Large Neighborhood Beam Search for Domain-Independent Dynamic Programming

Ryo Kuroiwa and J. Christopher Beck

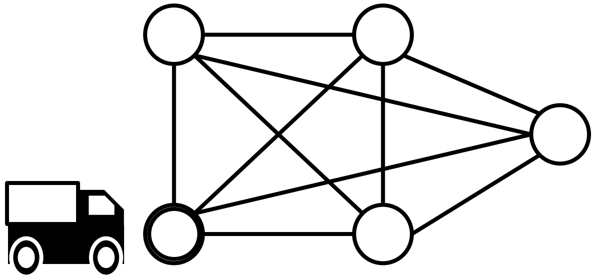
Toronto Intelligent Decision Engineering Laboratory (TIDEL)
Department of Mechanical and Industrial Engineering
University of Toronto

Background

Domain-Independent Dynamic Programming (DIDP)

CP-like **model & solve paradigm** based on dynamic programming (DP)

Combinatorial optimization problem



Model

State-based DP model

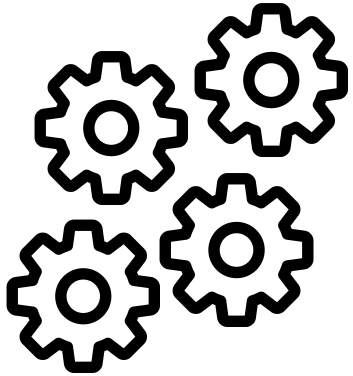
compute $V(\{1,2,3,4\},0)$

$$V(U,i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

$V(U,i) \geq h(U,i)$

Solve

DIDP solver



Current solvers use **state space search**

Kuroiwa and Beck ICAPS 2023a,b

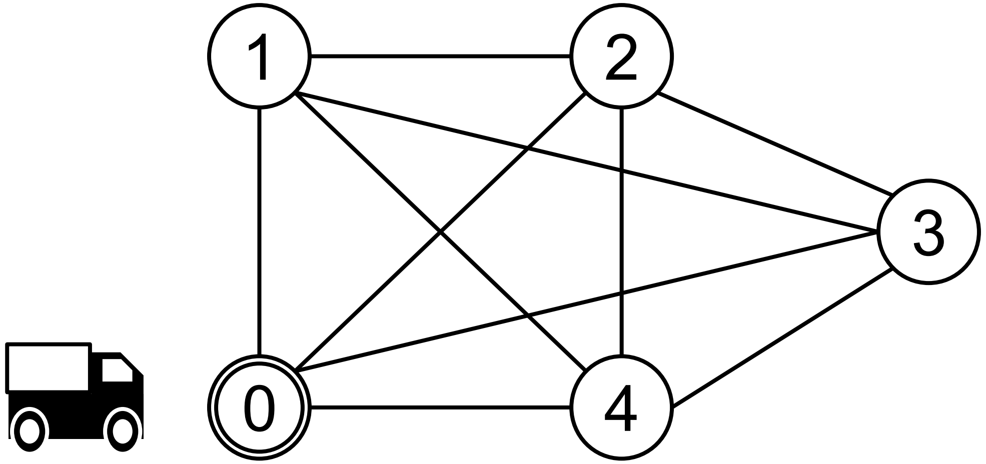
DIDP Sample Code

```python
import didppy as dp

model = dp.Model()
customer = model.add_object_type(number=4)
unvisited = model.add_set_var(object_type=customer, target=[1, 2, 3])
location = model.add_element_var(object_type=customer, target=0)
travel_time = model.add_int_table(
    [[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]]
)

for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=travel_time[location, j] + dp.IntExpr.state_cost(),
        effects=[(unvisited, unvisited.remove(j)), (location, j)],
        preconditions=[unvisited.contains(j)],
    )
    model.add_transition(visit)

return_to = dp.Transition(
    name="return",
    cost=travel_time[location, 0] + dp.IntExpr.state_cost(),
    effects=[(location, 0)],
    preconditions=[unvisited.is_empty(), location != 0],
)
model.add_transition(return_to)
model.add_base_case([unvisited.is_empty(), location == 0])

solver = dp.CAASDy(model)
solution = solver.search()
```

Visit https://didppy.rtfd.io

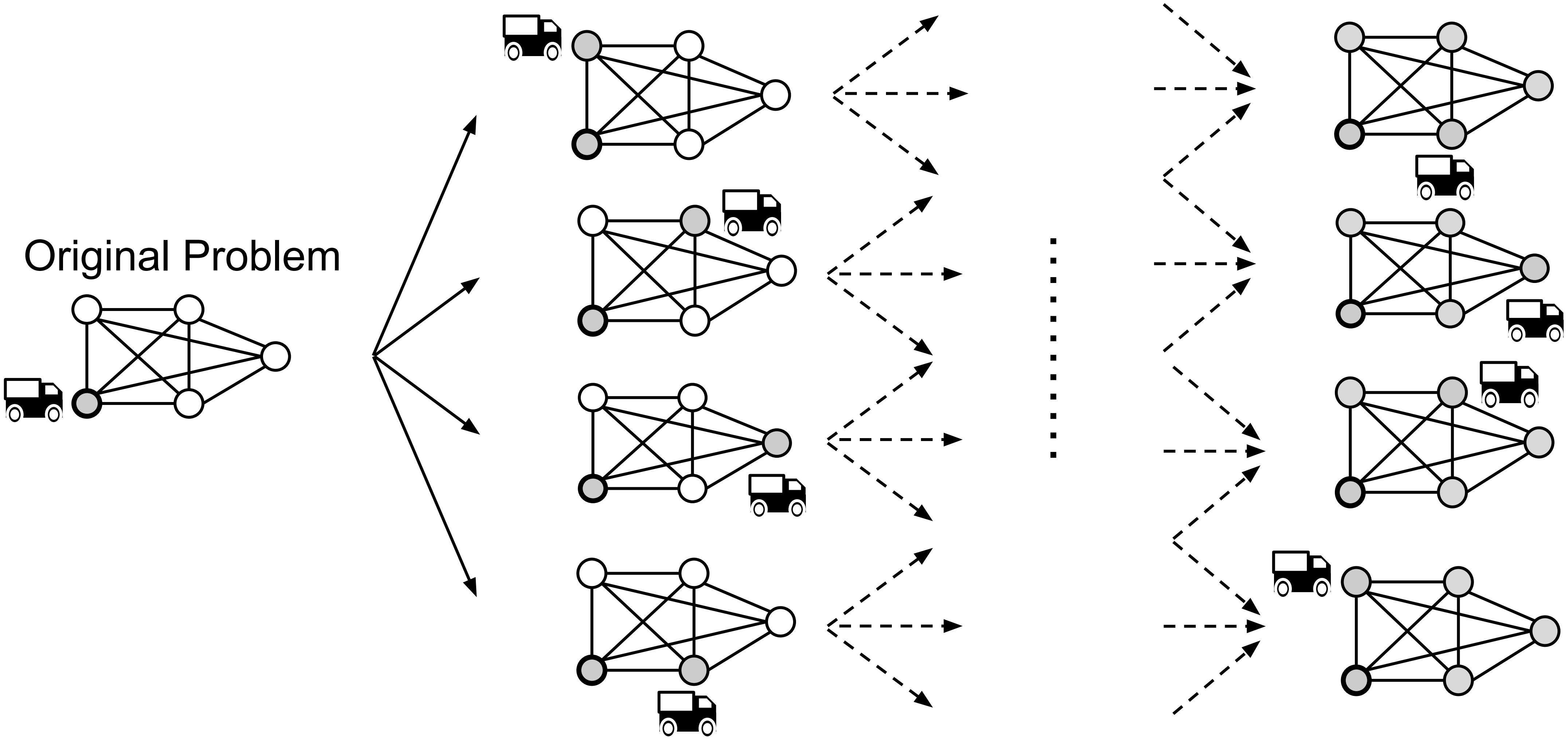
Example 1: a TSP-Like Routing Problem

Visit all nodes starting from customer 0 (no need to return) while minimizing the total travel cost (visiting j from i requires cost c_{ij})



DP Model for the Example Problem

Recursive decomposition into subproblems by visiting one customer



DP Model for the Example Problem

Recursively defined **value function** V maps a **state** (subproblem), defined by unvisited customers U and the current customer i, to the cost

$$\text{compute } V(\{1,2,3,4\},0)$$

Target state (original problem)

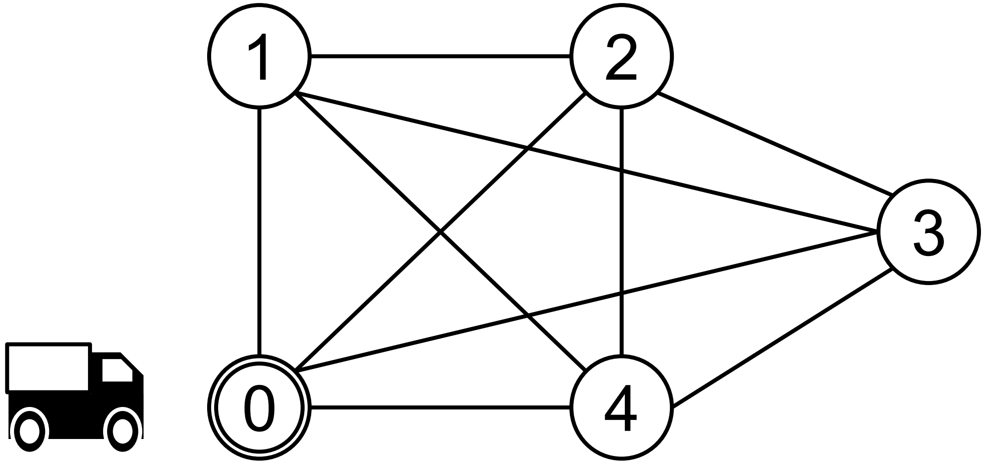
$$V(U,i) = \begin{cases} \min\limits_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

Transition (visiting one customer)

Base case (all customers are visited)

$$V(U,i) \geq h(U,i)$$

Dual bound function



DP Model for the Example Problem

Recursively defined **value function** V maps a **state** (subproblem), defined by unvisited customers U and the current customer i, to the cost

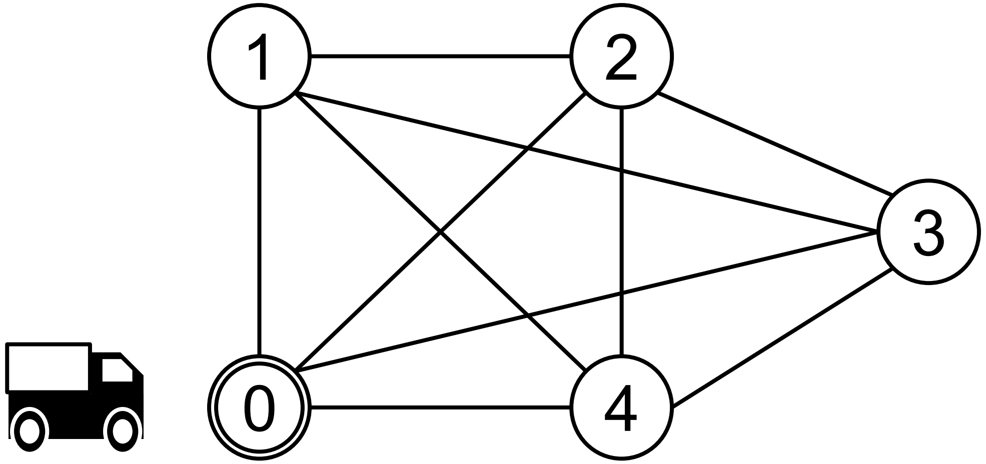
$\text{compute } V(\{1,2,3,4\},0)$ **Target state** (original problem)

$$V(U,i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

Transition (visiting one customer)

Base case (all customers are visited)

$V(U,i) \geq h(U,i)$ **Dual bound function**



DP Model for the Example Problem

Recursively defined **value function** V maps a **state** (subproblem), defined by unvisited customers U and the current customer i, to the cost

