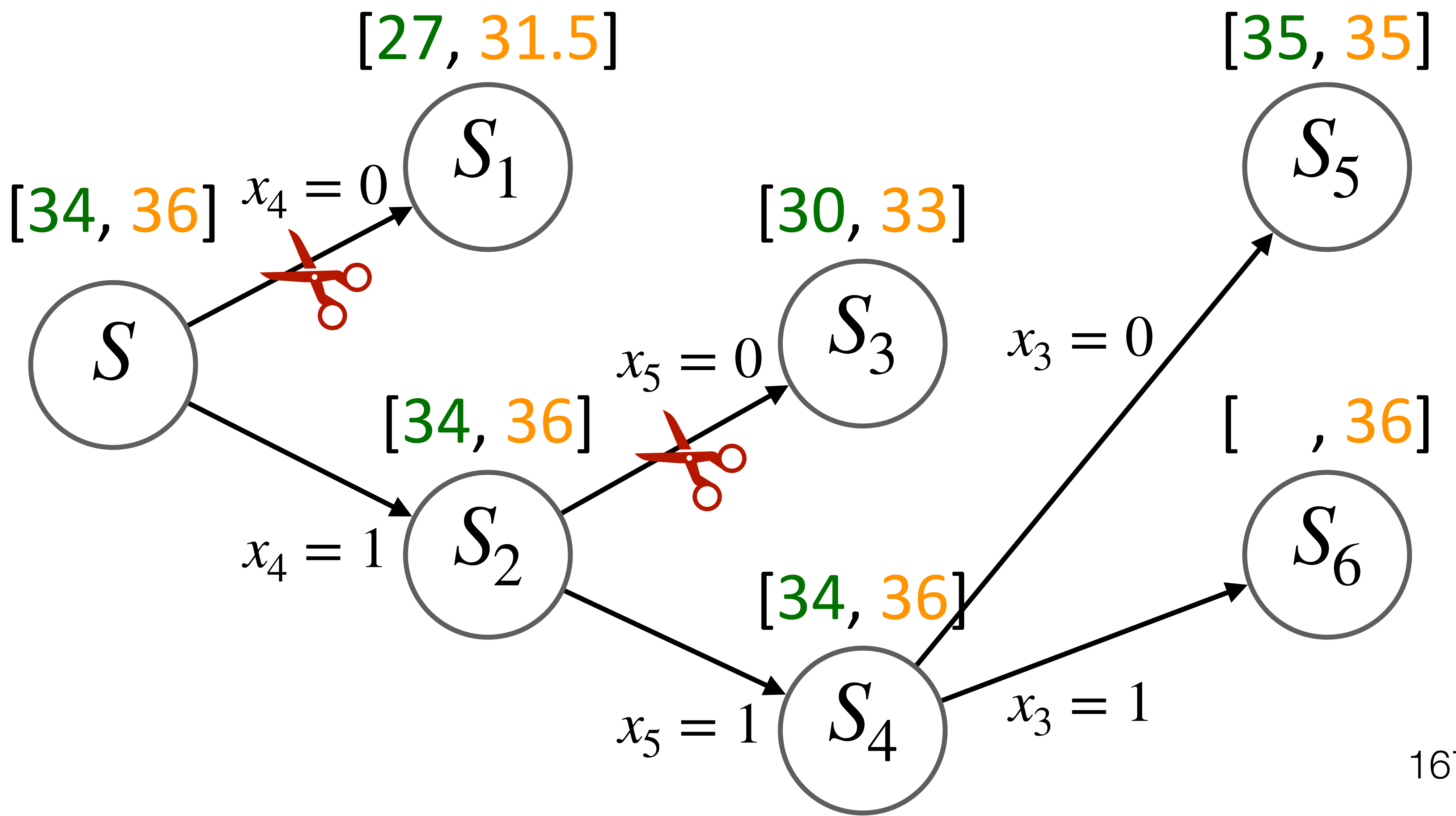
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

Use Branch-and-Bound to solve Knapsack

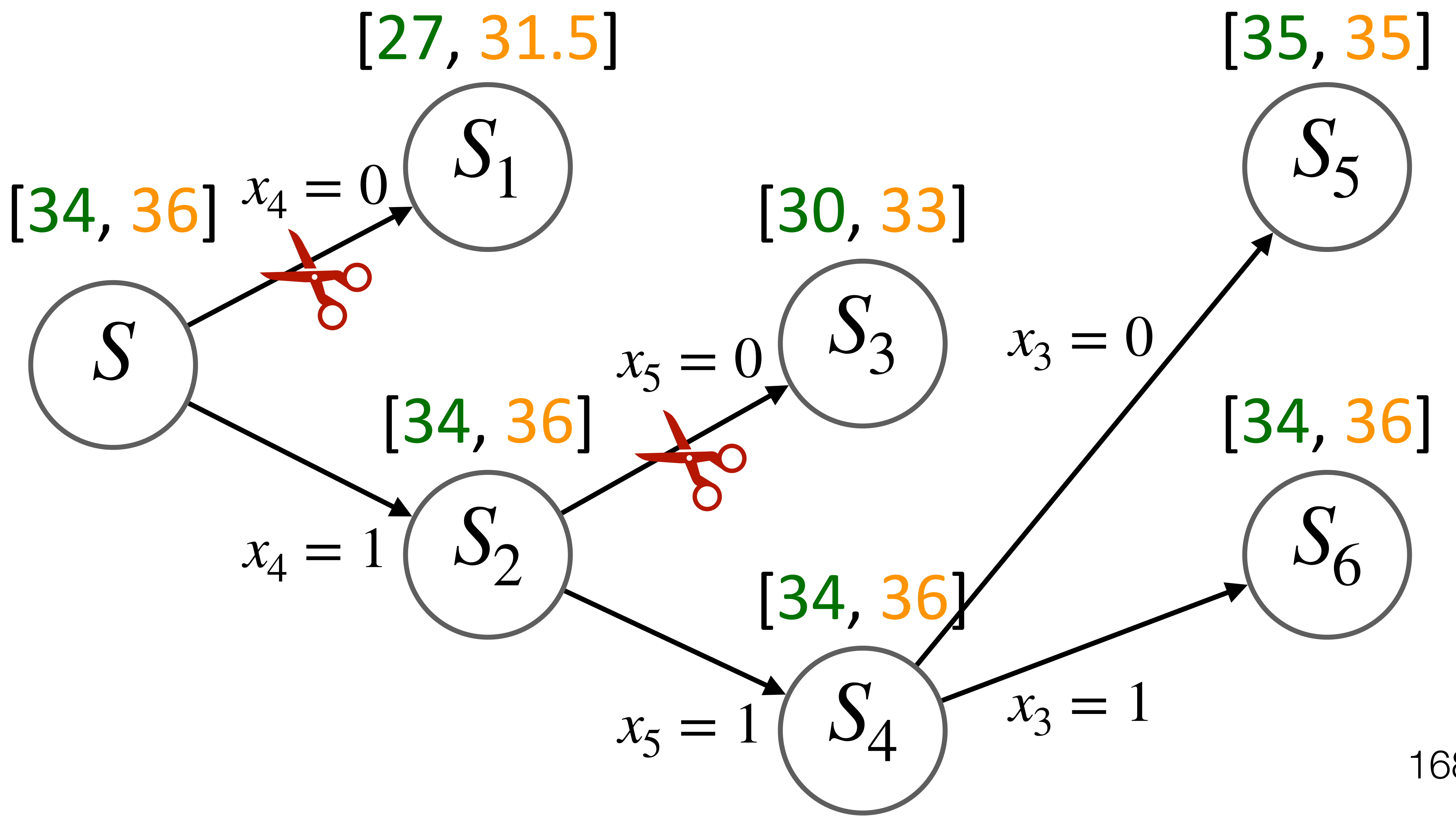
Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$	8 0	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



