14. State-Space Search

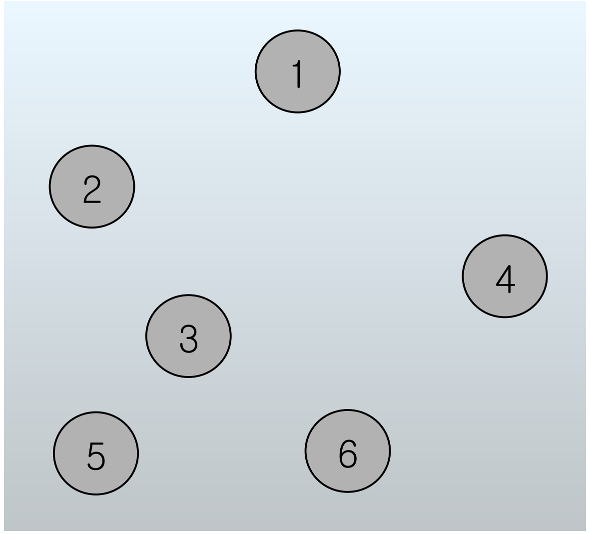
Goals

- Understand state-space tree.
- Learn backtracking.
- Learn branch-and-bound.
- Learn A* algorithm.

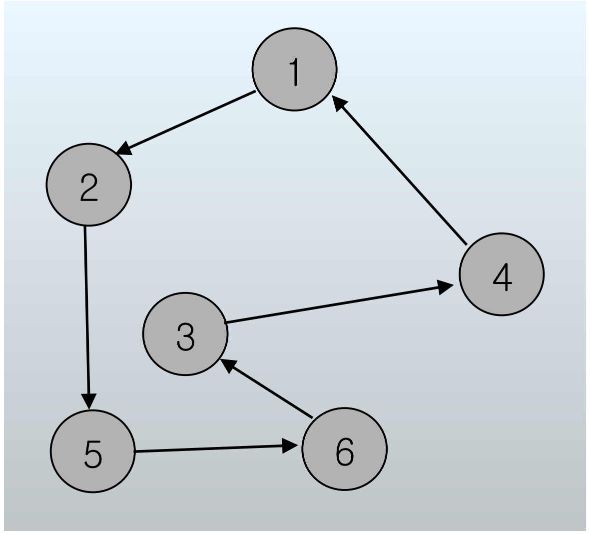
State-Space Tree

- State space: set of states that are generated in problem solving process
- State space tree: tree which represents states of problem solving process in a systematic way
  - Solution space: set of candidate solutions for the problem (leaves in a state space tree). Some nodes in the solution space are answers to the problem
  - Partial solution: internal node
- Search techniques for state space
  - Backtracking
  - Branch-and-bound
  - A^* algorithm