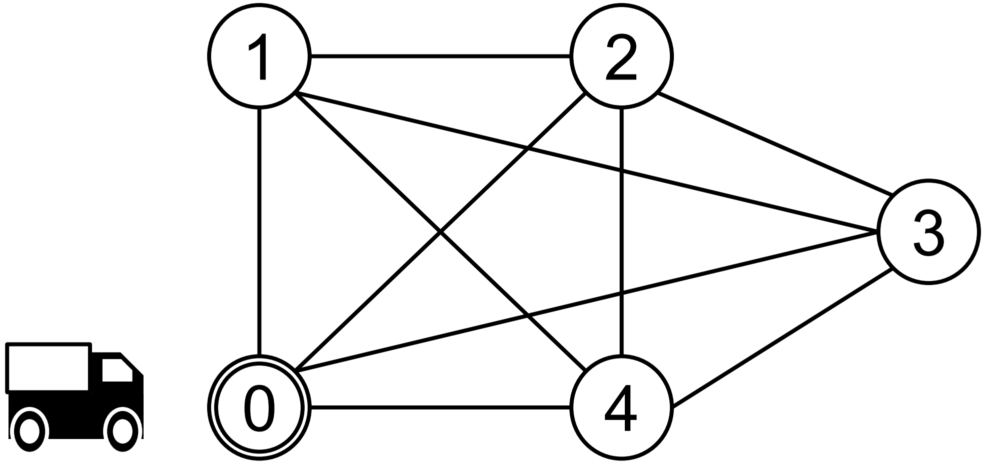
$\text{compute } V(\{1,2,3,4\},0)$ — **Target state** (original problem)

$$V(U,i) = \begin{cases} \min\limits_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

Transition (visiting one customer)

Base case (all customers are visited)

$V(U,i) \geq h(U,i)$ — **Dual bound function**



DP Model for the Example Problem

Recursively defined **value function** V maps a **state** (subproblem), defined by unvisited customers U and the current customer i, to the cost

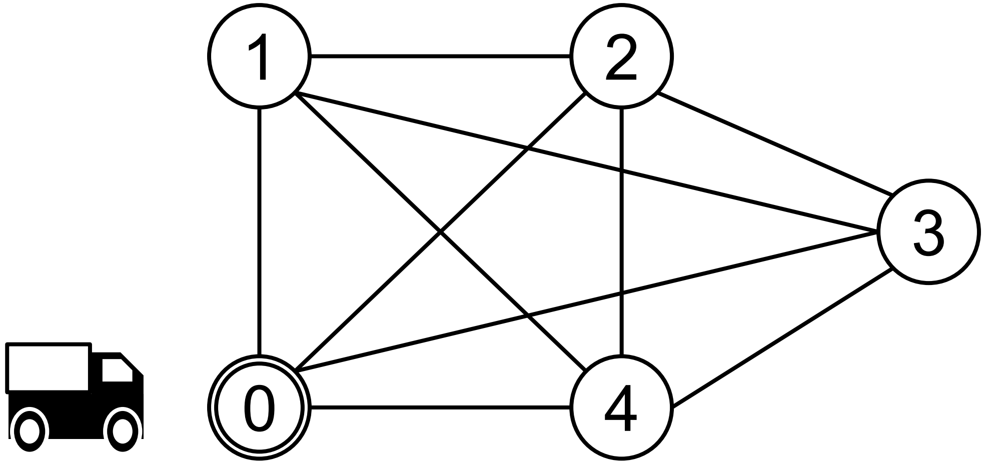
compute $V(\{1,2,3,4\},0)$ — **Target state** (original problem)

$$V(U,i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ \boxed{0 \quad \text{if } U = \emptyset} \end{cases}$$

Transition (visiting one customer)

Base case (all customers are visited)

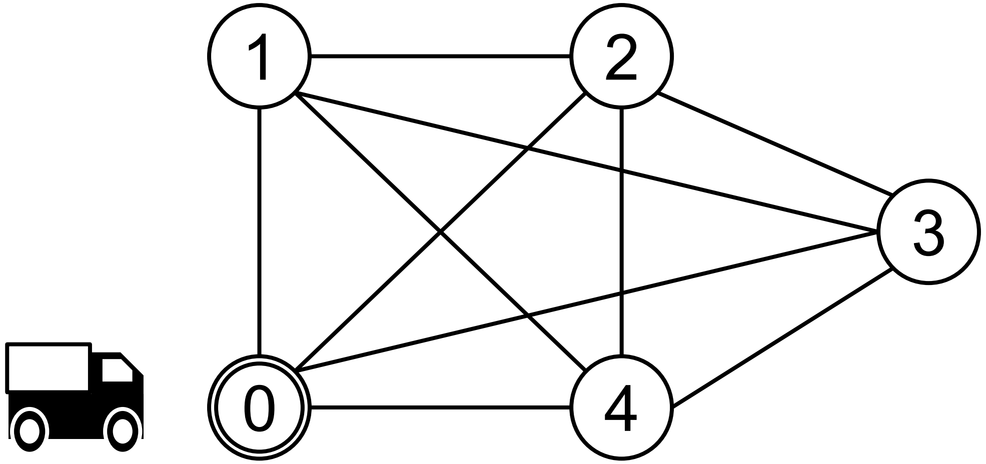
$V(U,i) \geq h(U,i)$ — **Dual bound function**



DP Model for the Example Problem

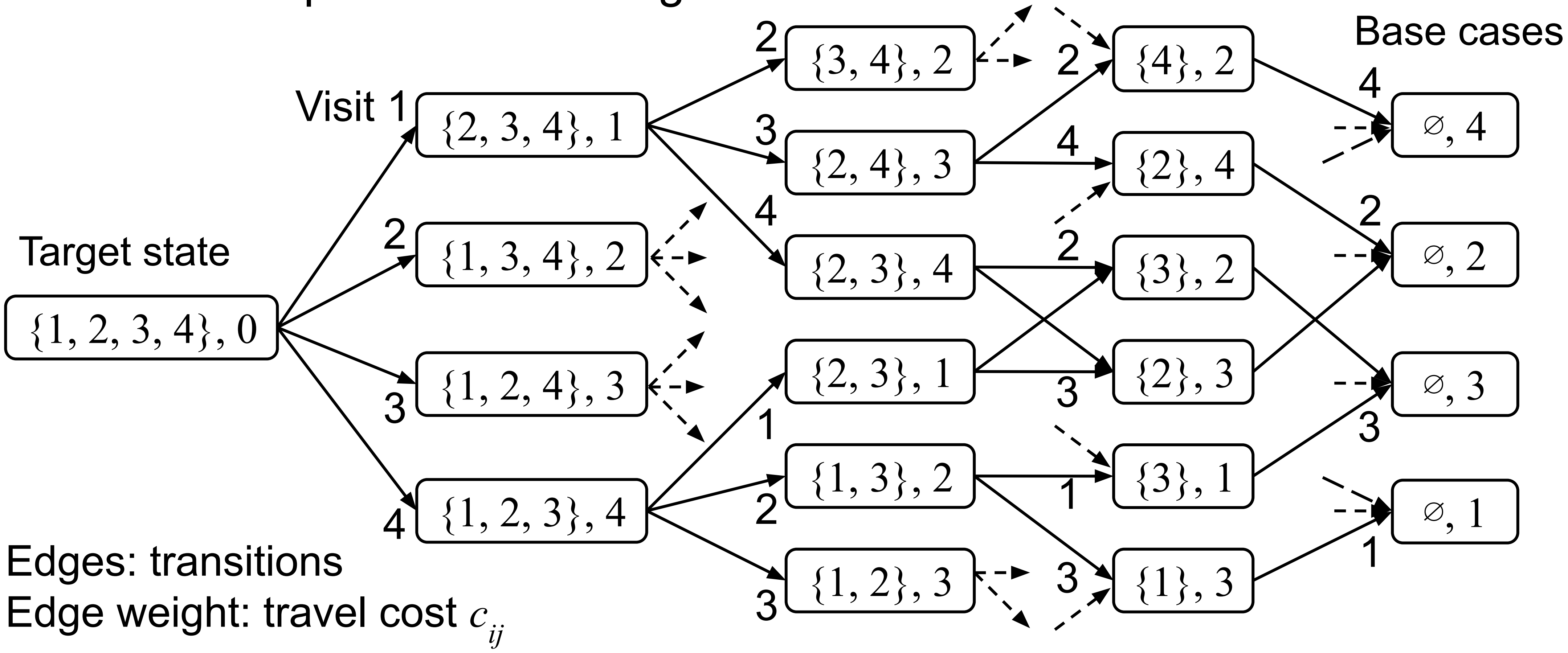
Recursively defined **value function** V maps a **state** (subproblem), defined by unvisited customers U and the current customer i, to the cost

compute $V(\{1,2,3,4\},0)$	**Target state** (original problem)
$V(U,i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$	**Transition** (visiting one customer) **Base case** (all customers are visited)
$V(U,i) \geq h(U,i)$	**Dual bound function**



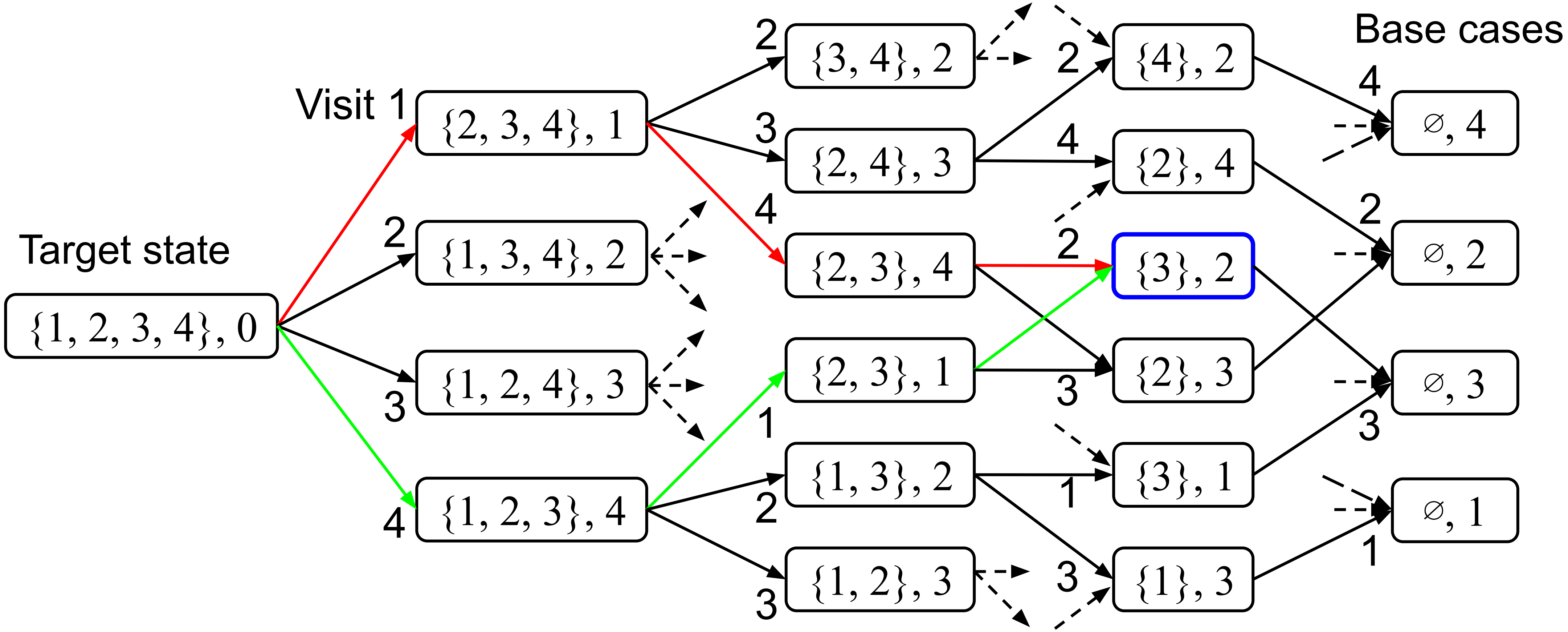
DP as State Space Search

- $V(S)$: the shortest path cost from S to a base case in a state space
- Solution: a path from the target state