total outlay: 15

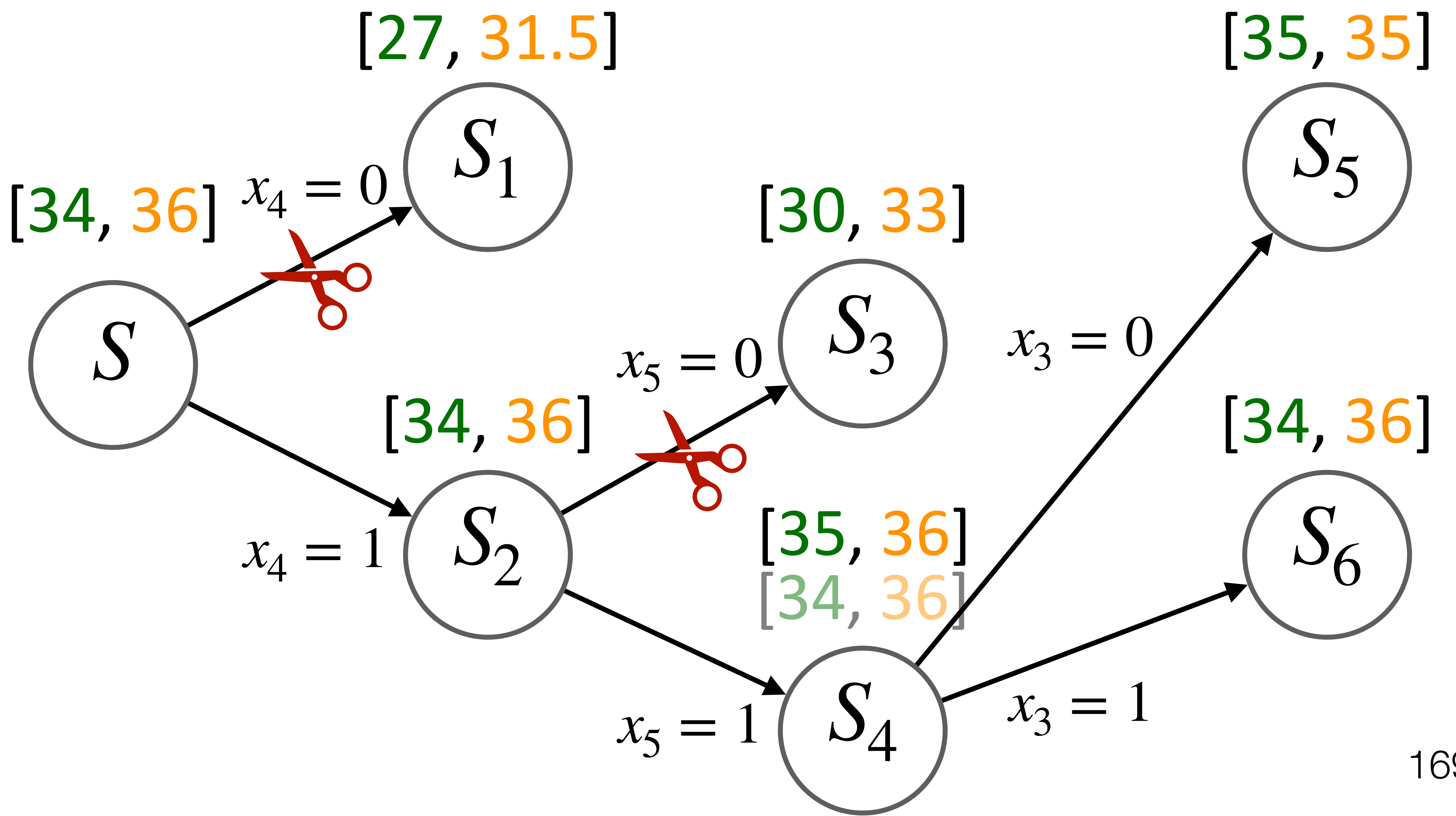
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



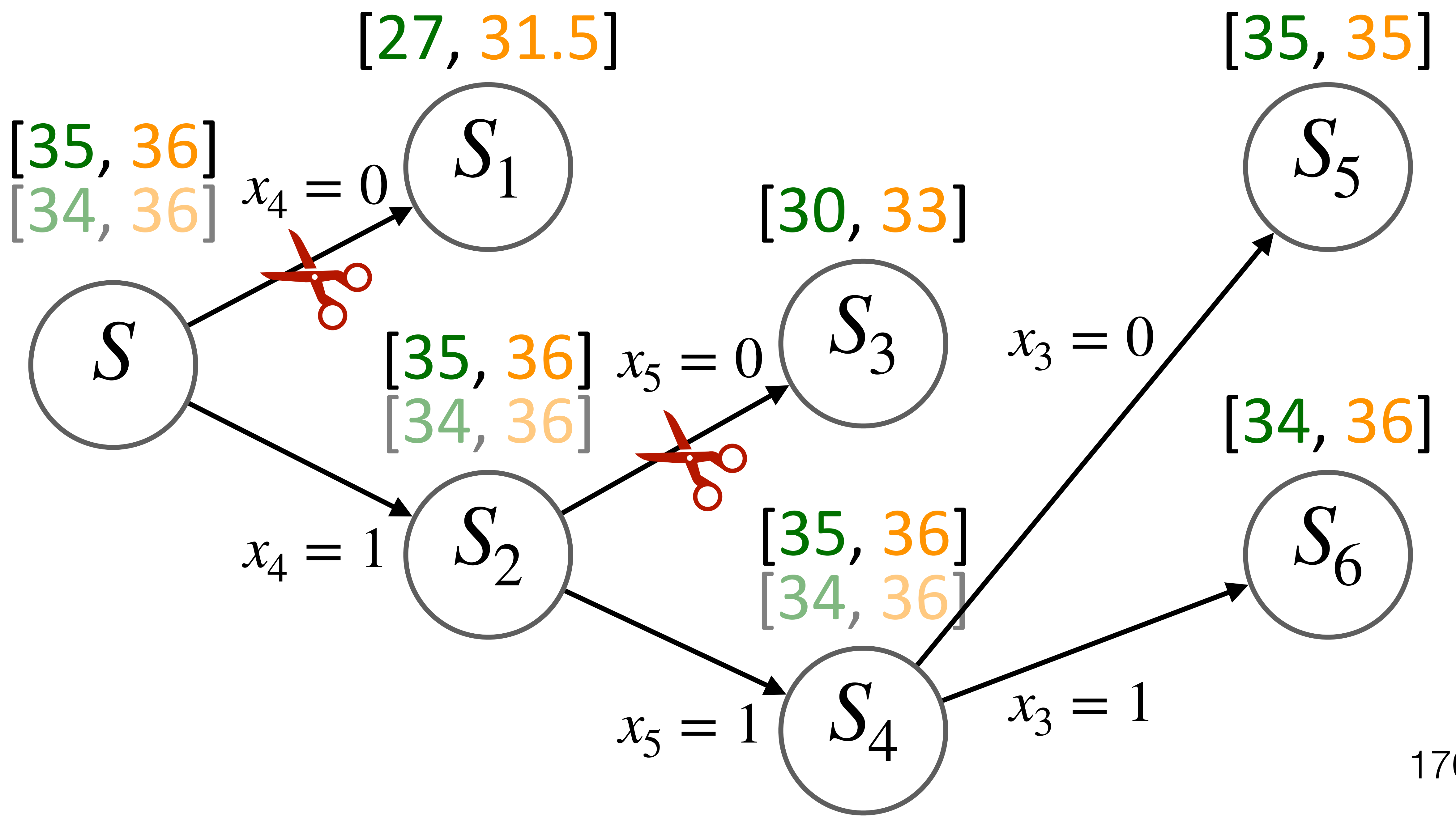
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