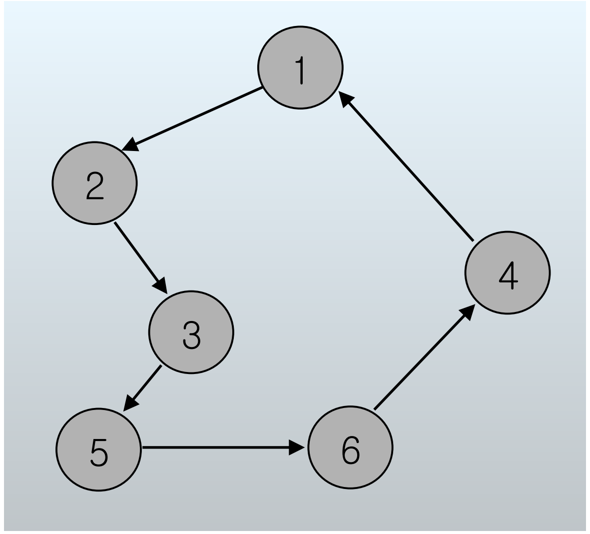
TSP



(a) instance of TSP



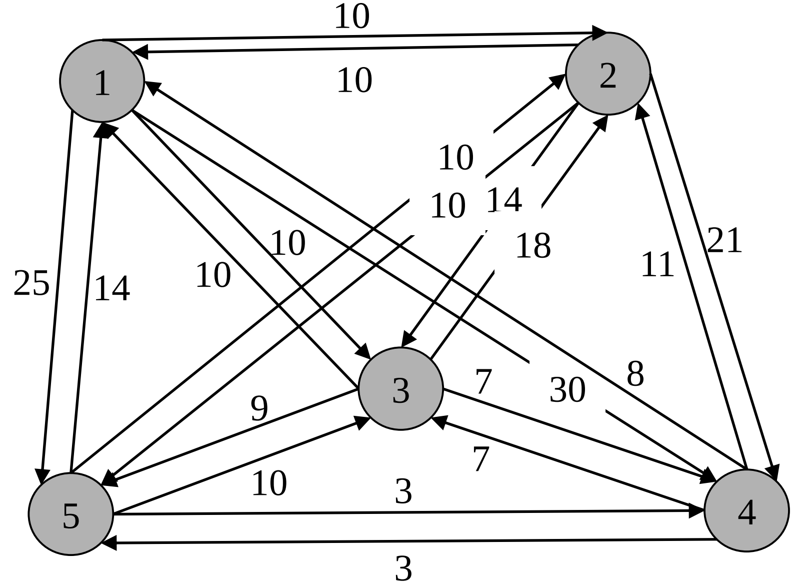
(b) a solution



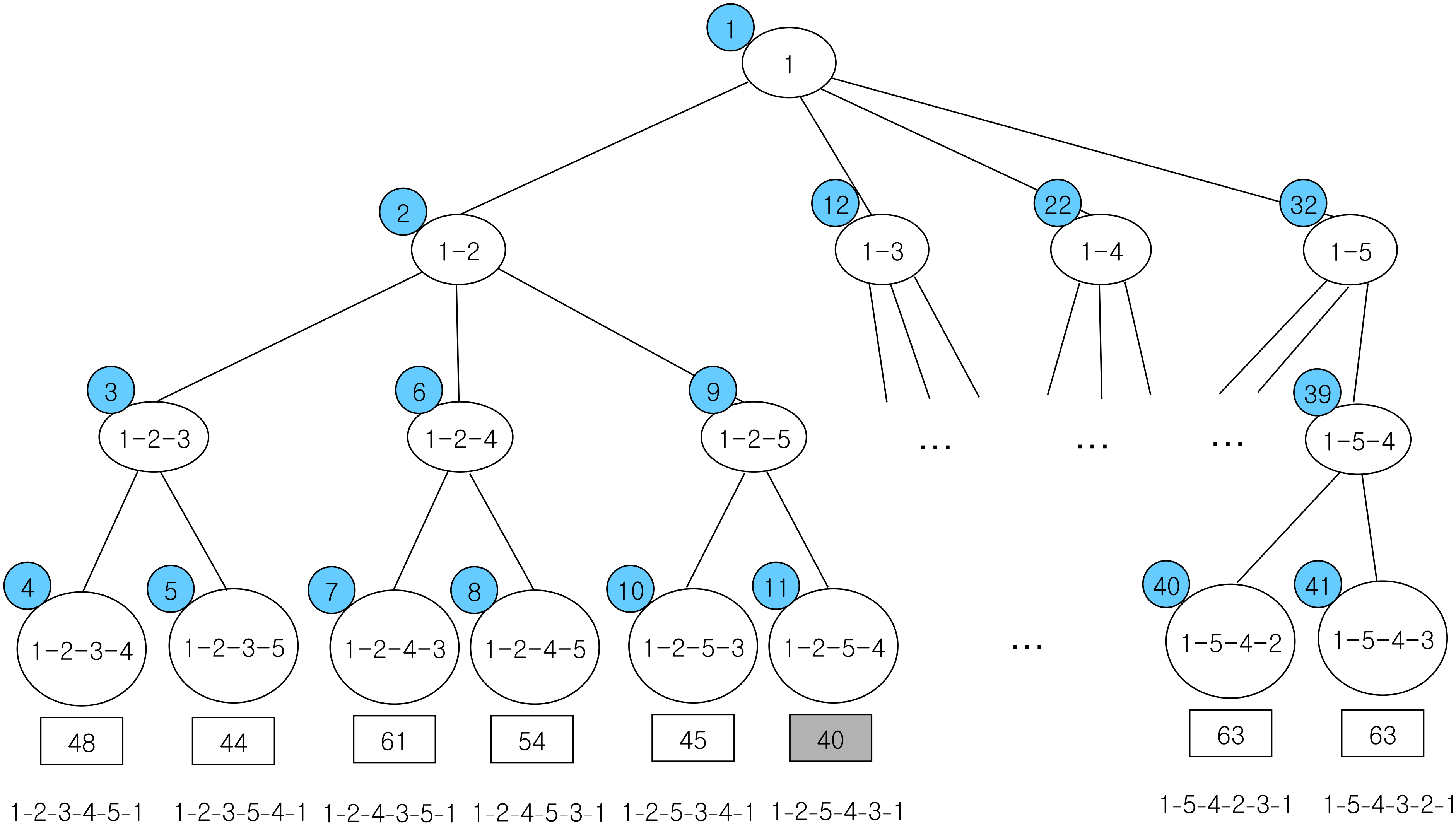
(c) optimal solution

TSP and Adjacency Matrix

	1	2	3	4	5
1	0	10	10	30	25
2	10	0	14	21	10
3	10	18	0	7	9
4	8	11	7	0	3
5	14	10	10	3	0



Lexicographic order search of state space



Backtracking

- Current state $(x_1, x_2, \dots, x_{i-1})$: a path from root to a node in a state space tree
- $T(x_1, \dots, x_{i-1})$: set of all possible values for x_i
- $f(x_1, \dots, x_i) =$ false only if the path cannot be extended to reach an answer node

Backtrack(n) {
    $i \leftarrow 1$;
    while ($i > 0$) {
        if there is an untried x_i such that
        $x_i \in T(x_1, \dots, x_{i-1})$ and $f(x_1, \dots, x_i)$ **then**
            if $(x_1, \dots, x_i)$ is an answer node **then** print $(x_1, \dots, x_i)$;
            else $i \leftarrow i+1$;
        else $i \leftarrow i - 1$;
    }
}

Backtracking

- Recursive version of backtracking
- Initial call: BacktrackR(1)
- Current state $(x_1, x_2, \ldots, x_{i-1})$

BacktrackR(i) {
    for each x_i such that $x_i \in T(x_1, \ldots, x_{i-1})$ and $f(x_1, \ldots, x_i)$ {
        if $(x_1, \ldots, x_i)$ is an answer node **then** print $(x_1, \ldots, x_i)$;
        else BacktrackR(i+1);
    }
}

8-Queens Problem

- Place 8 queens on an 8×8 chessboard so that no two can attack (i.e., no two are on the same row, column, or diagonal)

			Q				
					Q		
							Q
	Q						
						Q	
Q							
		Q					
				Q			

8-Queens Problem

- Queen i is placed on row i.
- All answers are represented by $(x_1, \dots, x_8)$, where x_i is the column on which queen i is placed.
- x_i increases from 1 to n+1.
- Current state $(x_1, x_2, \dots, x_{i-1})$

Queens(n) {
    $i \leftarrow 1$;
    while ($i > 0$) {
        $x_i \leftarrow$ next column on which queen i can be placed
        if $x_i \leq n$ **then**
            if $i = n$ **then** print $(x_1, \dots, x_i)$;
            else $i \leftarrow i+1$;
        else $i \leftarrow i - 1$;
    }
}

8-Queens Problem

- Current state: $(x_1 = 2, x_2 = 4, x_3 = 1)$
- x_4 can be 3, 7, 8.
- Next current state: $(x_1 = 2, x_2 = 4, x_3 = 1, x_4 = 3)$
- x_5 can be 5, 8.