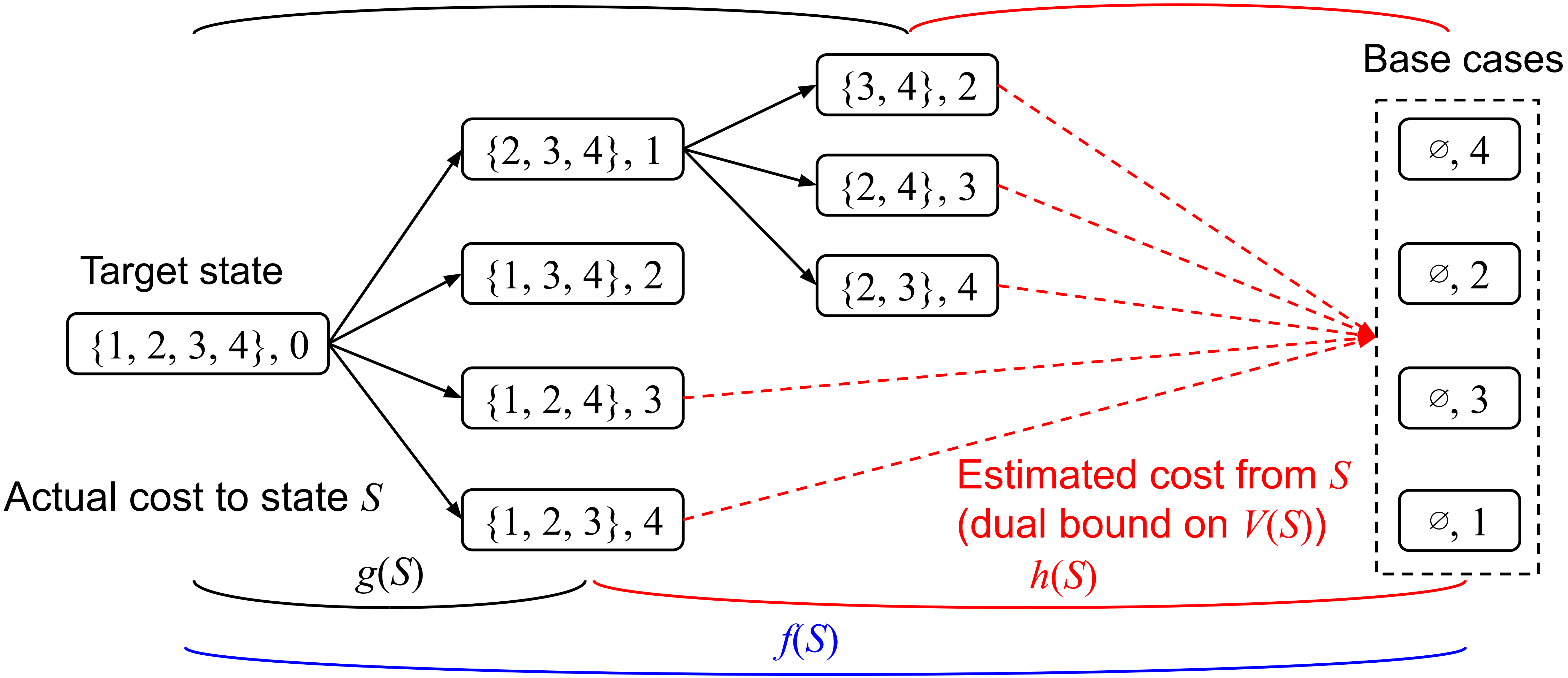
DP as State Space Search

Different paths can lead to the same state => store states in memory



Heuristic Search for DIDP

Guide the search using $f(S) = g(S) + h(S)$ (LB on the path cost via S)



Beam Search

Keep the best b states according to the f-value in each layer

Beam width: $b = 2$

Target state

$\boxed{\{1, 2, 3, 4\}, 0}$ f: 10

Beam Search

Keep the best b states according to the f-value in each layer

Beam width: $b = 2$

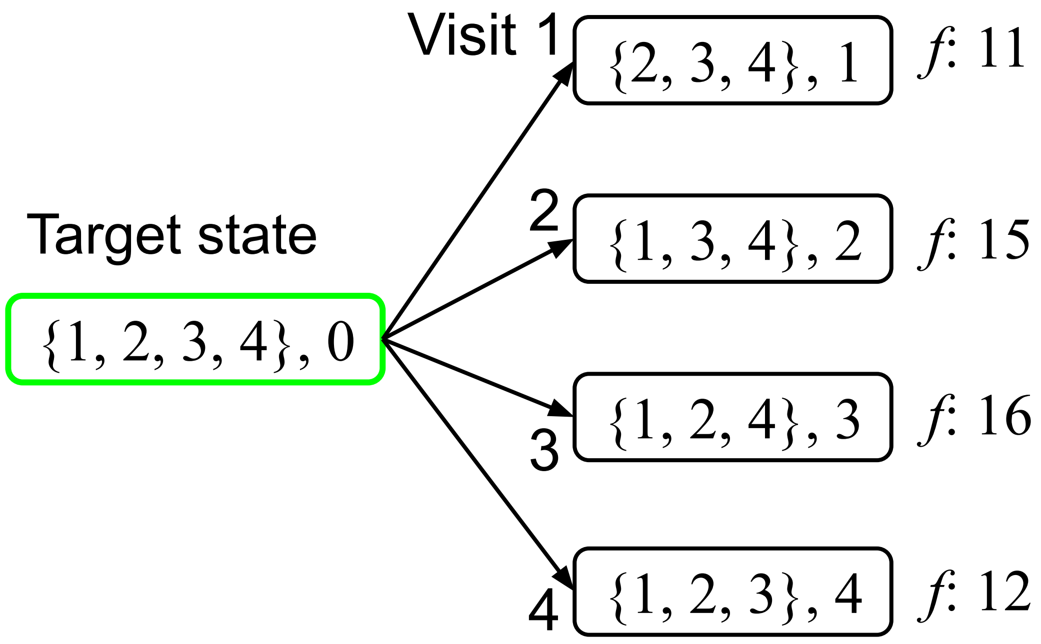
Target state

$\boxed{\{1, 2, 3, 4\}, 0}$ f: 10

Beam Search

Keep the best b states according to the f-value in each layer

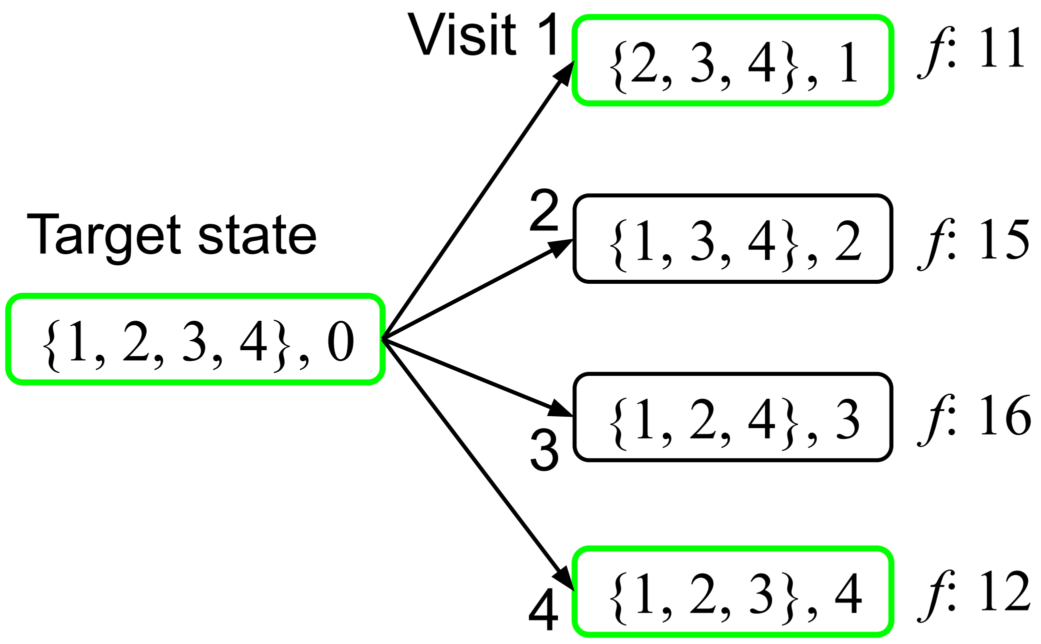
Beam width: $b = 2$



Beam Search

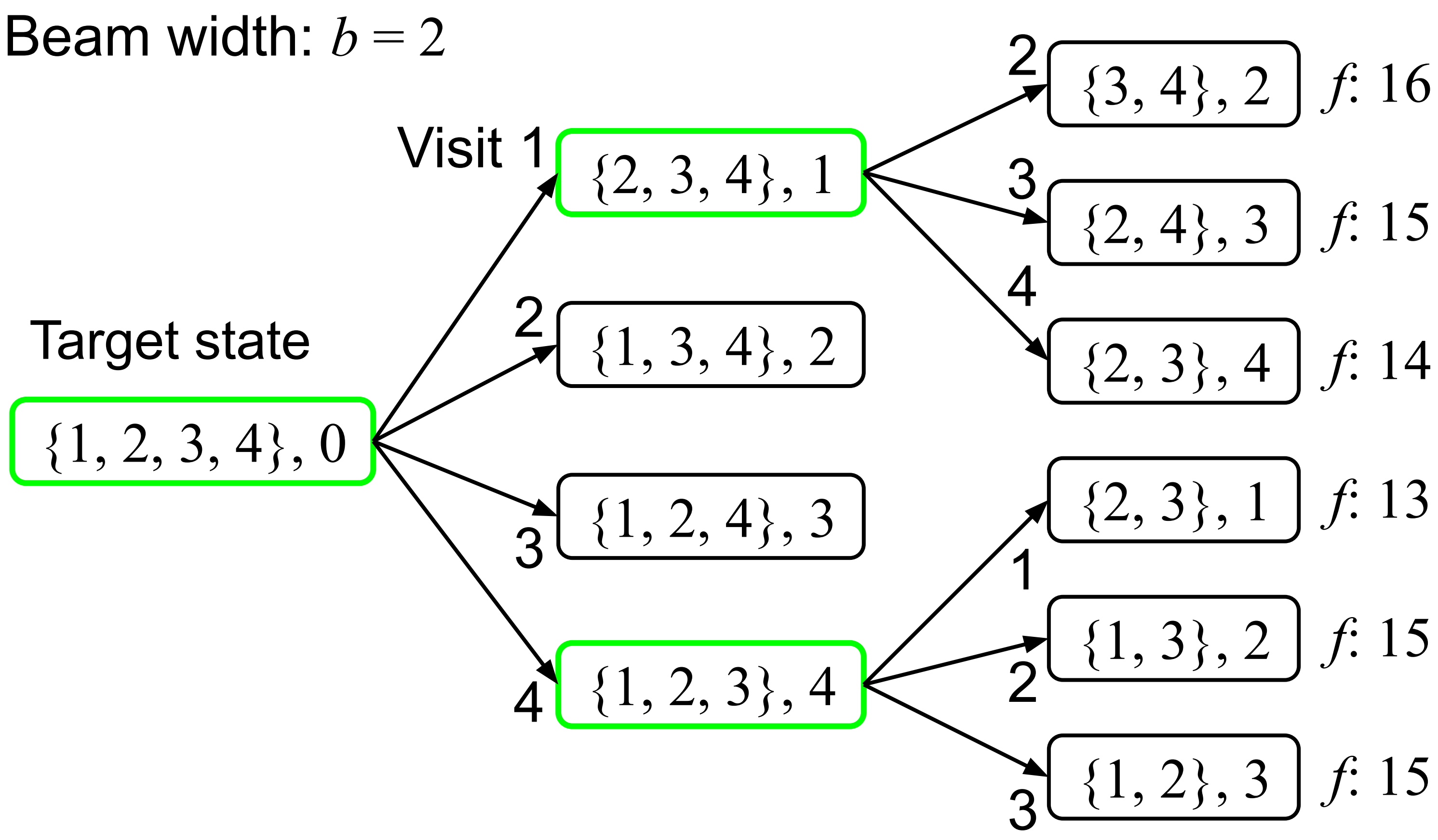
Keep the best b states according to the f-value in each layer