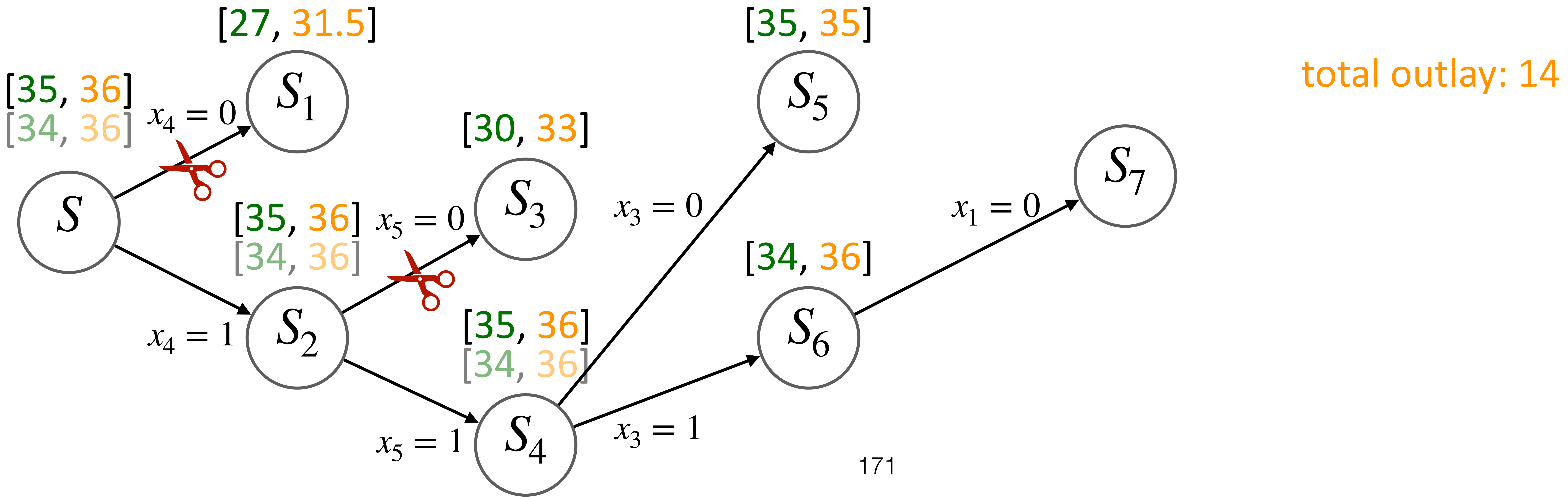
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 $\frac{1}{4}$ 0	8 0 0	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



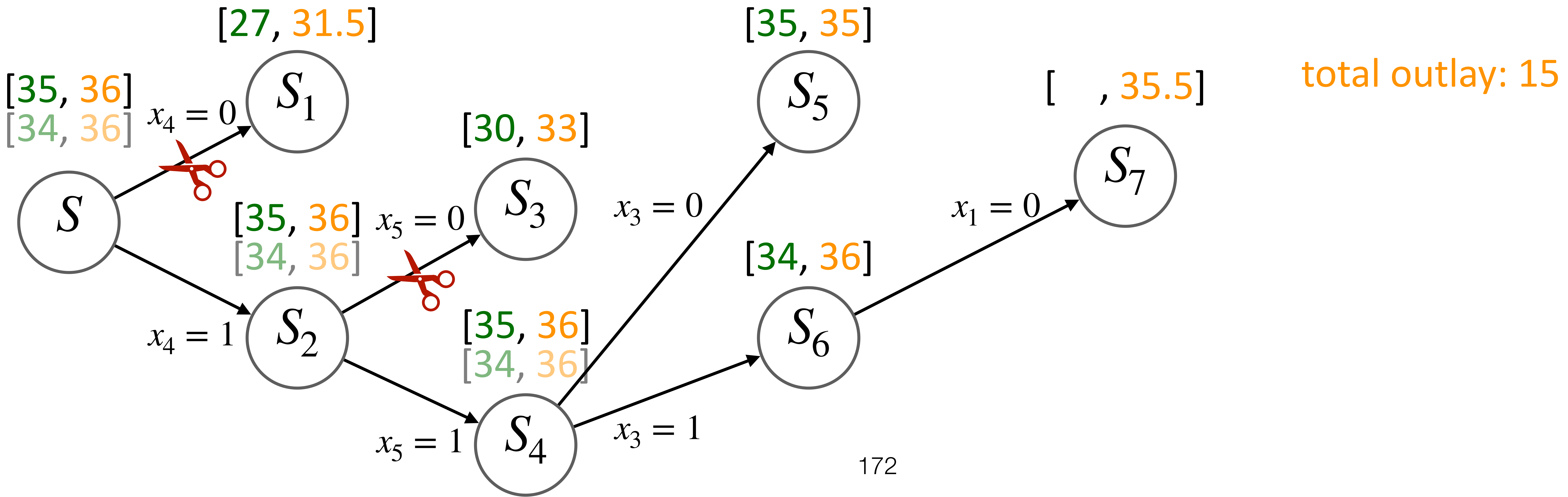
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



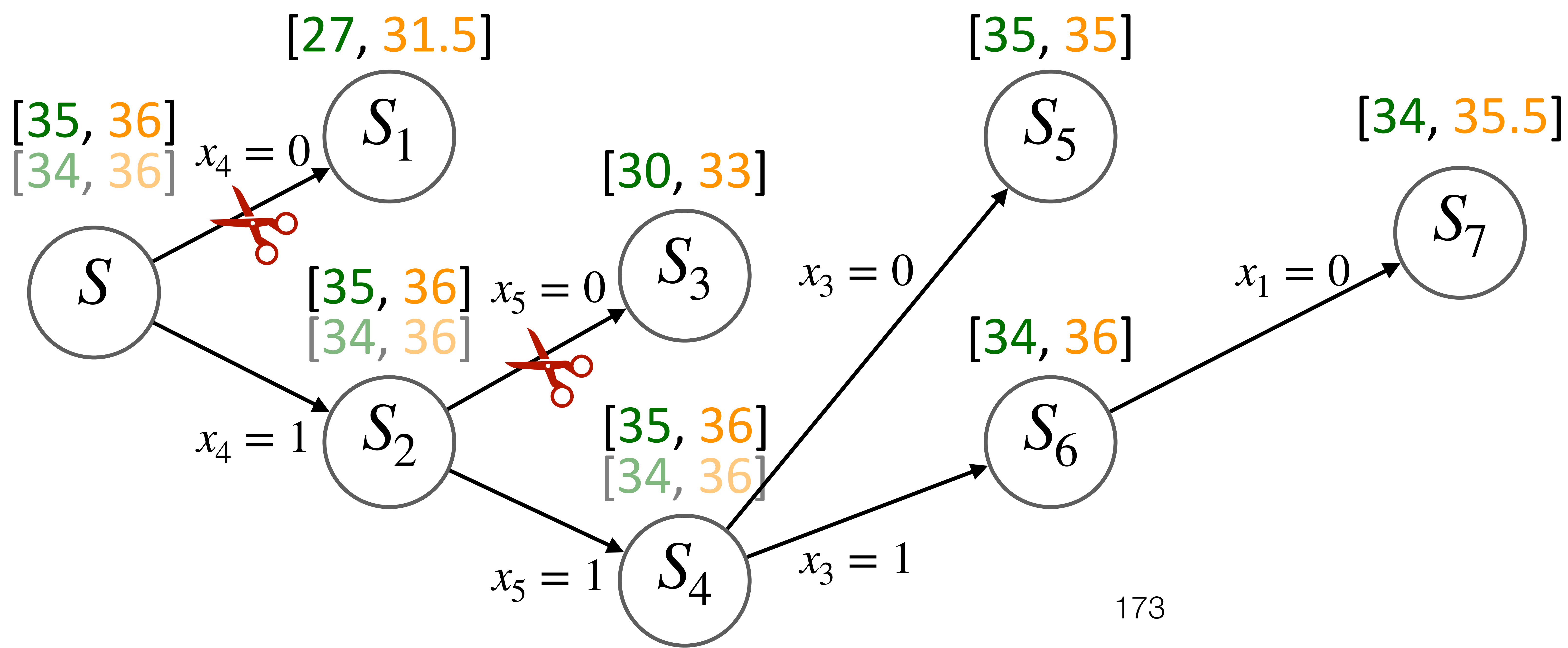
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 $\frac{1}{8}$	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