	1						
			2				
3							
		X				X	X

	1						
			2				
3							
		4					
				X			X

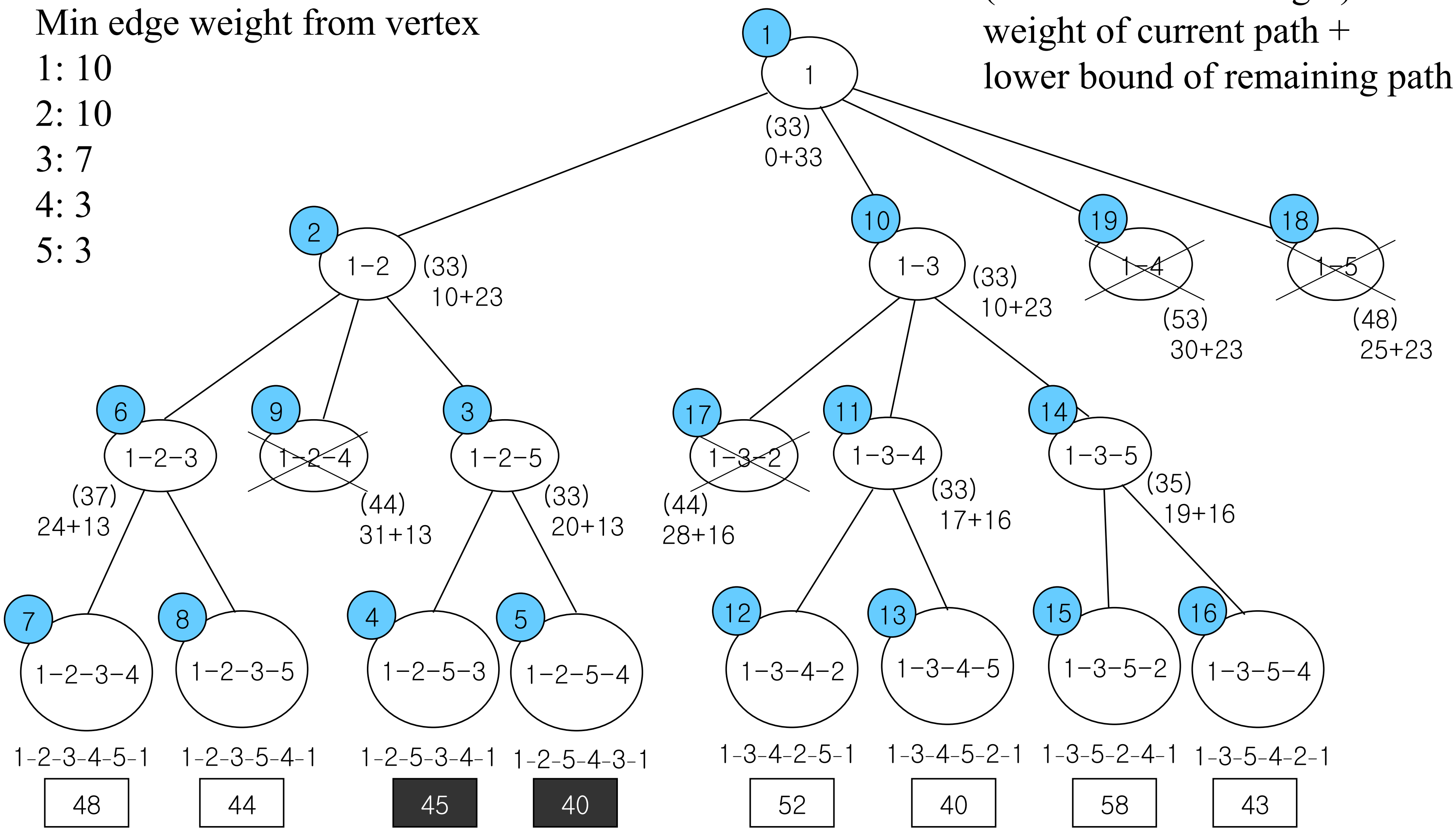
Branch-and-Bound

- Combination of ‘branch’ and ‘bound’
  - A cost function exists
  - A minimum cost answer node is to be found
- Comparison with backtracking
  - common
    - Require a method to list states
  - different
    - Backtracking – backtrack when there is no further way to go
    - Branch-and-bound – don’t branch if it is guaranteed that there is no optimal solution in that branch