Beam width: $b = 2$



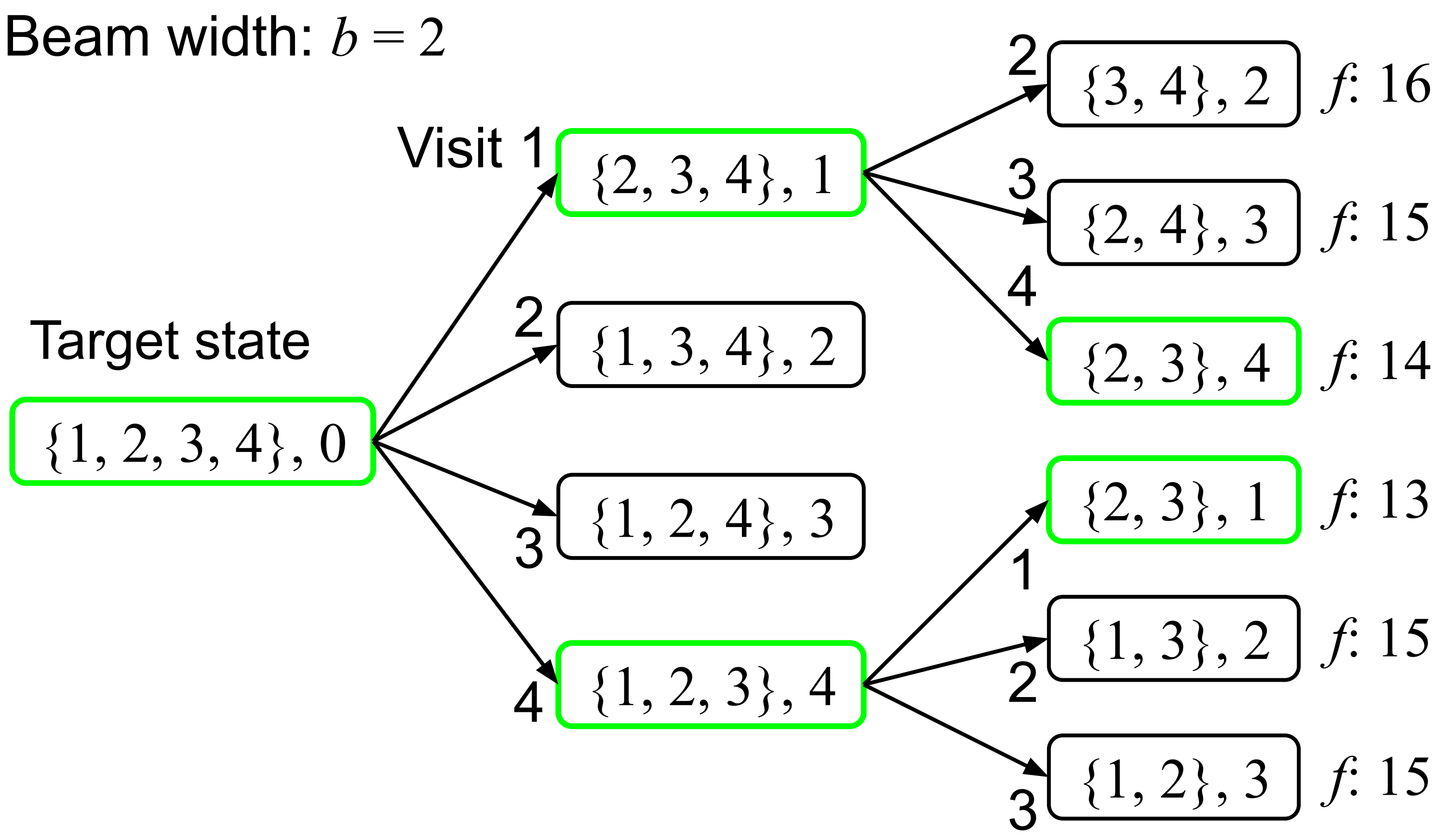
Beam Search

Keep the best b states according to the f-value in each layer



Beam Search

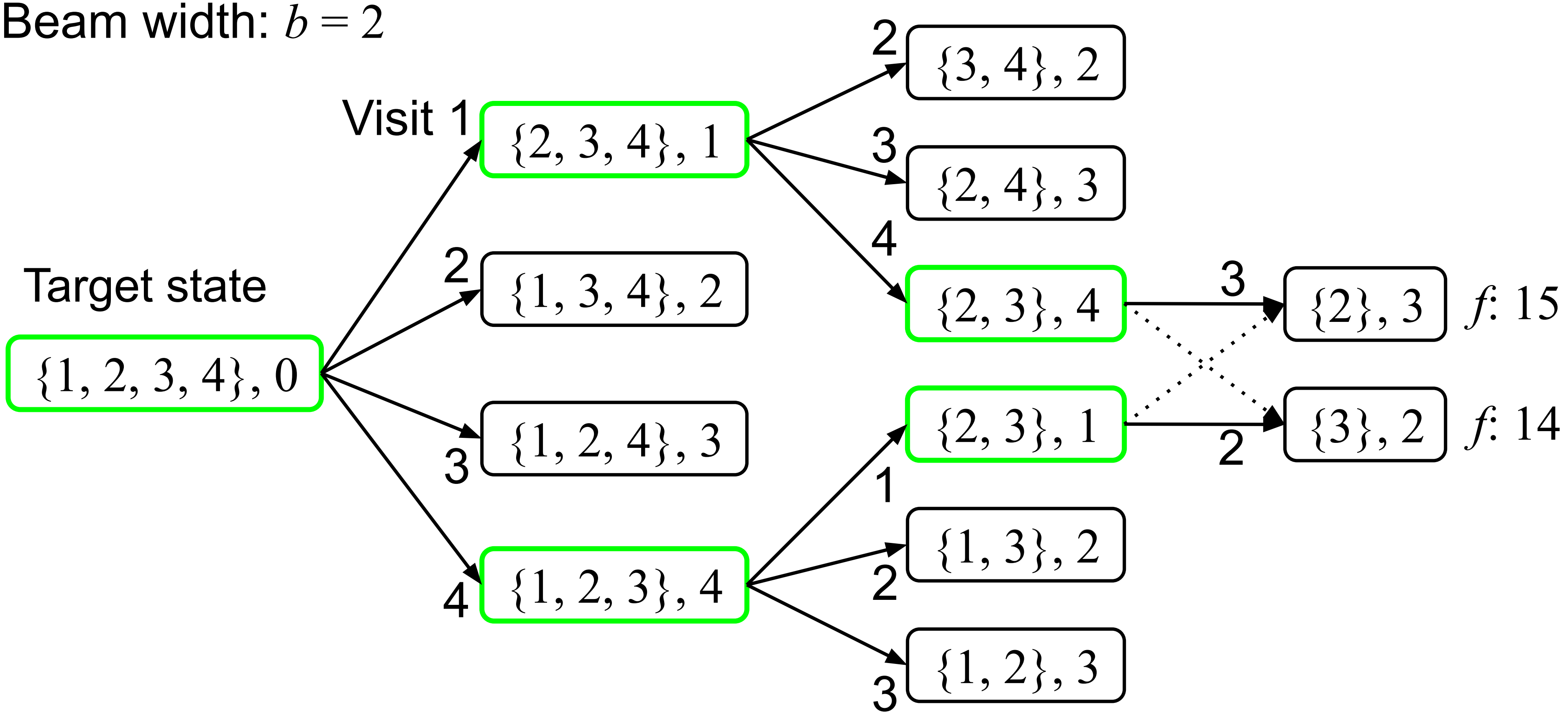
Keep the best b states according to the f-value in each layer



Beam Search

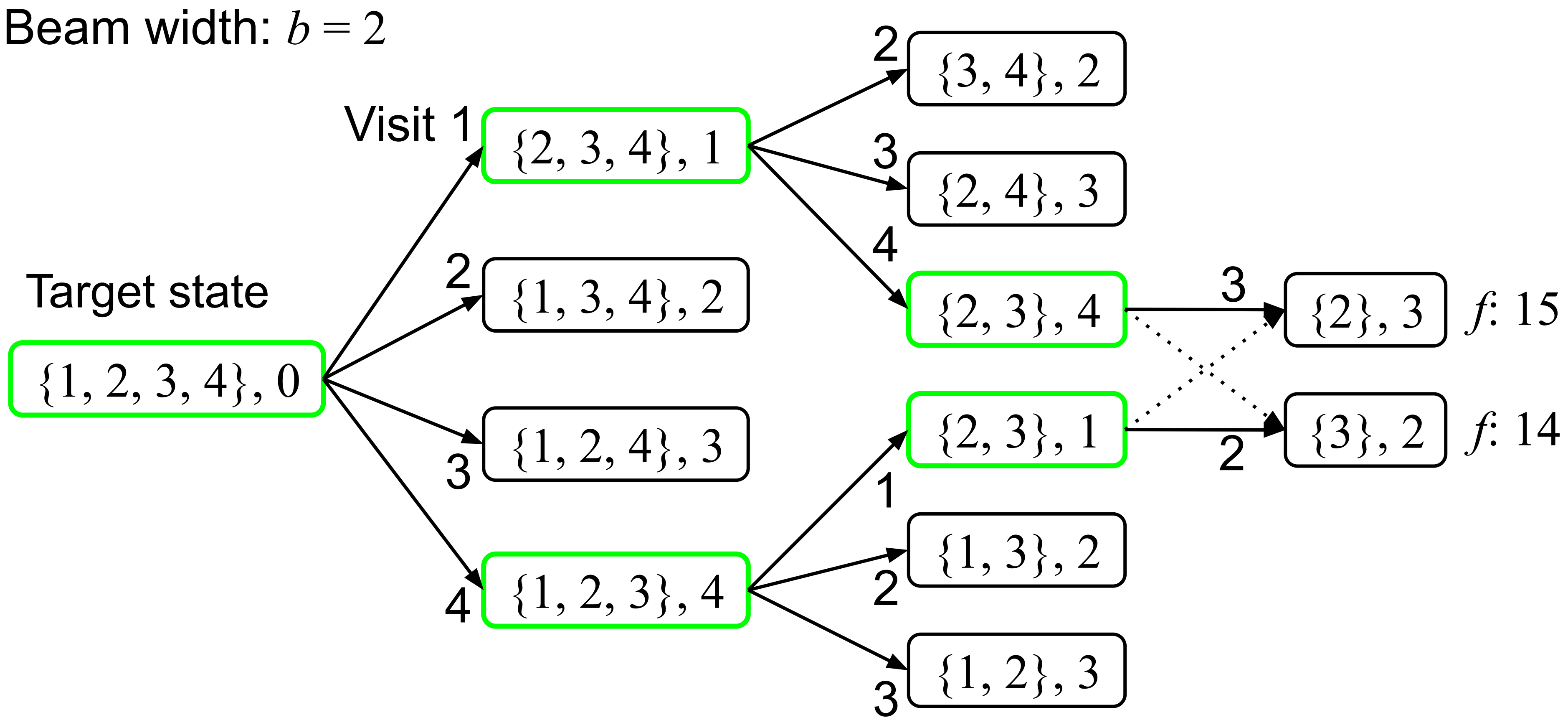
Keep the best b states according to the f-value in each layer