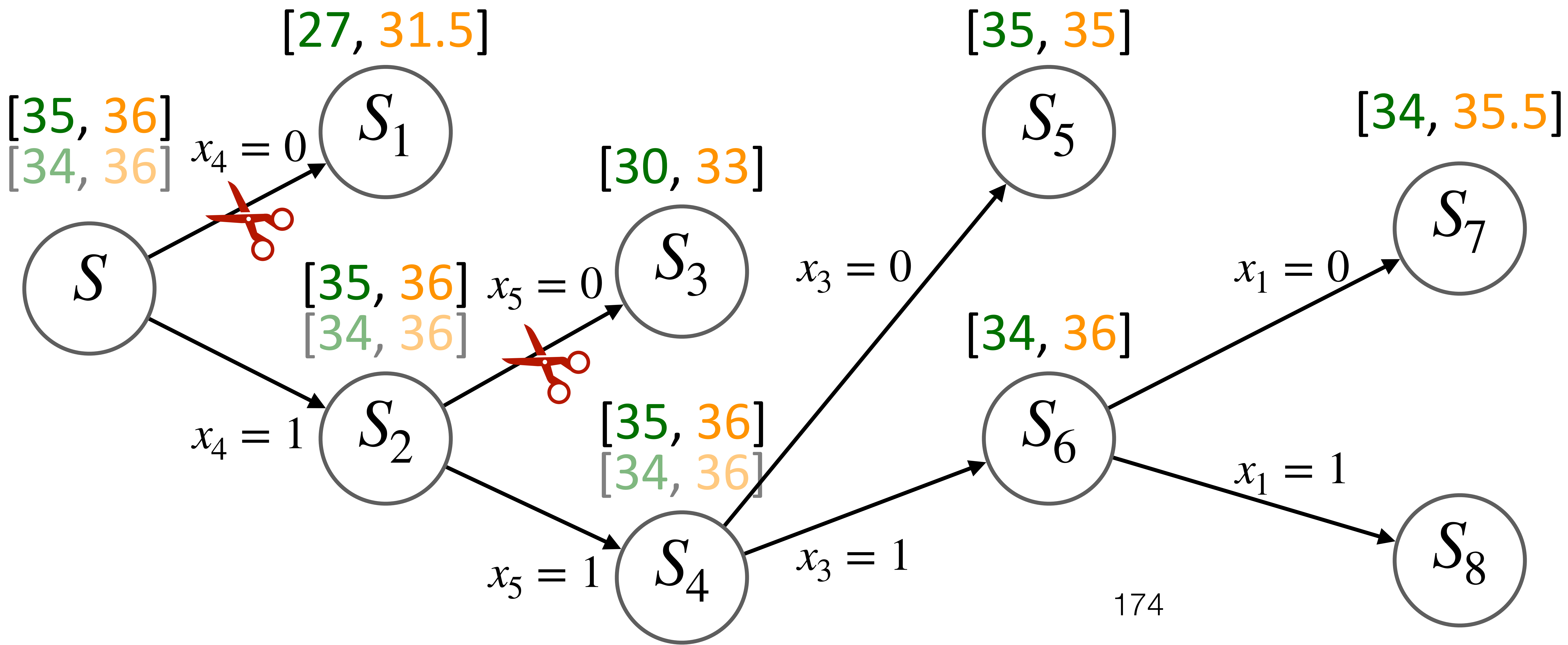
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 $\frac{1}{8}$ 0	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



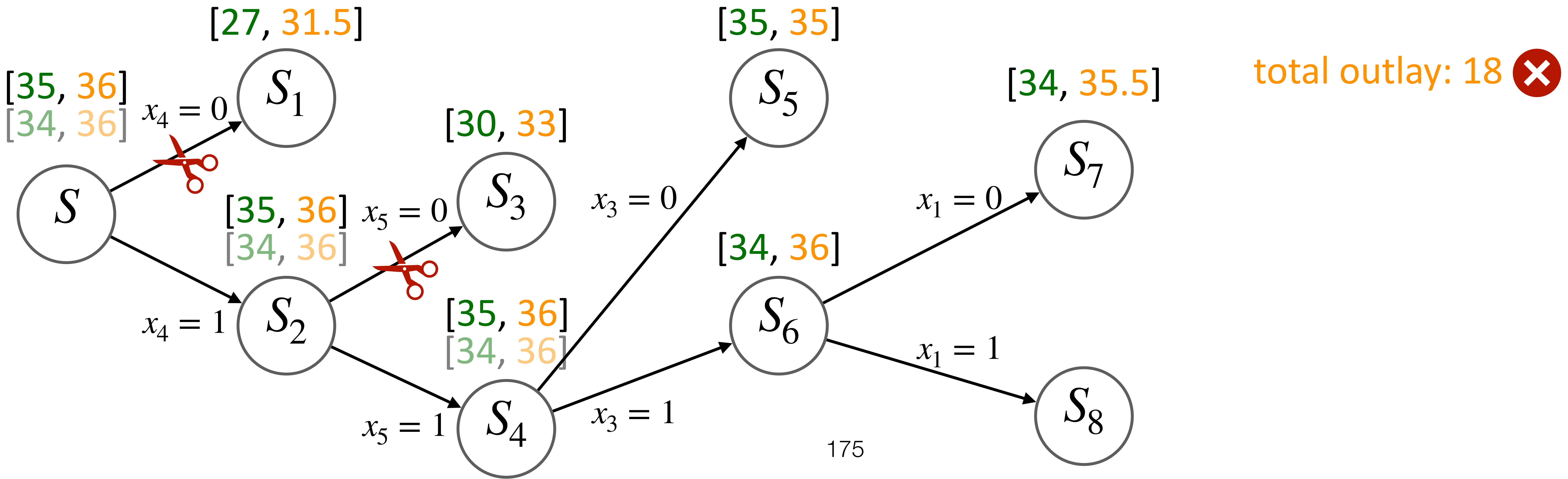
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **1** **1**	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



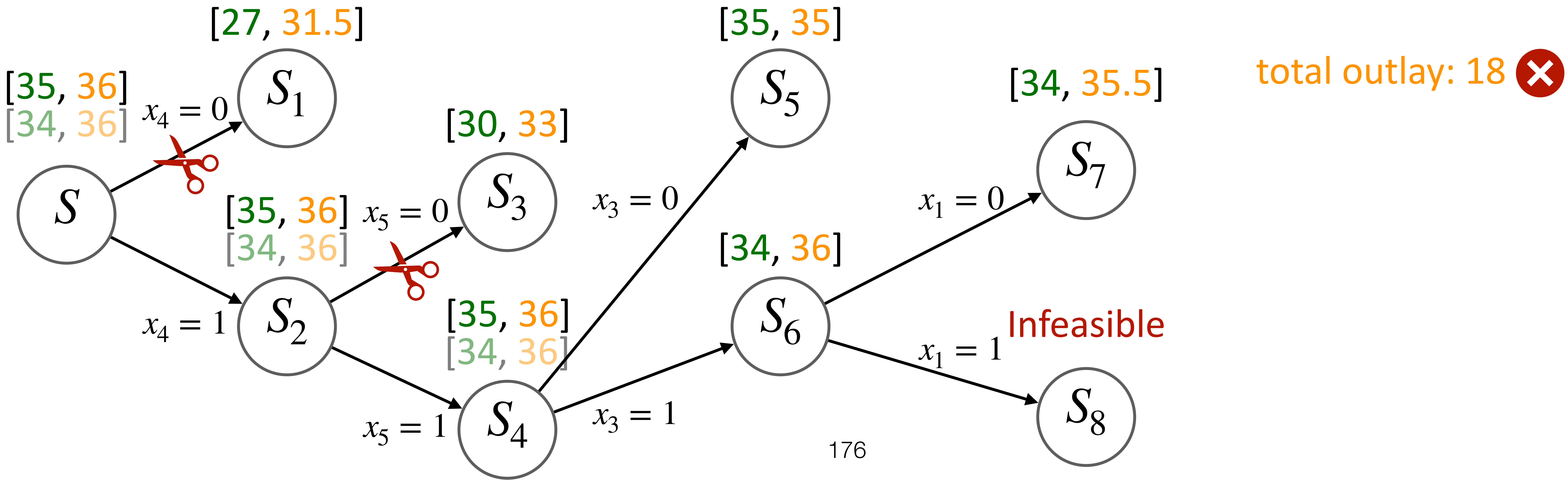
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **1** **1**	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