State-Space Tree of Branch-and-Bound for TSP

Min edge weight from vertex
1: 10
2: 10
3: 7
4: 3
5: 3

(lower bound of weight)
weight of current path +
lower bound of remaining path



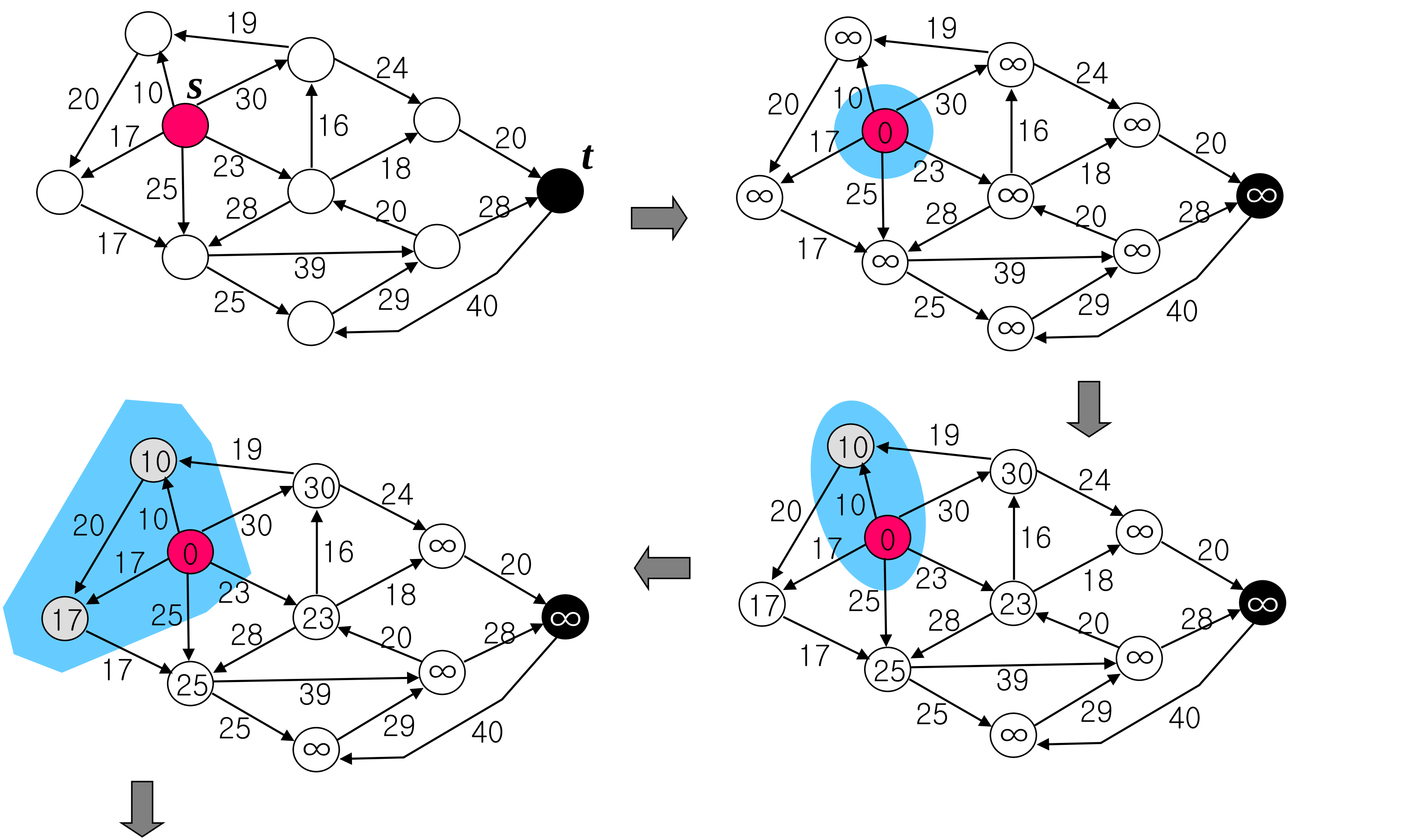
A^* Algorithm

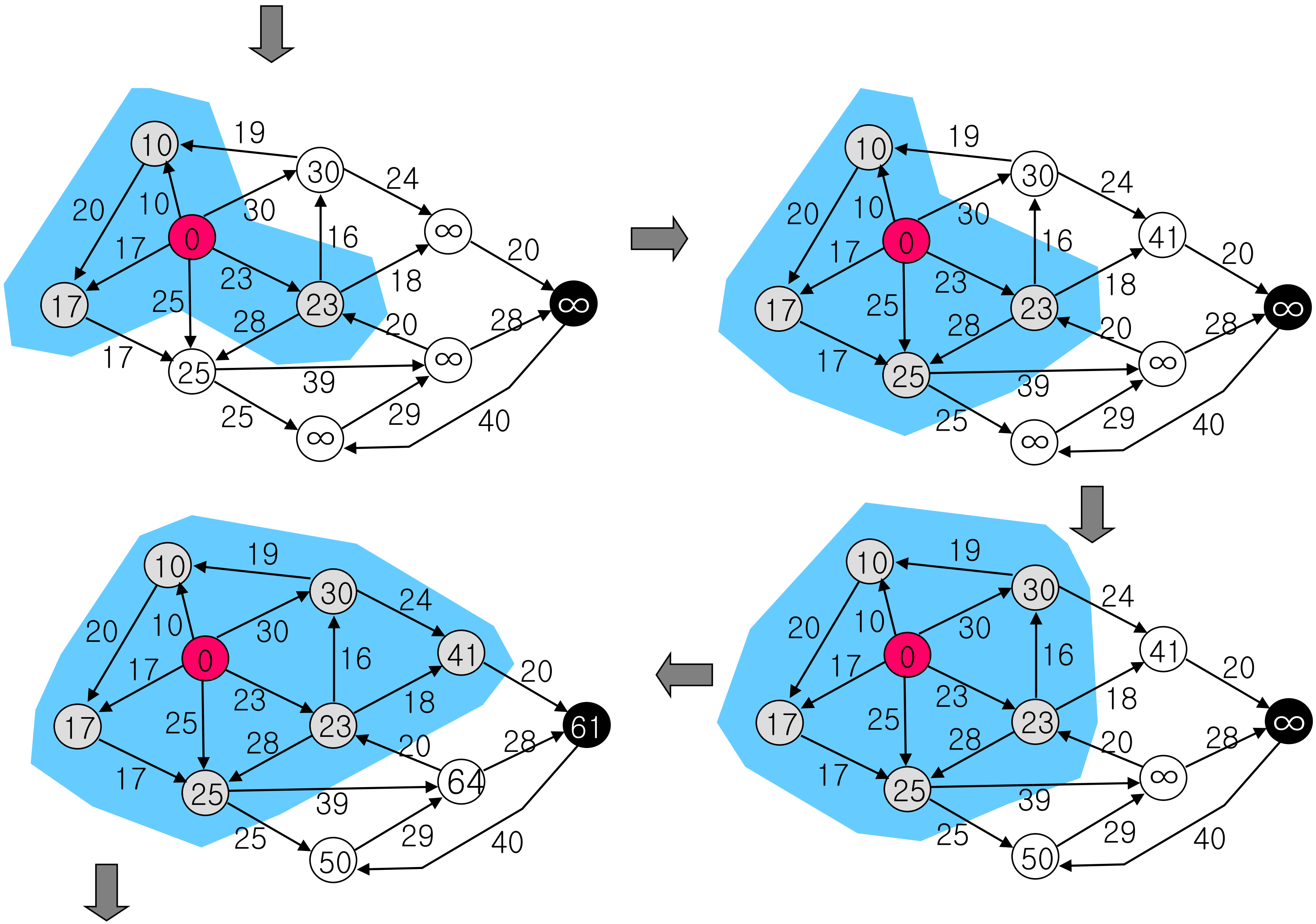
- Find the shortest path from a source to a destination
- Can be applied to NP-hard and P problems
- cf. Dijkstra algorithm
  - Single source
  - All destinations