Beam width: $b = 2$



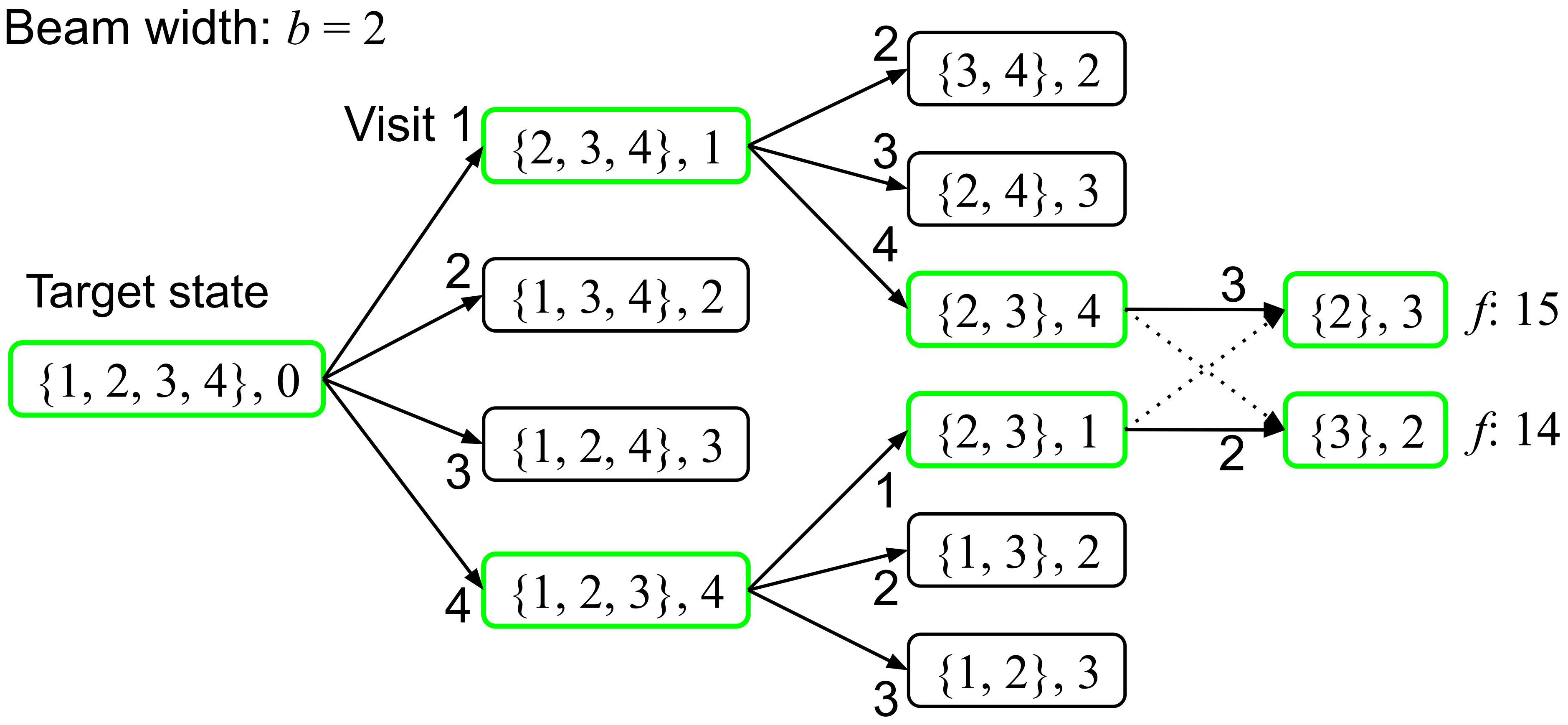
Beam Search

Keep the best b states according to the f-value in each layer



Beam Search

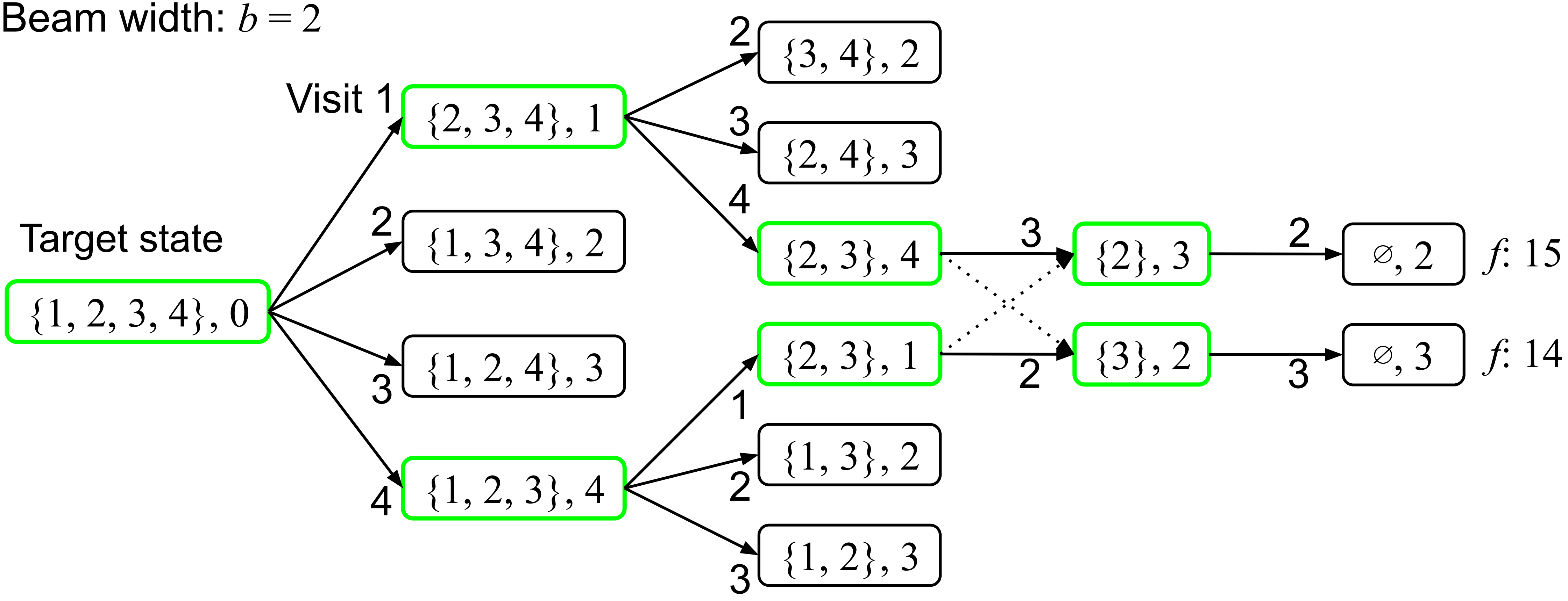
Keep the best b states according to the f-value in each layer



Beam Search

Keep the best b states according to the f-value in each layer

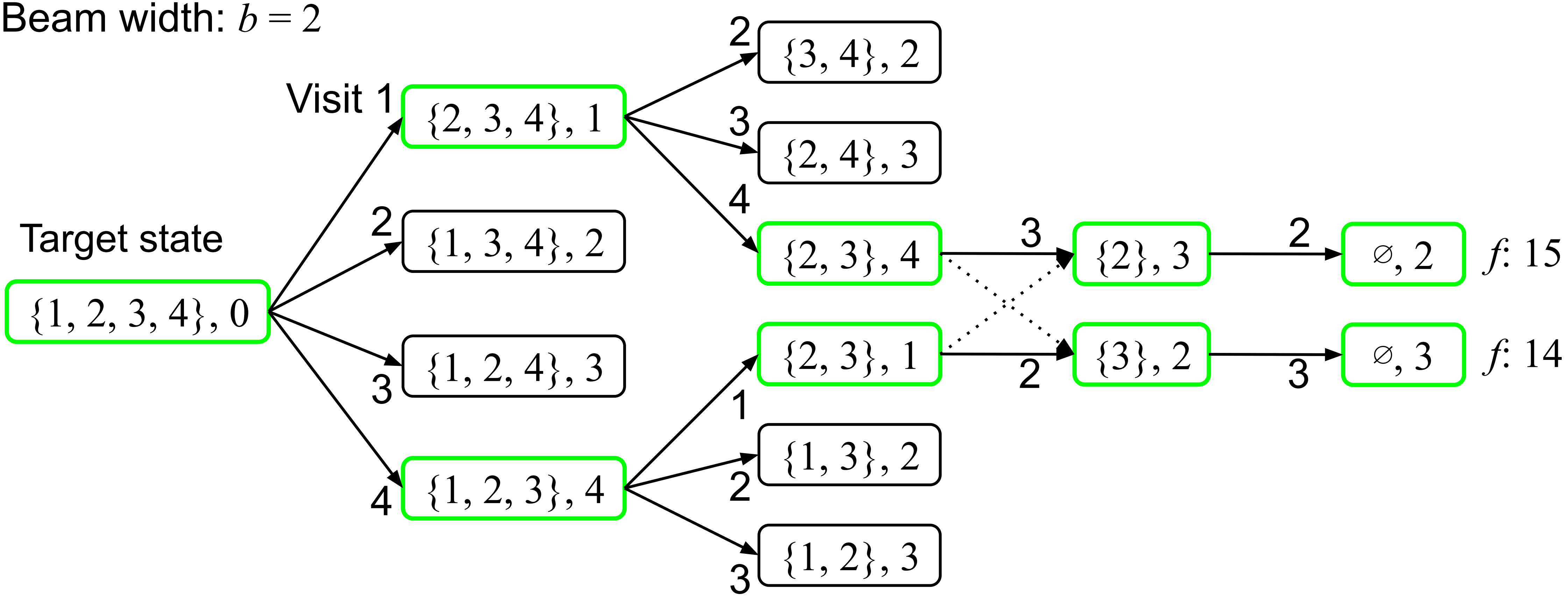
Beam width: $b = 2$



Beam Search

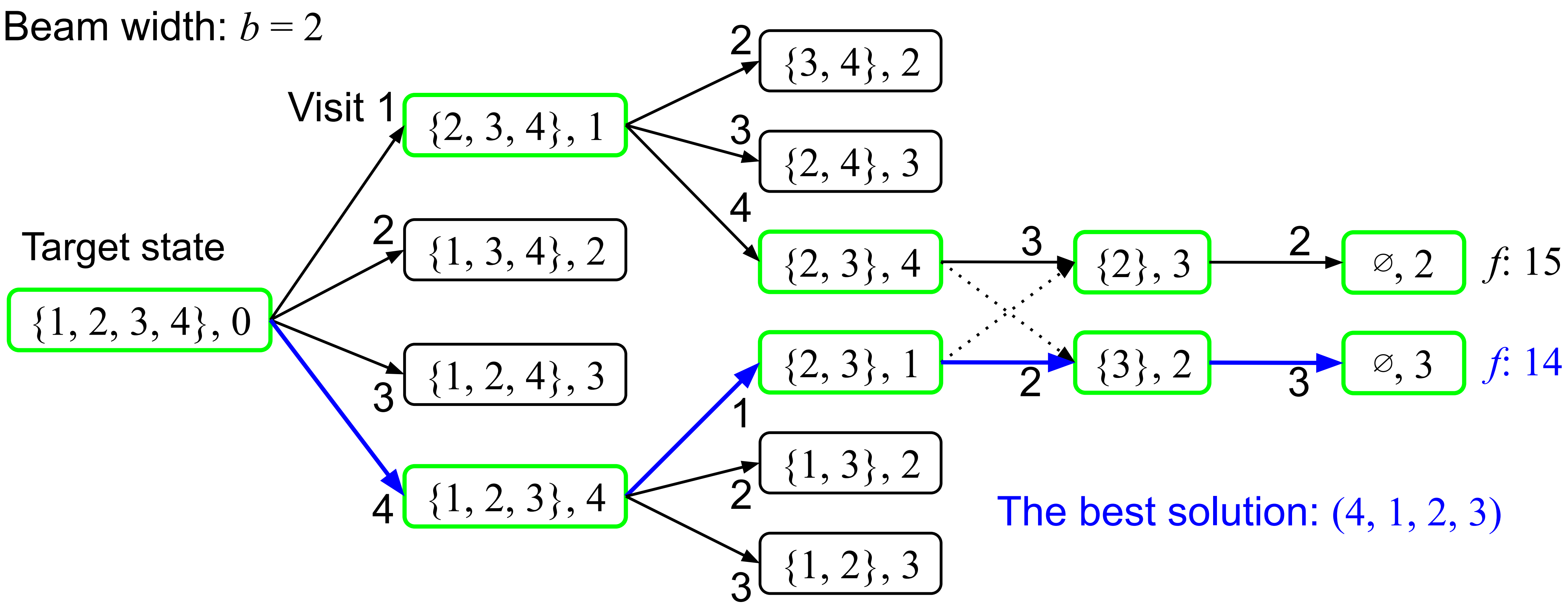
Keep the best b states according to the f-value in each layer