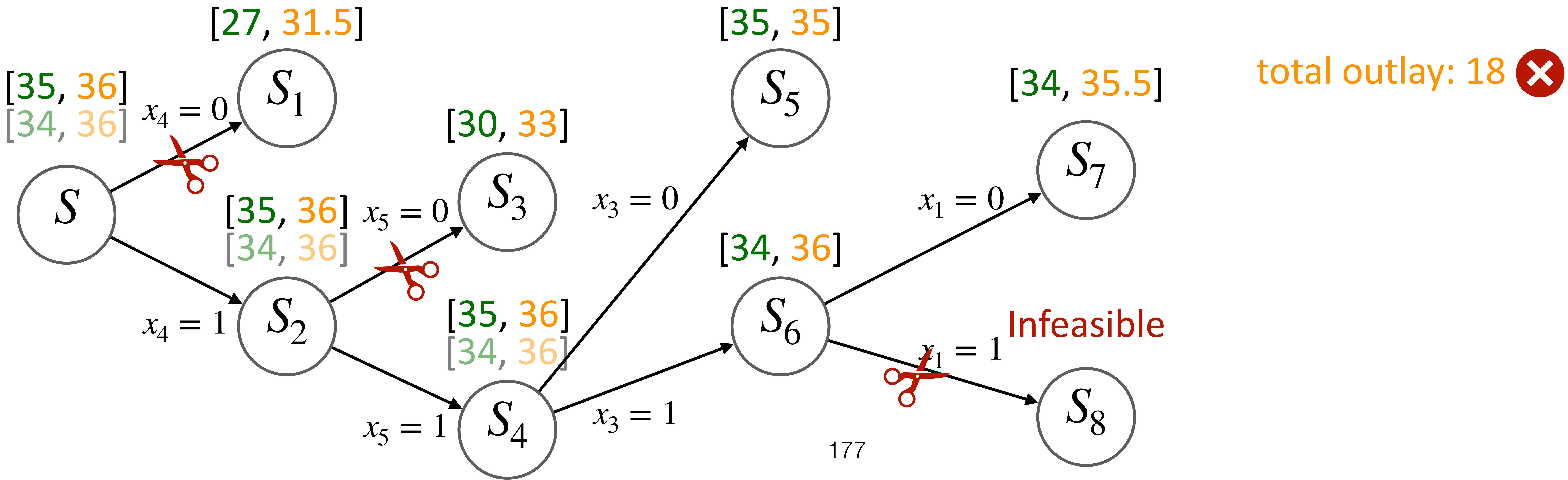
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **1** **1**	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



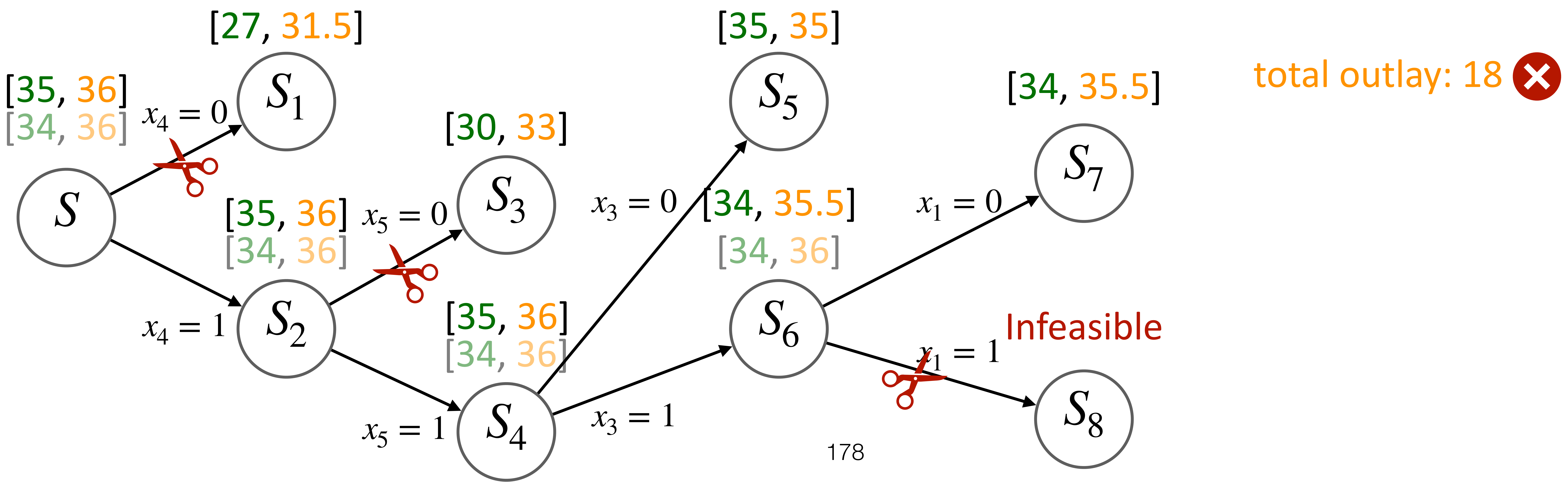
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **1** **1**	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



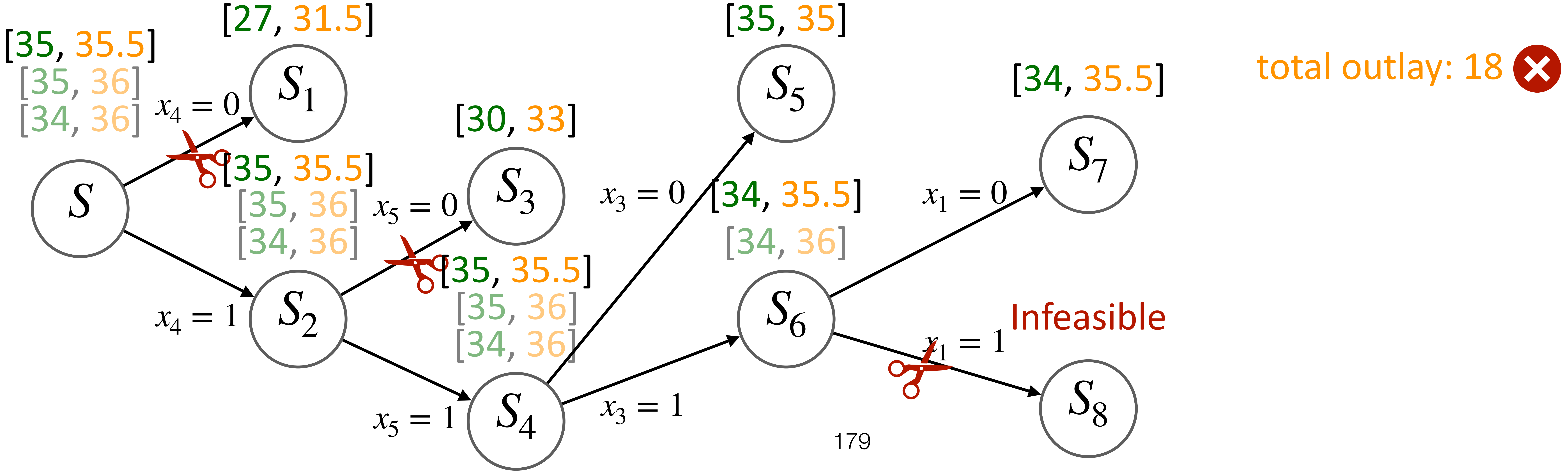
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **1** **1**	8	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