A* Algorithm

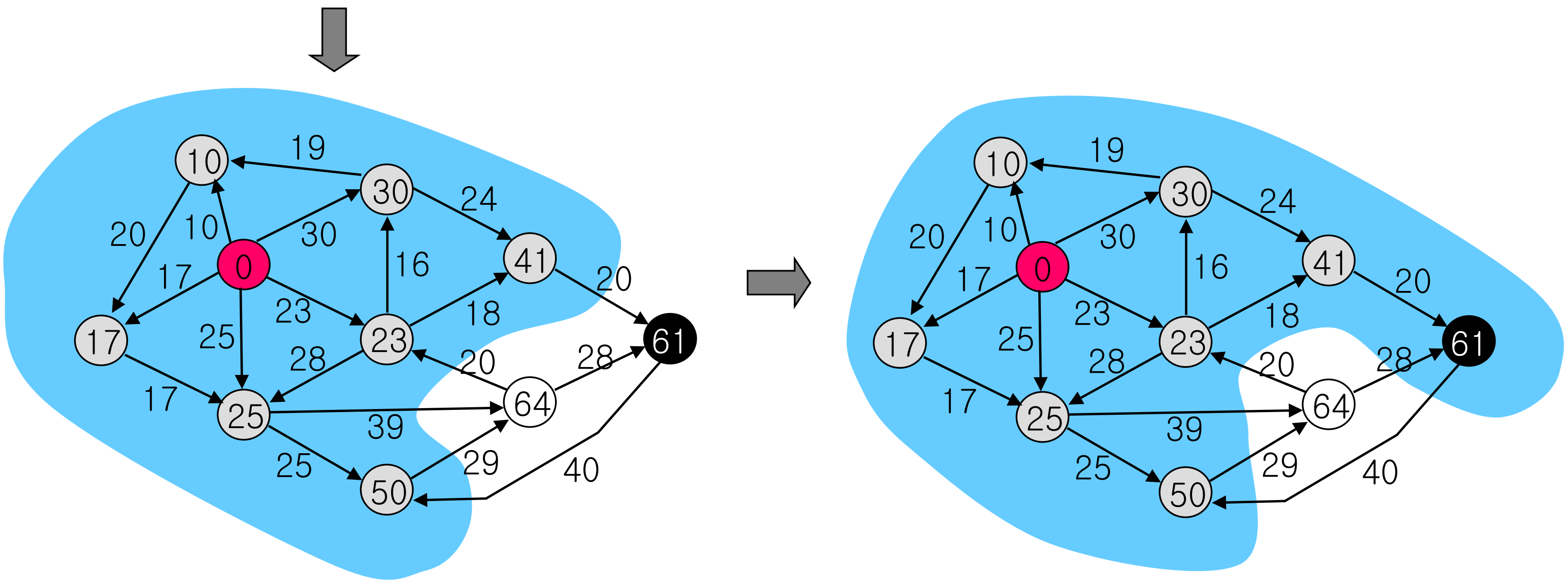
- Best-first search
  - Each vertex x has g(x): cost (shortest path weight) from source to x
  - Each vertex x has h(x): estimate of cost from x to destination. Properties of h(x):
    - h(x) must be less than or equal to actual cost from x to destination
    - For all x, y, $h(x) \le w(x,y) + h(y)$
  - A* always selects a vertex x that minimizes $g(x) + h(x)$