Beam width: $b = 2$



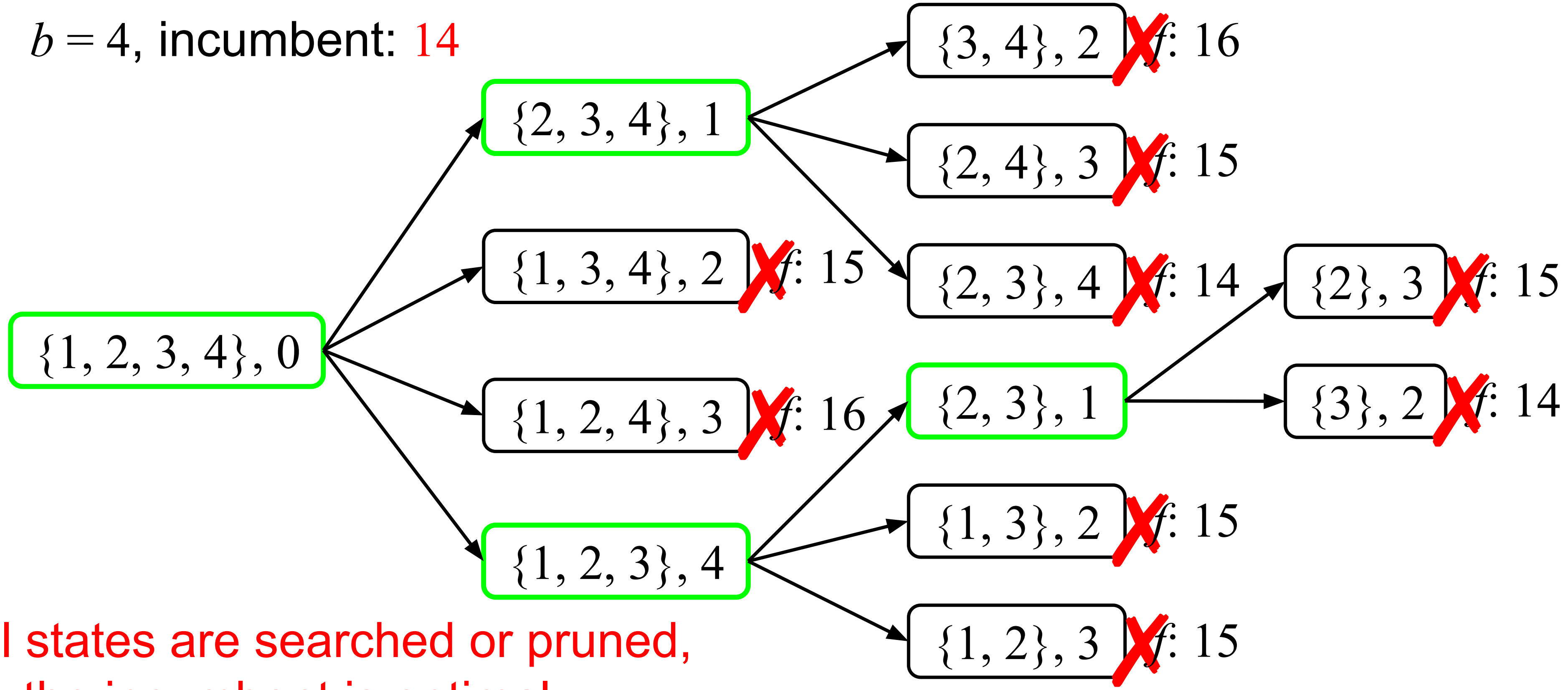
Beam Search

Keep the best b states according to the f-value in each layer



SOTA: Complete Anytime Beam Search (CABS)

- Beam search with $b = 1, 2, 4, 8, \ldots,$ until exhausting the state space
- Prune a state S if $f(S) \geq$ the incumbent solution cost



All states are searched or pruned, so the incumbent is optimal

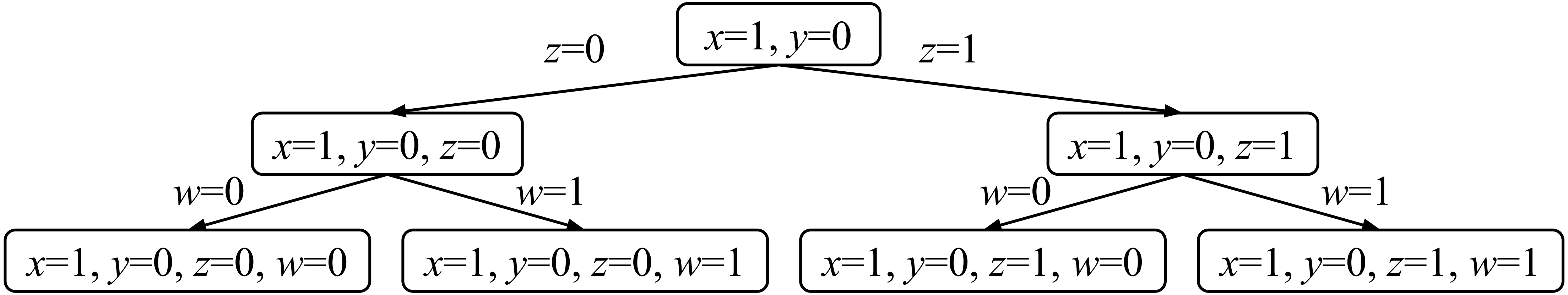
Zhang 1998; Kuroiwa and Beck ICAPS 2023b

Large Neighborhood Beam Search

Large Neighborhood Search (LNS)

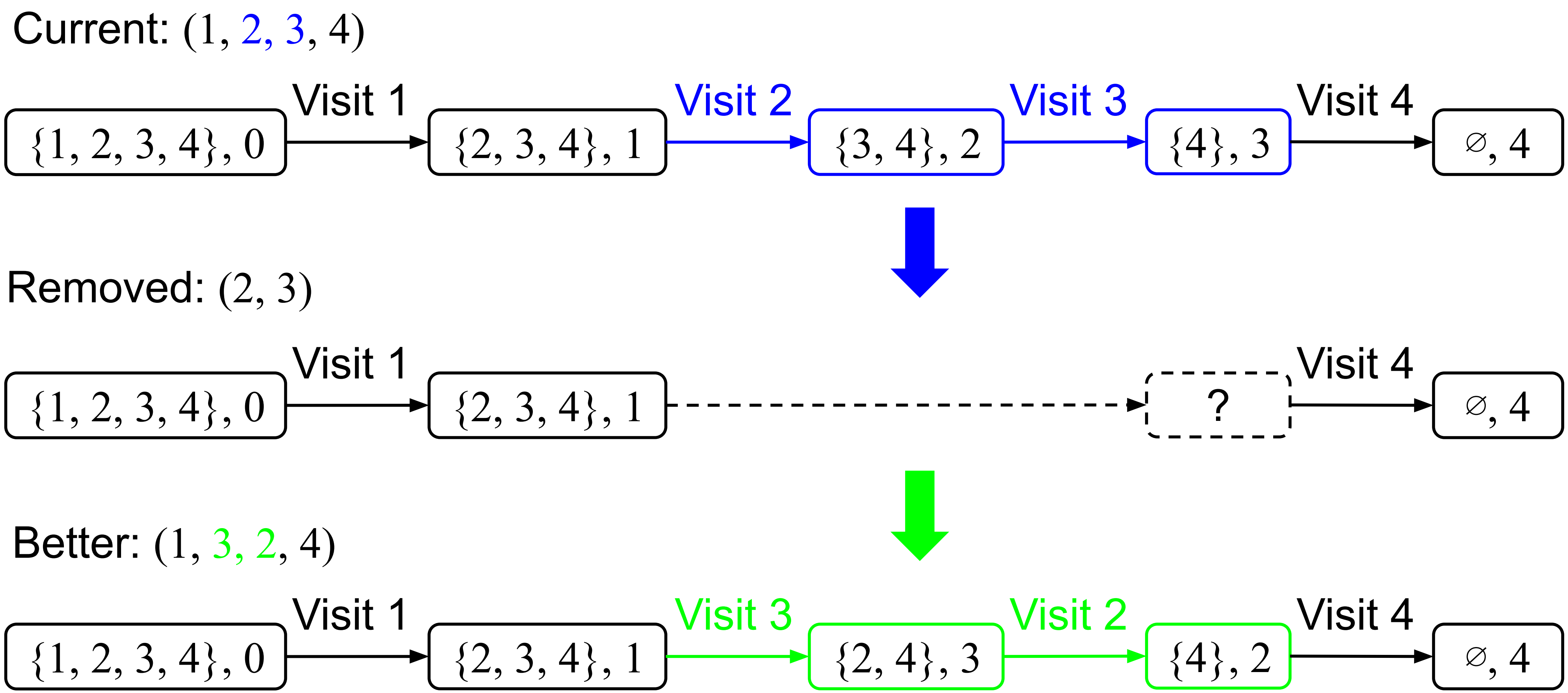
LNS for CP: remove value assignments to some variables from a solution and perform tree search to find a better solution

Current solution: $x = 1, y = 0, z = 0, w = 0$
Removed assignments: $z = 0, w = 0$

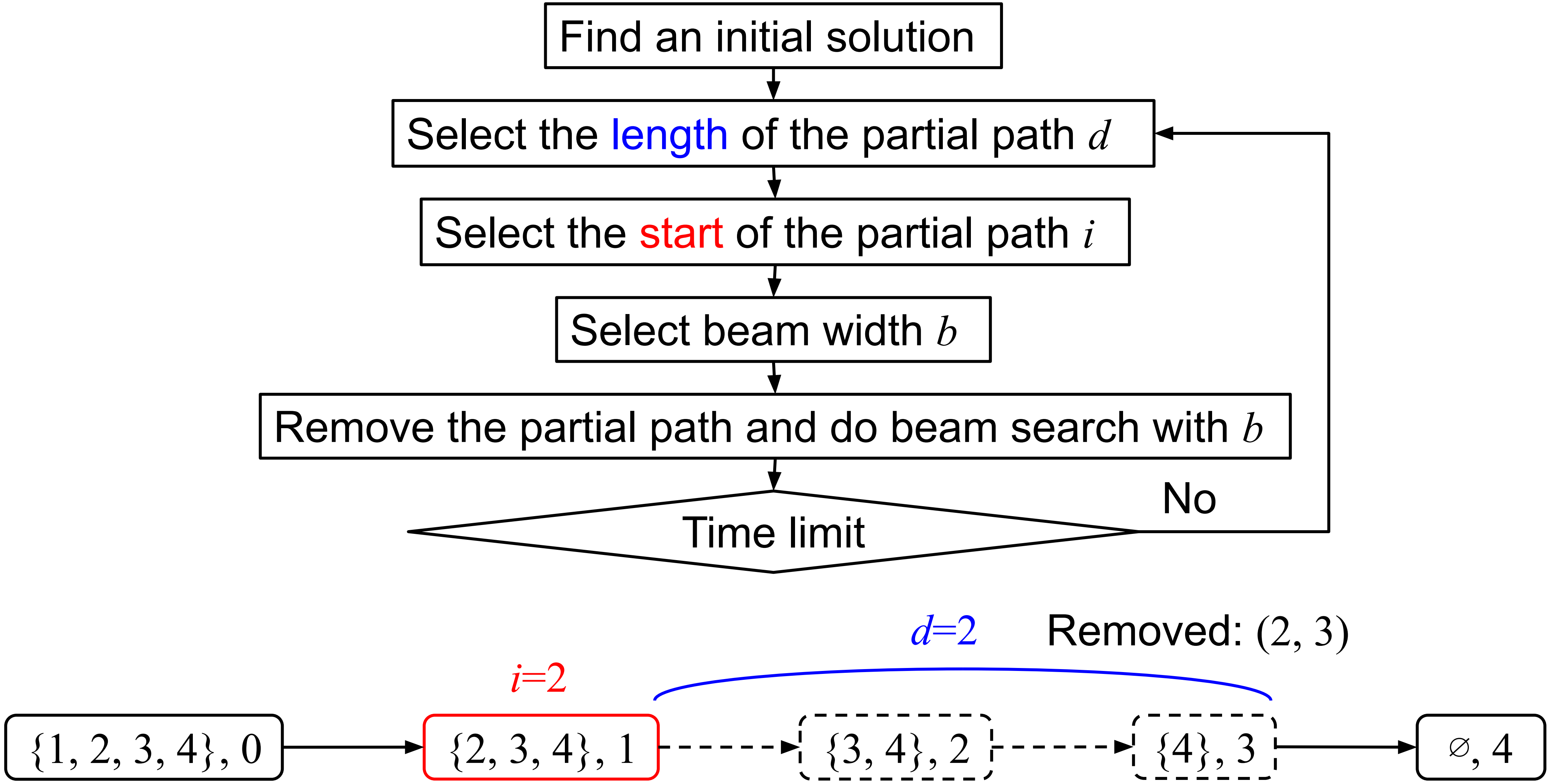


LNS for DIDP

Remove **a partial path** and search in a **partial state space**



Large Neighborhood Beam Search (LNBS)