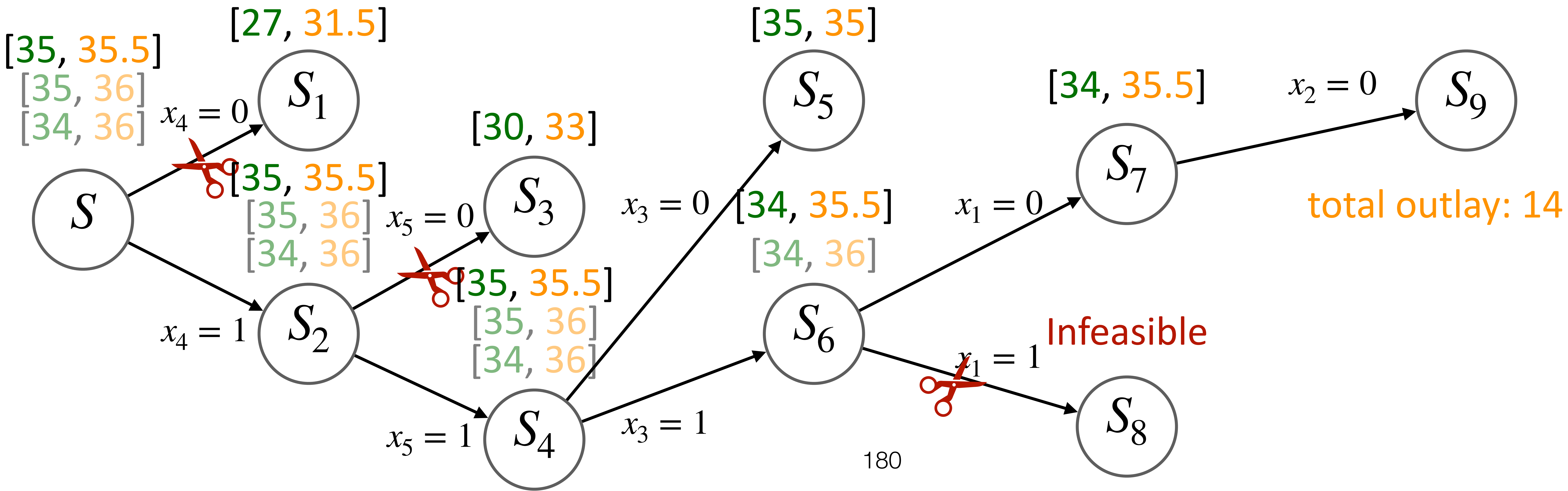
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 1 1	8	3 1 1	6 1 1	5 1 1
outlay/return	2	1.5	2.333	2.5	2.4



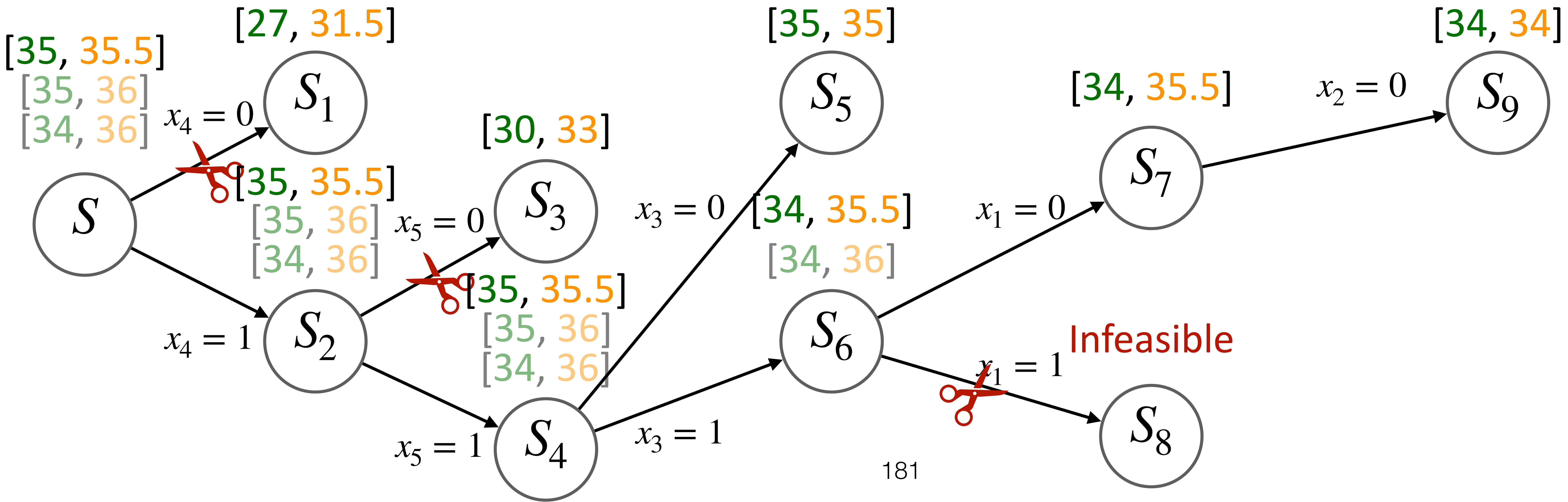
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **0** **0**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



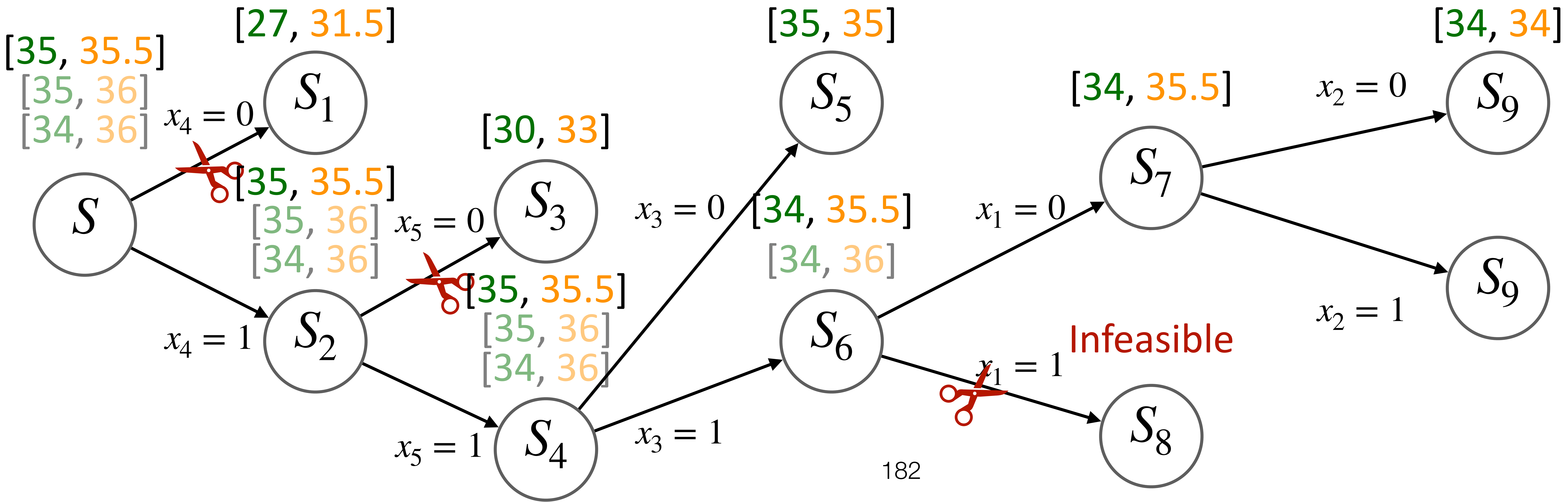
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **0** **0**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