Dijkstra Algorithm

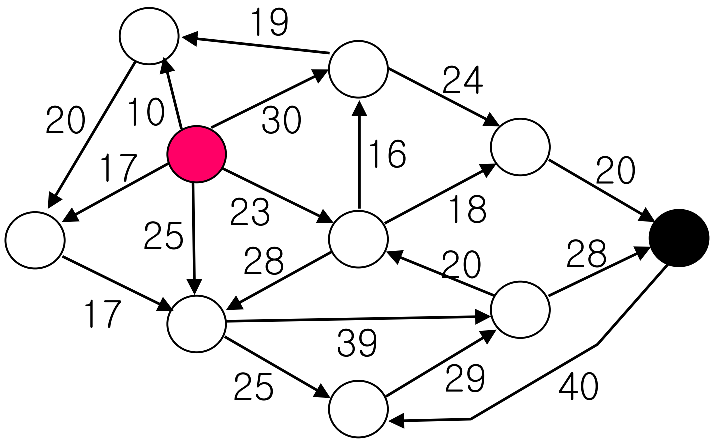




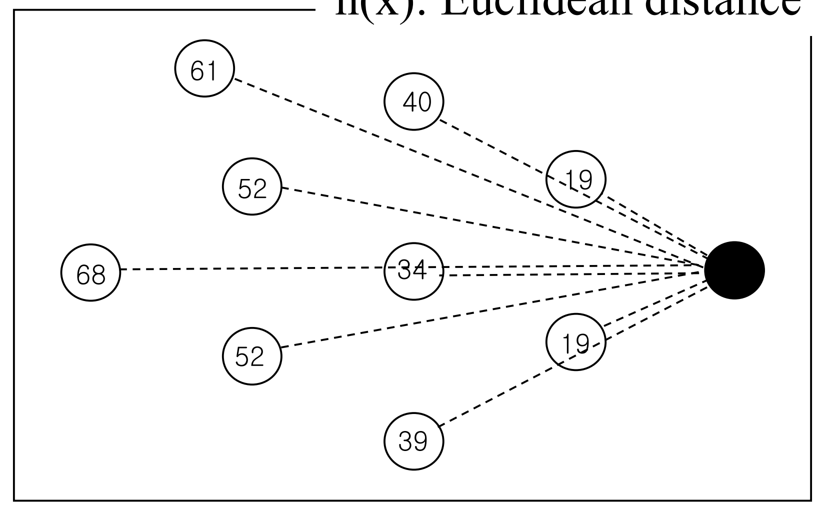
10 19 30 24 ∞ 20 10 30 0 16 20 17 23 23 18 ∞ 17 25 28 20 28 ∞ 17 25 39 25 ∞ 29 40
10 19 30 24 41 20 10 30 0 16 20 17 23 23 18 ∞ 17 25 28 20 28 ∞ 17 25 39 25 ∞ 29 40
10 19 30 24 41 20 10 30 0 16 20 17 23 23 18 61 17 25 28 20 28 64 17 25 39 25 50 29 40
10 19 30 24 41 20 10 30 0 16 20 17 23 23 18 ∞ 17 25 28 20 28 ∞ 17 25 39 25 50 29 40

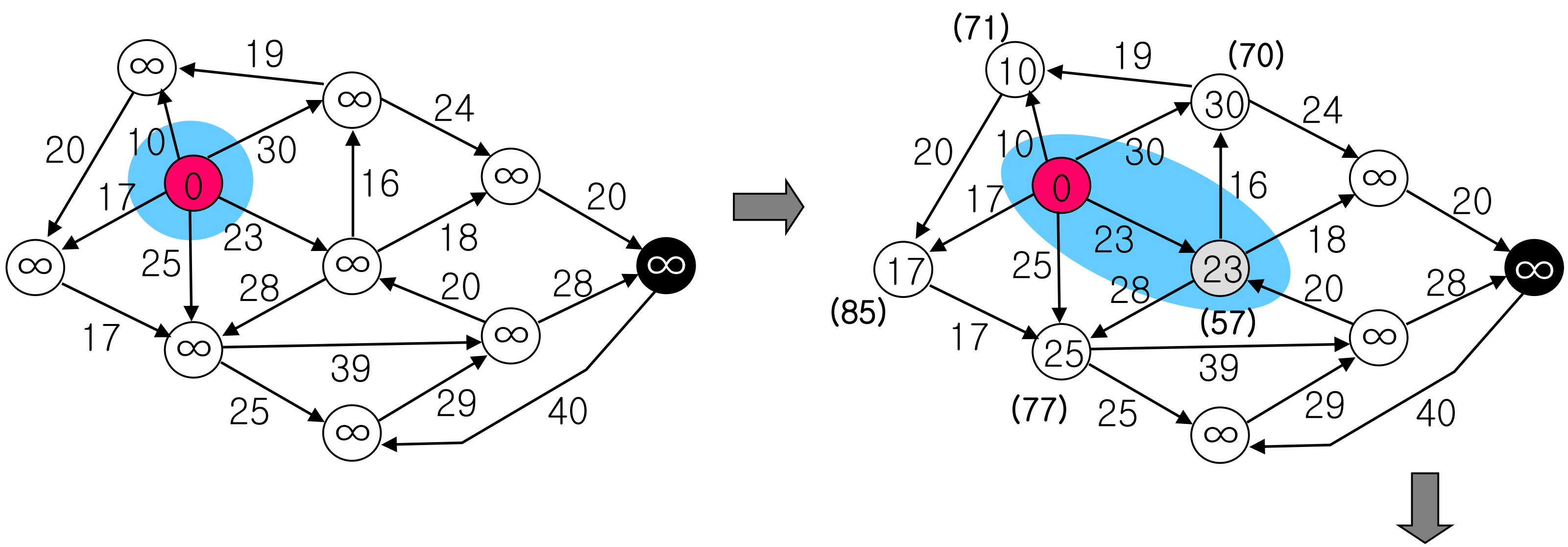

10
19
30
24
20
10
0
30
41
16
17
20
61
17
23
18
23
25
28
20
28
25
64
17
39
25
29
40
50
10
19
30
24
20
10
0
30
41
16
17
20
61
17
23
18
23
25
28
20
28
25
64
17
39
25
29
40
50