Multi-Armed Bandit-Based Length Selection

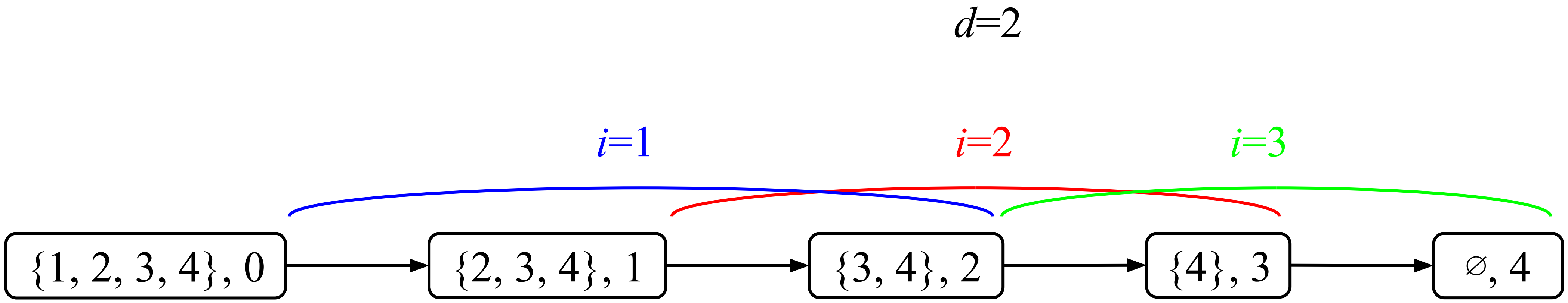
How many transitions to remove given the remaining time 0.8?

	length $d = 2$ (arm)	length $d = 4$ (arm)
	Average cost improvement: 0.5 Average time: 0.01 # of times $d = 2$ used: 10	Average cost improvement: 0.8 Average time: 0.02 # of times $d = 4$ used: 5
Random variables	Cost improvement (reward) r_2 Time t_2	Cost improvement (reward) r_4 Time t_4

=> Use Budgeted-UCB [Xia et al. 2015] to decide

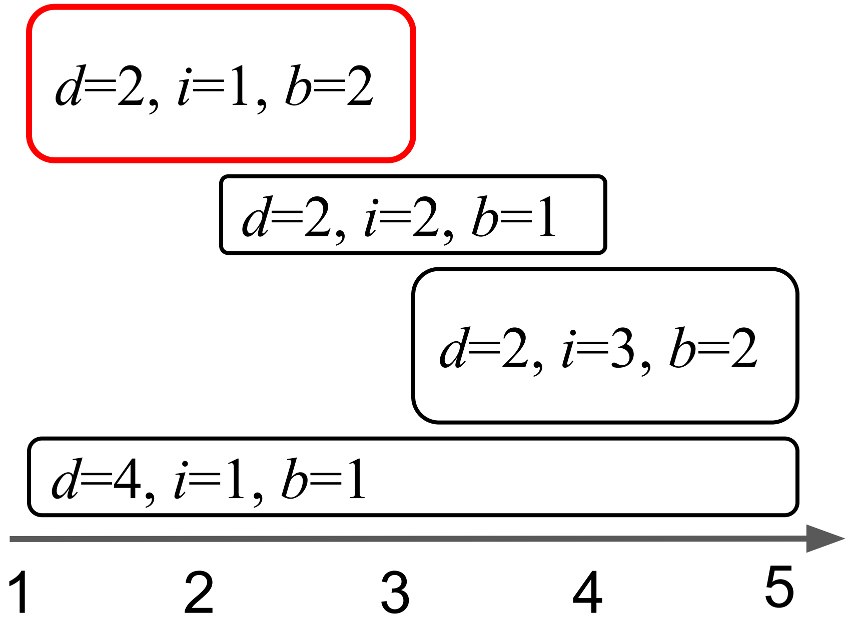
Start Selection

Given length d, sample start i uniformly at random

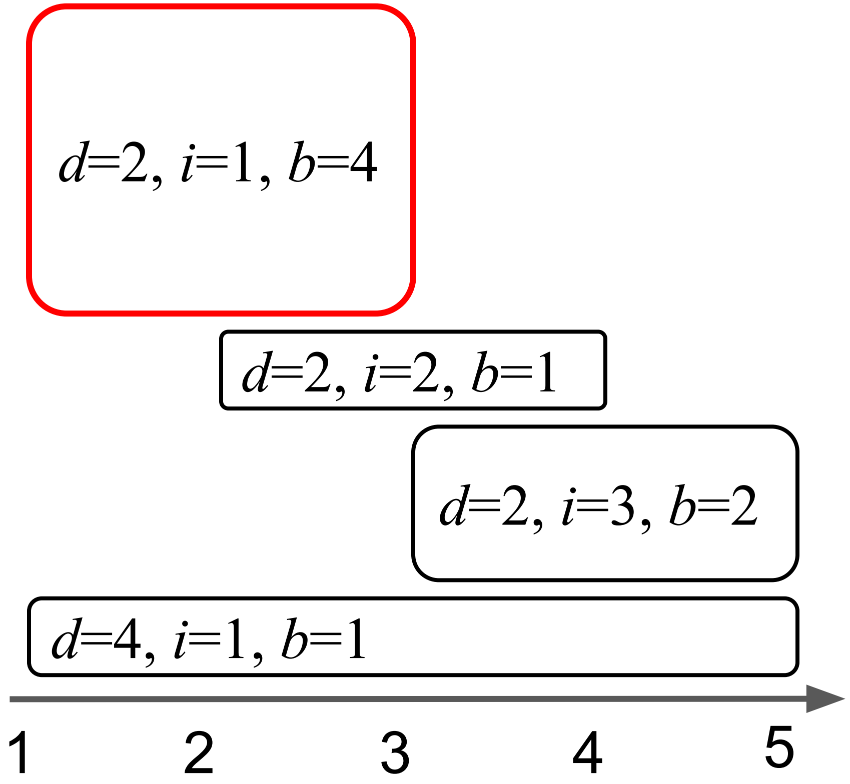


Beam Width Selection

Double beam width b_{di} for length d and start i after each beam search starting from b_{di}=1

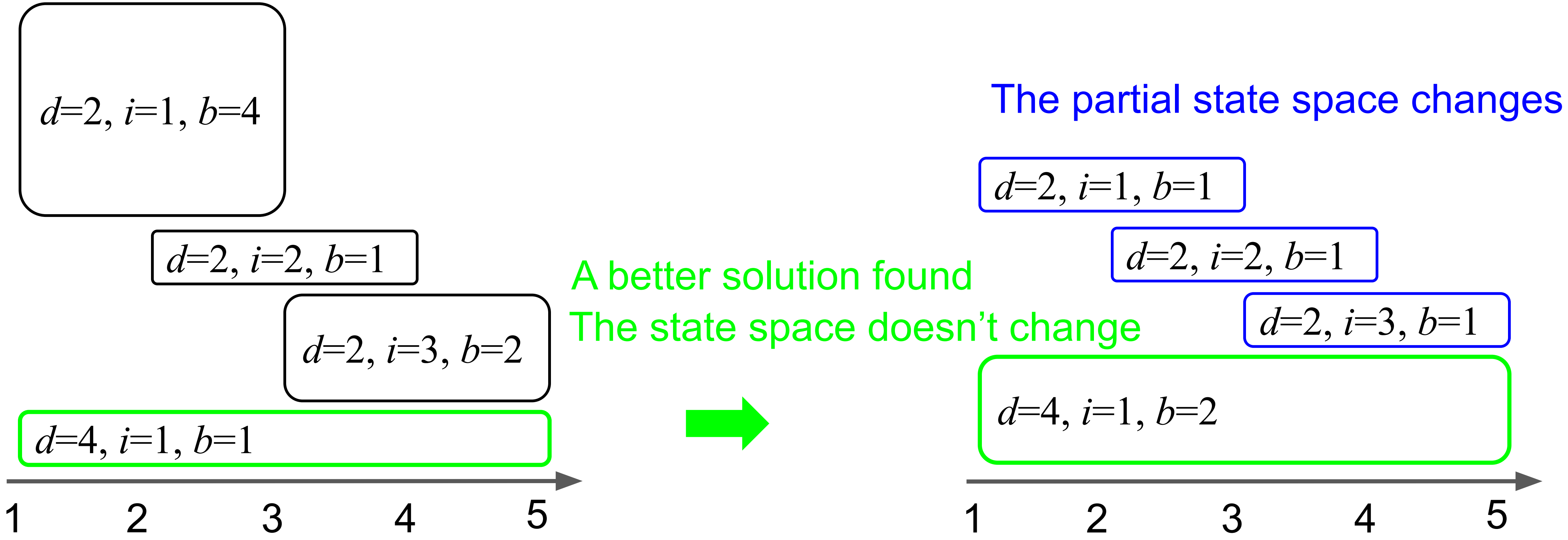


No better solution found



Beam Width Selection

- Reset b_{di} to 1 if the partial state space changes
- Prove optimality if i=1, d is the solution length, and b is large enough