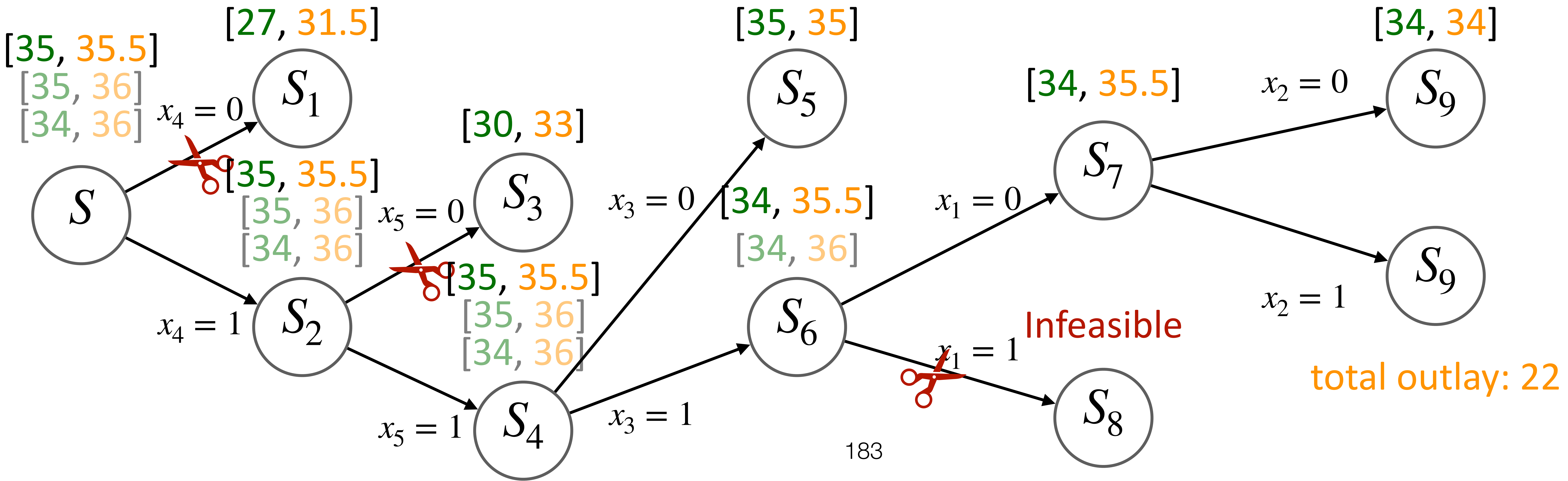
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0 0**	8 **1 1**	3 **1 1**	6 **1 1**	5 **1 1**
outlay/return	2	1.5	2.333	2.5	2.4



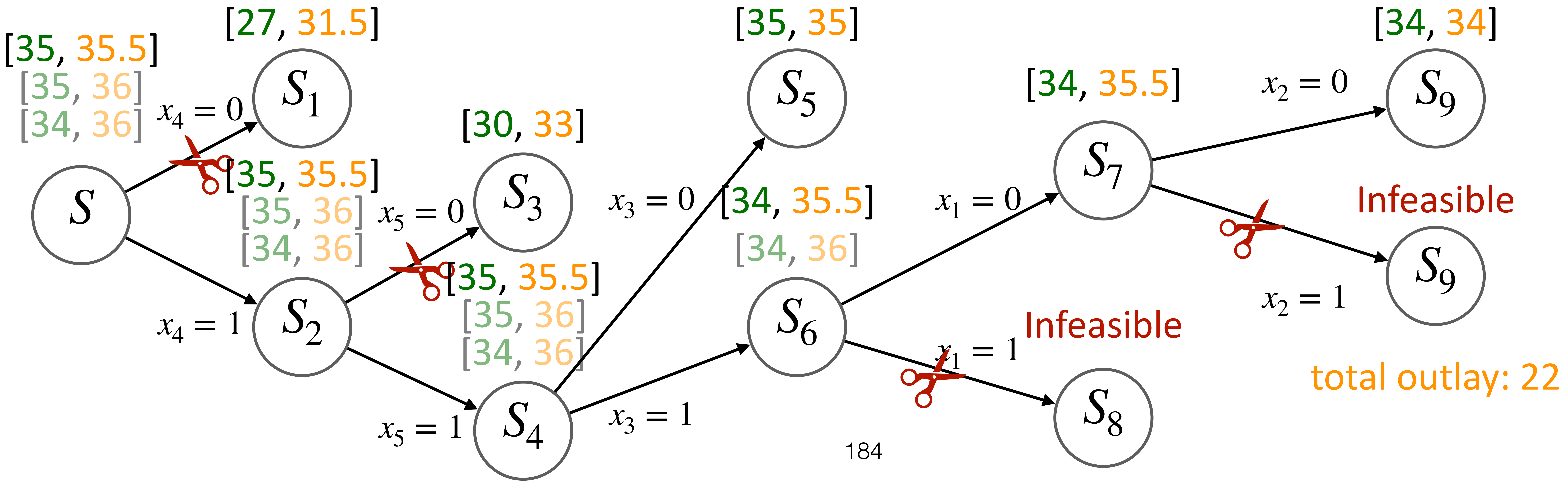
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **1** **1**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



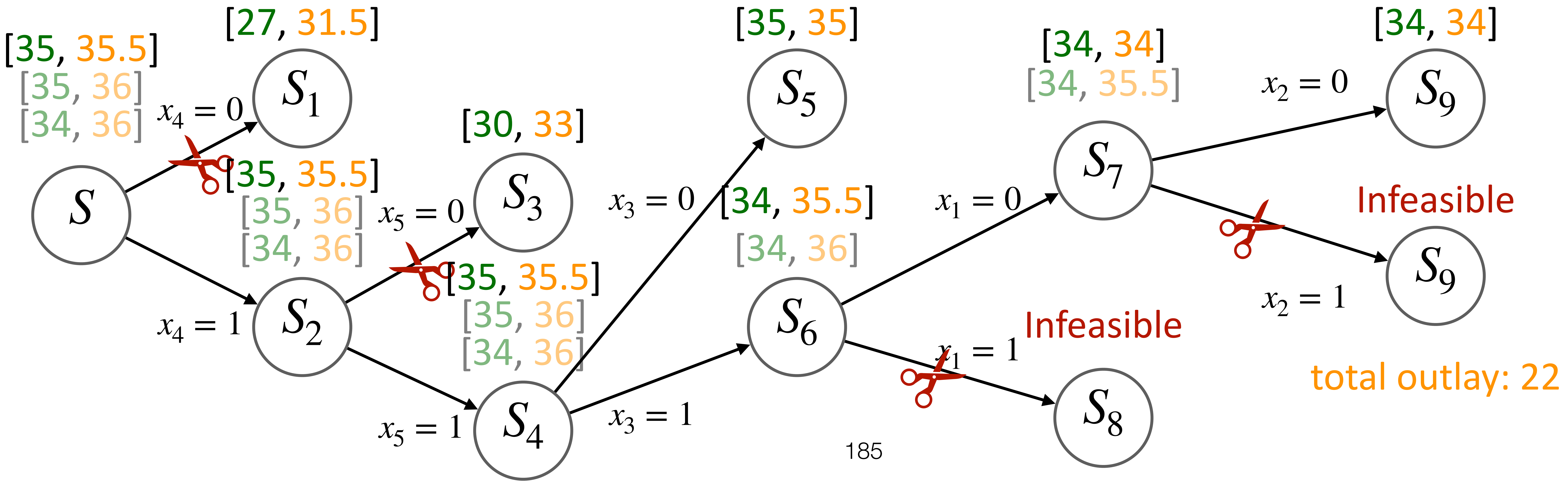
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **1** **1**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