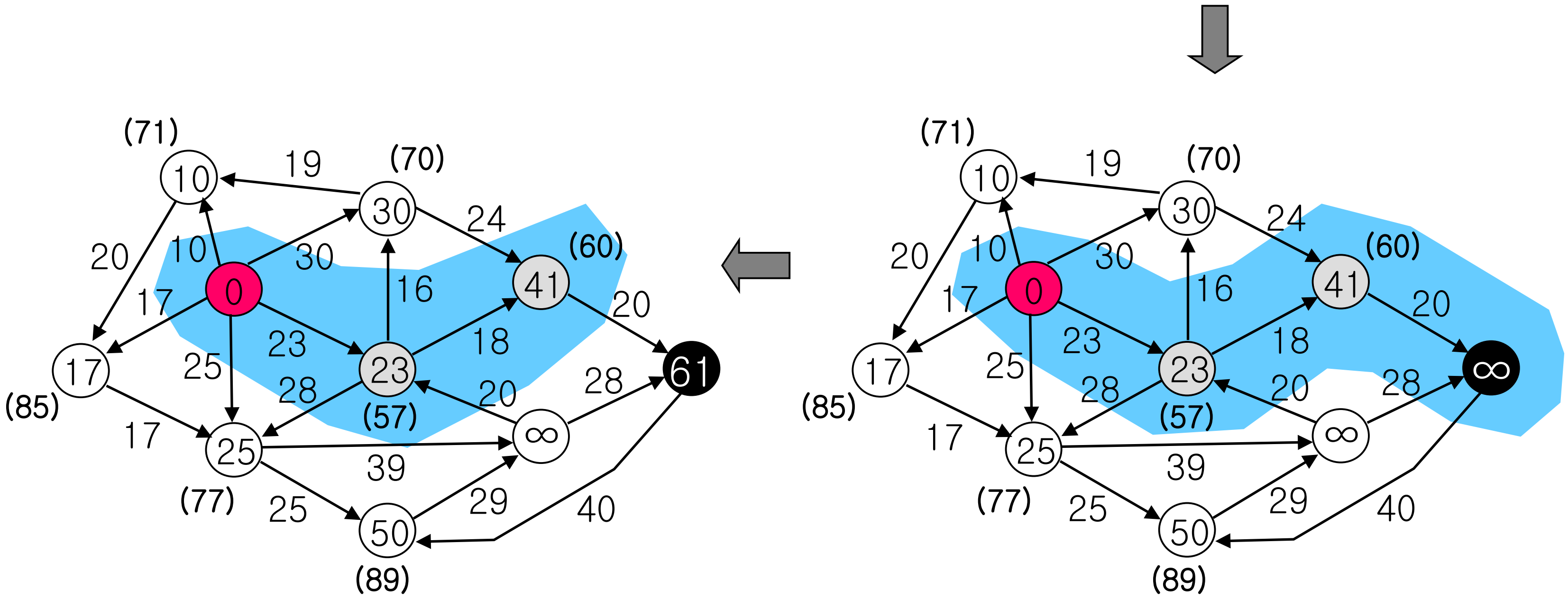
A* Algorithm



h(x): Euclidean distance



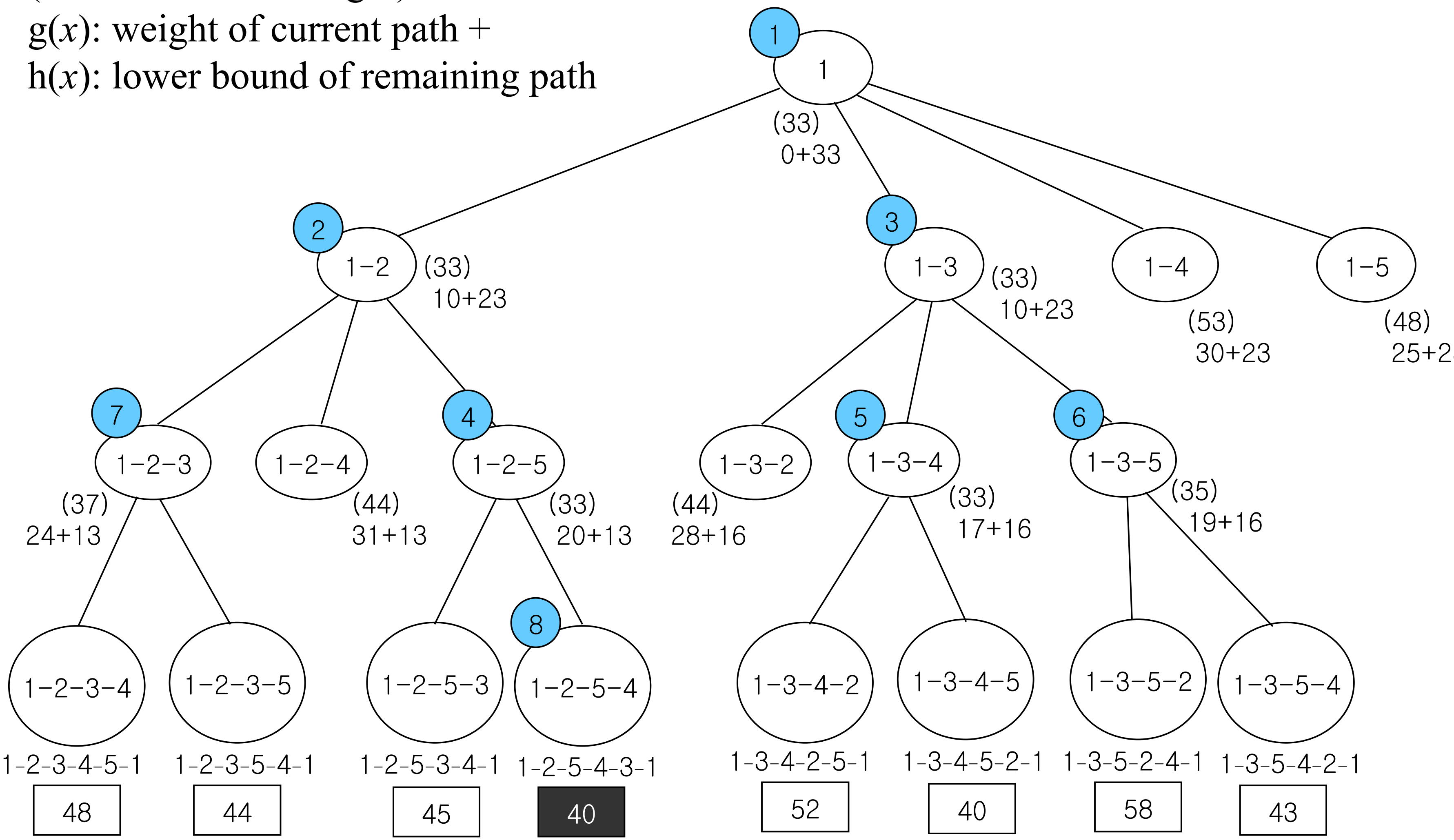




✓ A* is faster than Dijkstra by using h(x)

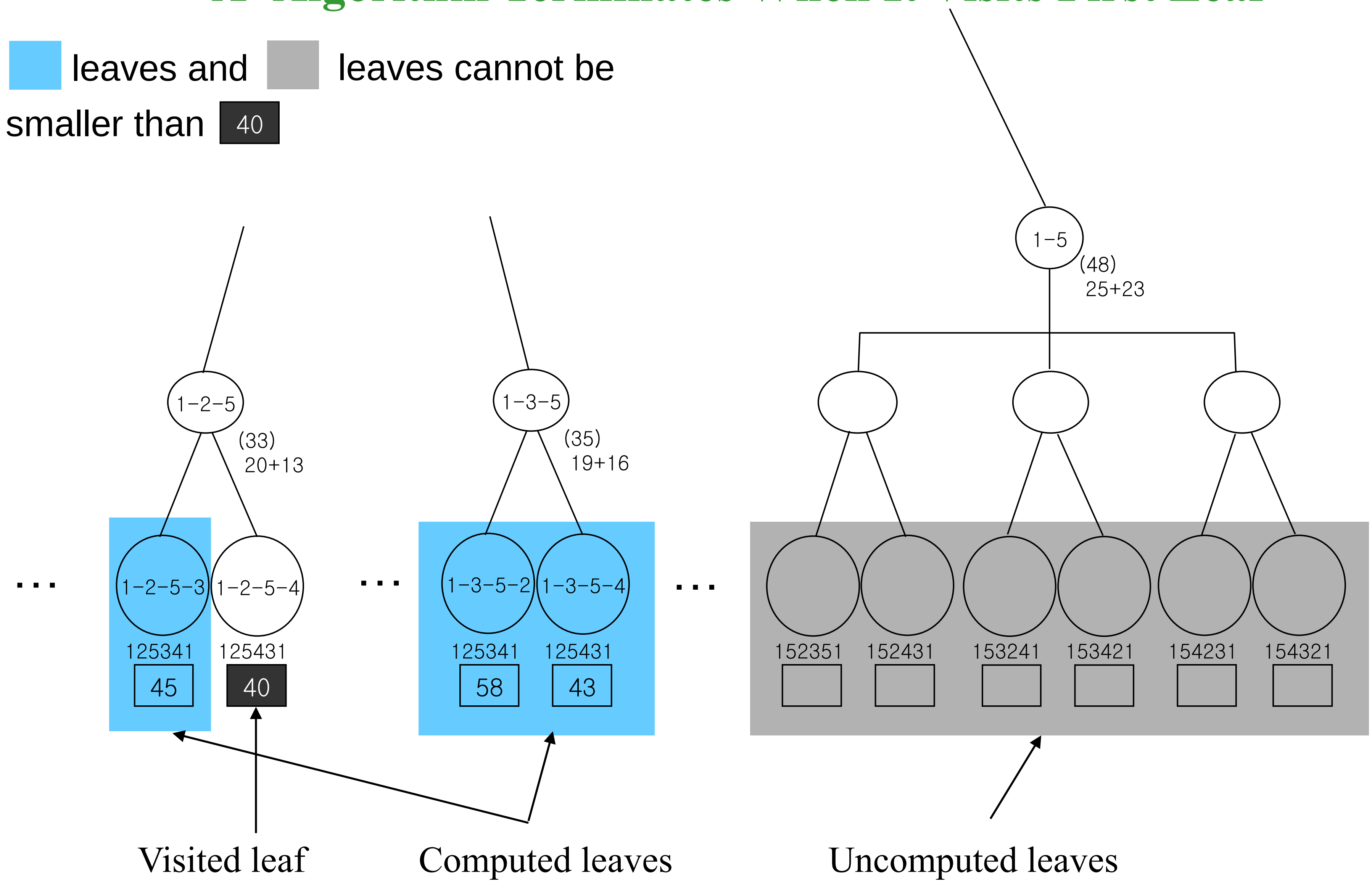
State-Space Tree of A* Algorithm for TSP

(lower bound of weight)
g(x): weight of current path +
h(x): lower bound of remaining path



A* Algorithm Terminates When It Visits First Leaf

leaves and leaves cannot be smaller than 40



Thank you