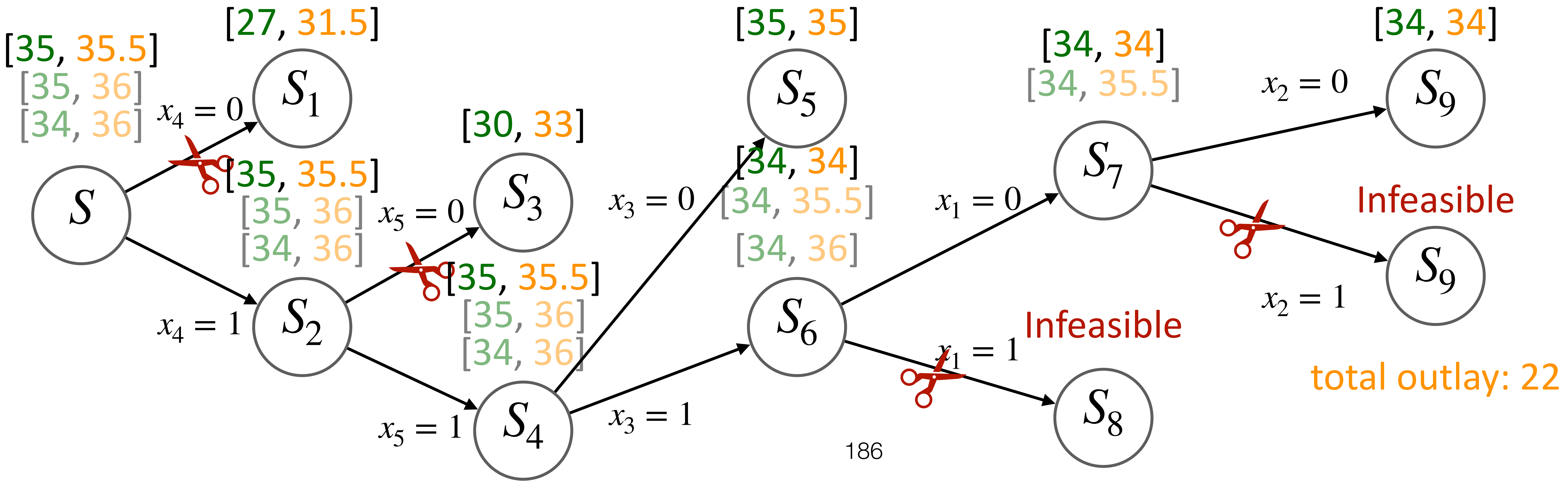
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **1** **1**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



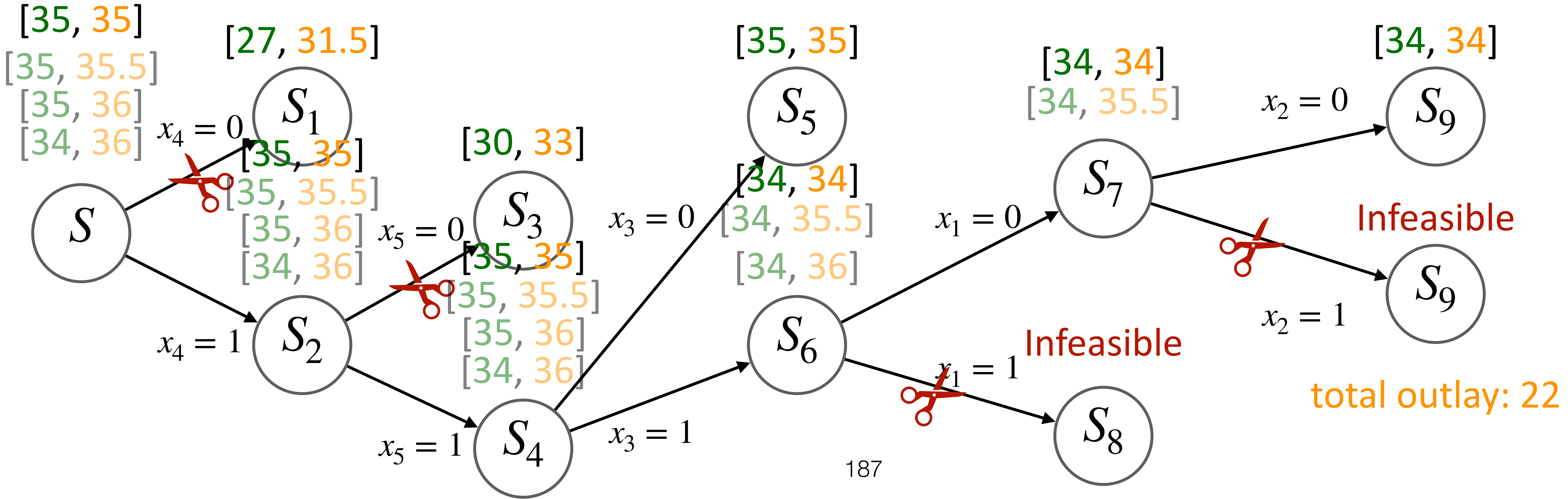
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **1** **1**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



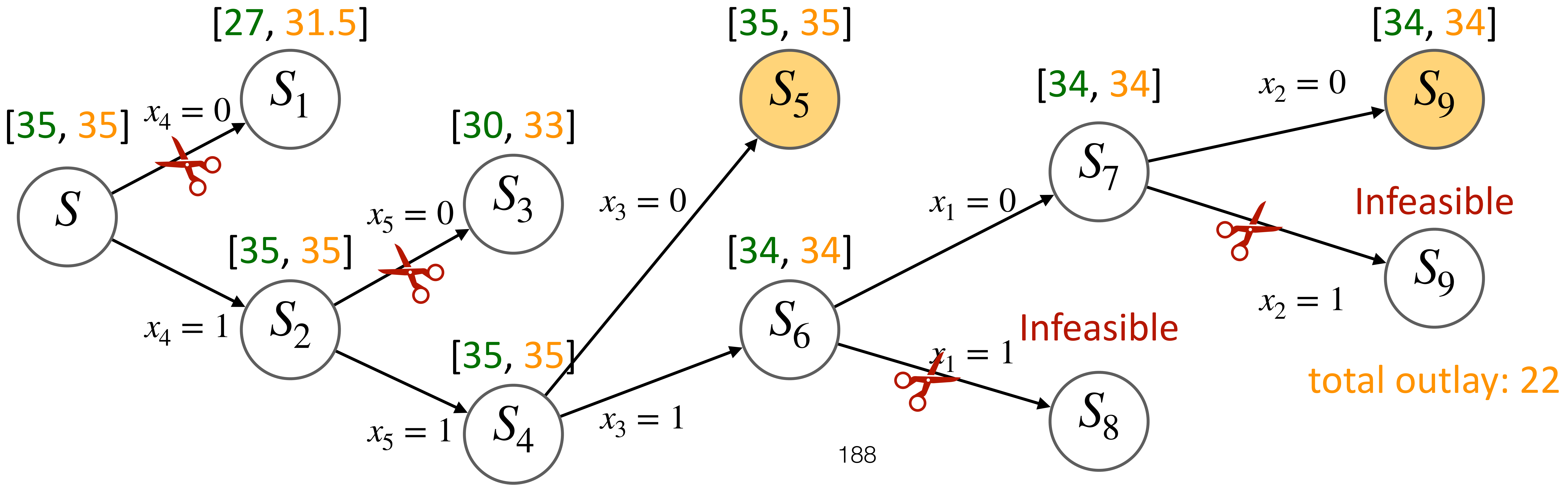
Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4 **0** **0**	8 **1** **1**	3 **1** **1**	6 **1** **1**	5 **1** **1**
outlay/return	2	1.5	2.333	2.5	2.4



Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6	5
outlay/return	2	1.5	2.333	2.5	2.4



Use Branch-and-Bound to solve Knapsack

Item	1	2	3	4	5
return	8	12	7	15	12
outlay **15**	4	8	3	6	5
outlay/return	2	1.5	2.333	2.5	2.4