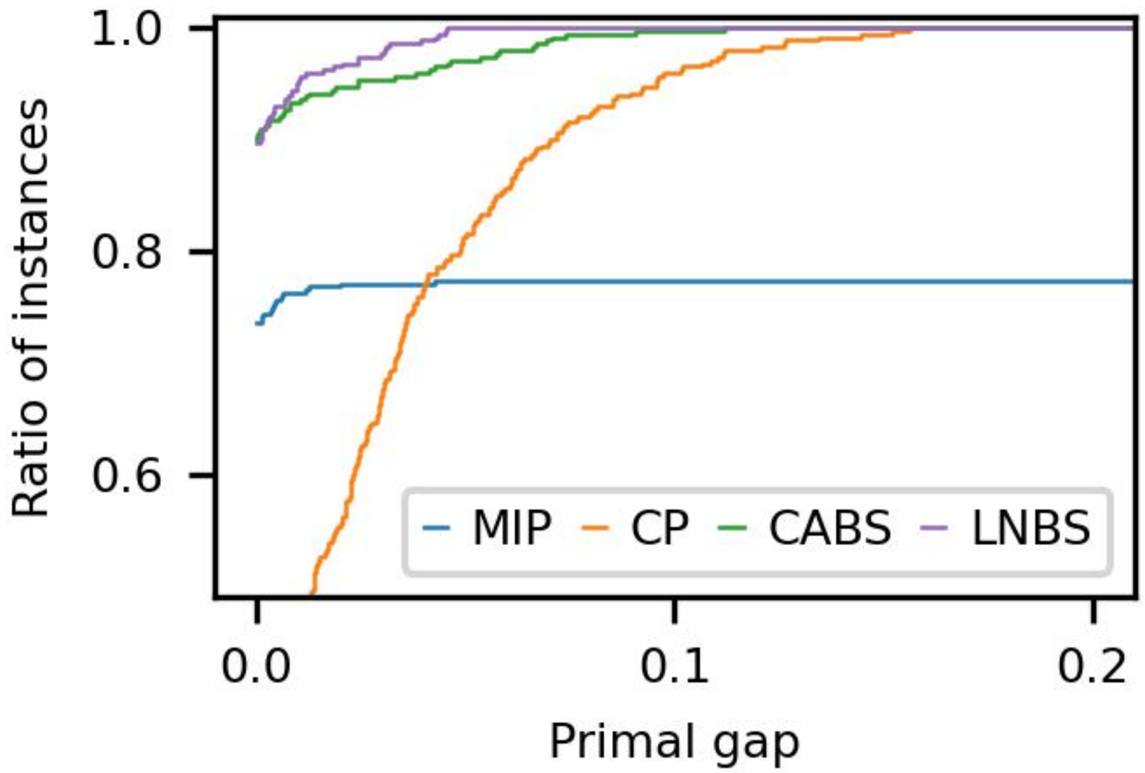


Experimental Evaluation

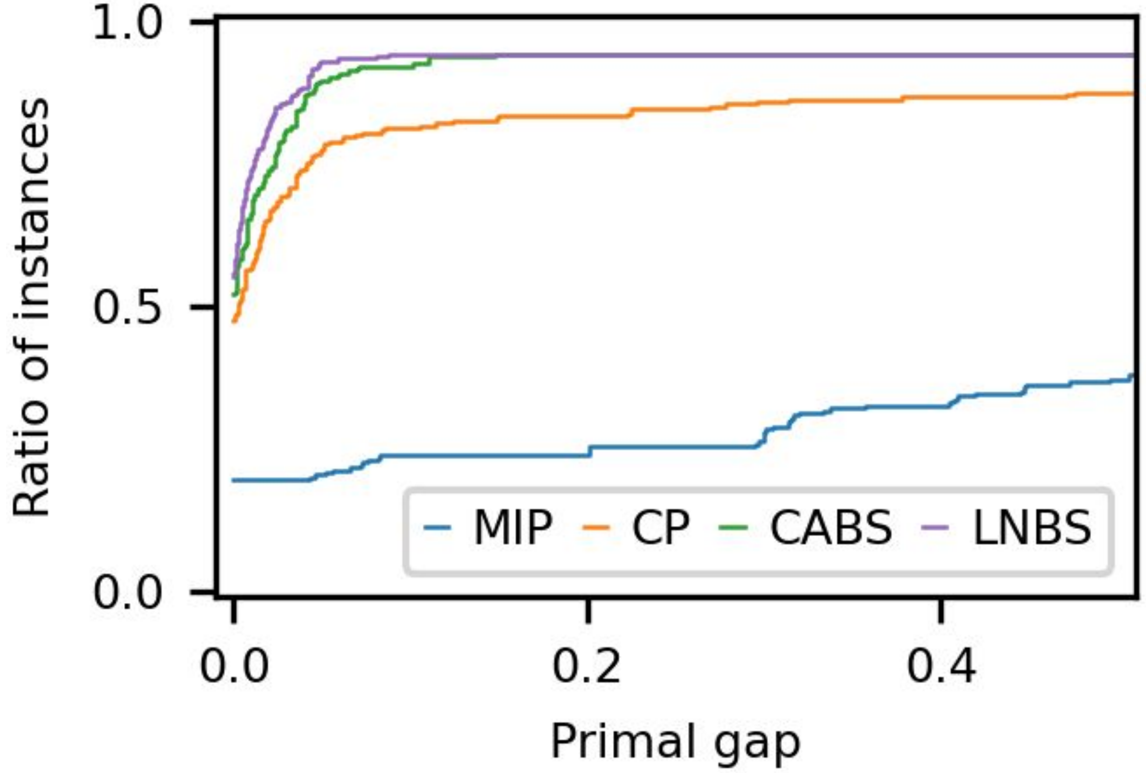
Distribution of Primal Gap in Routing Problems

Primal gap: relative gap to the best known solution (smaller is better) achieved with 30 min

TSP with Time Windows (TSPTW)



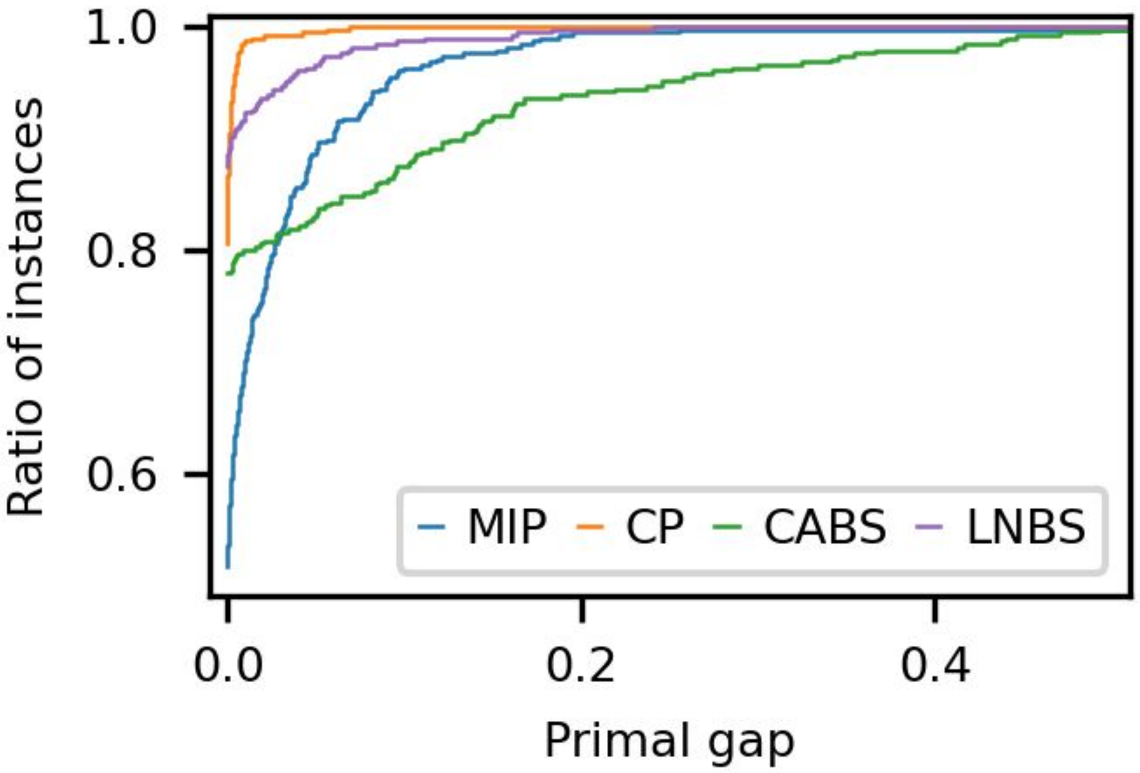
Pick-and-Delivery TSP (m-PDTSP)



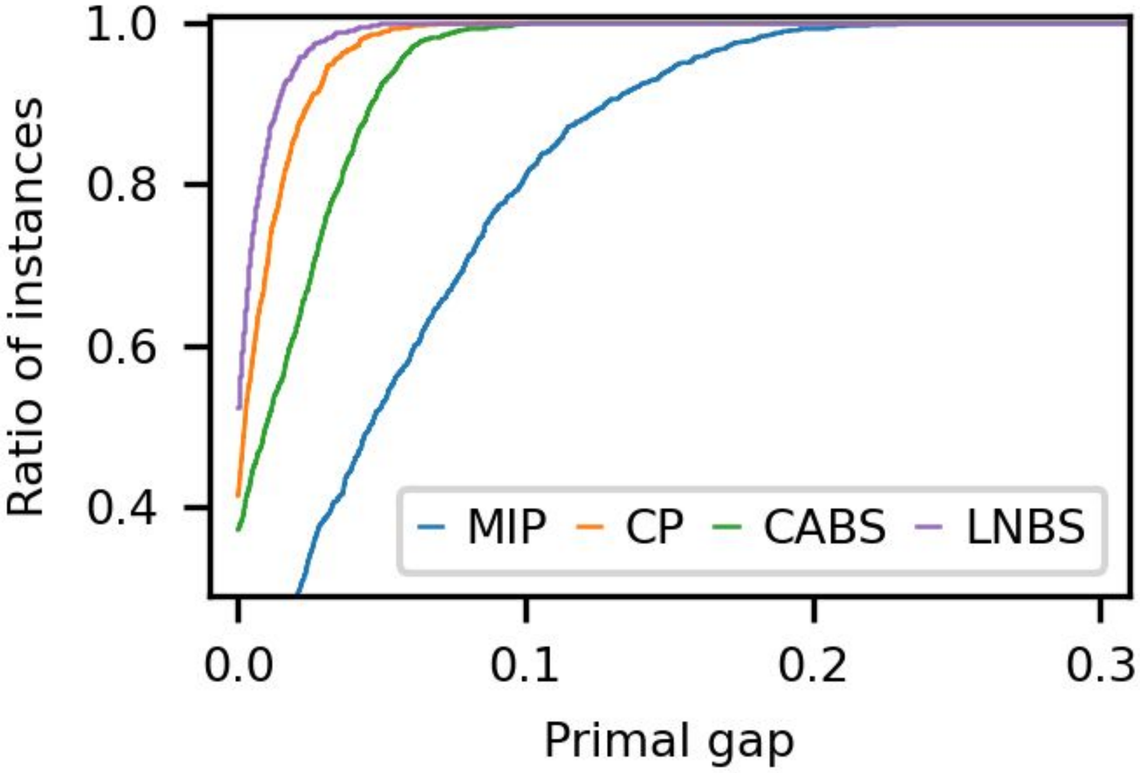
Distribution of Primal Gap in Scheduling Problems

Primal gap: relative gap to the best known solution (smaller is better) achieved with 30 min

Single Machine Total Weighted Tardiness



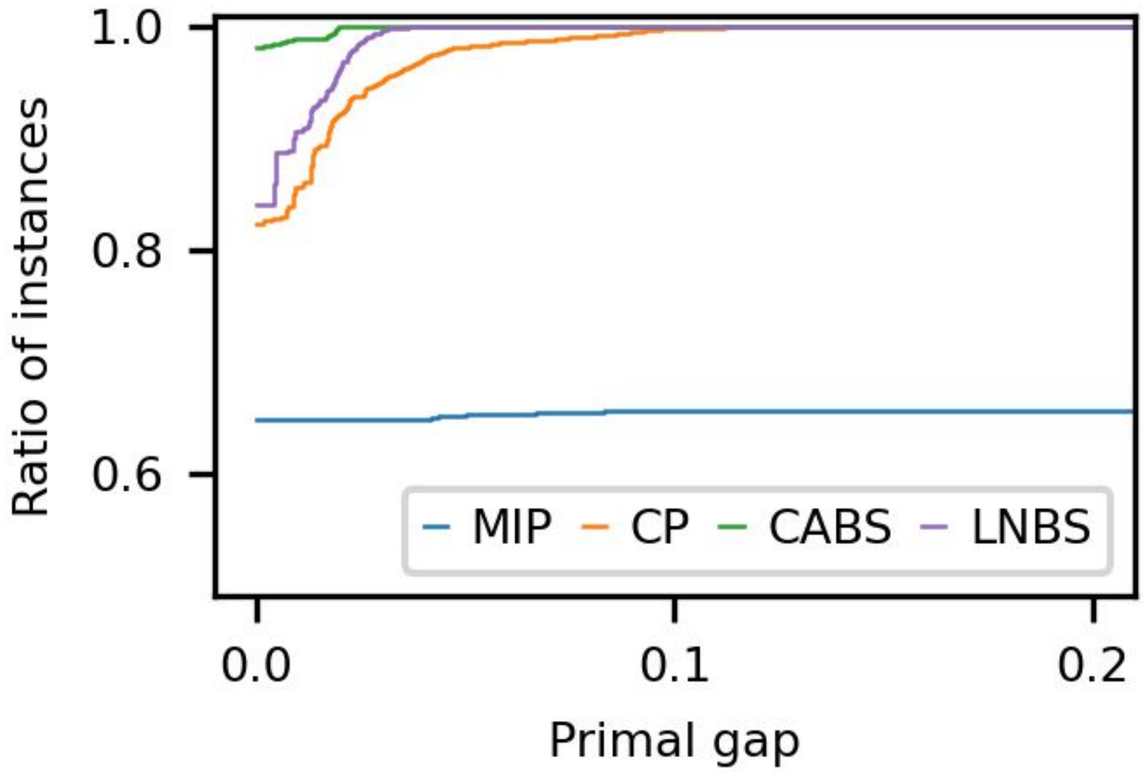
Talent Scheduling



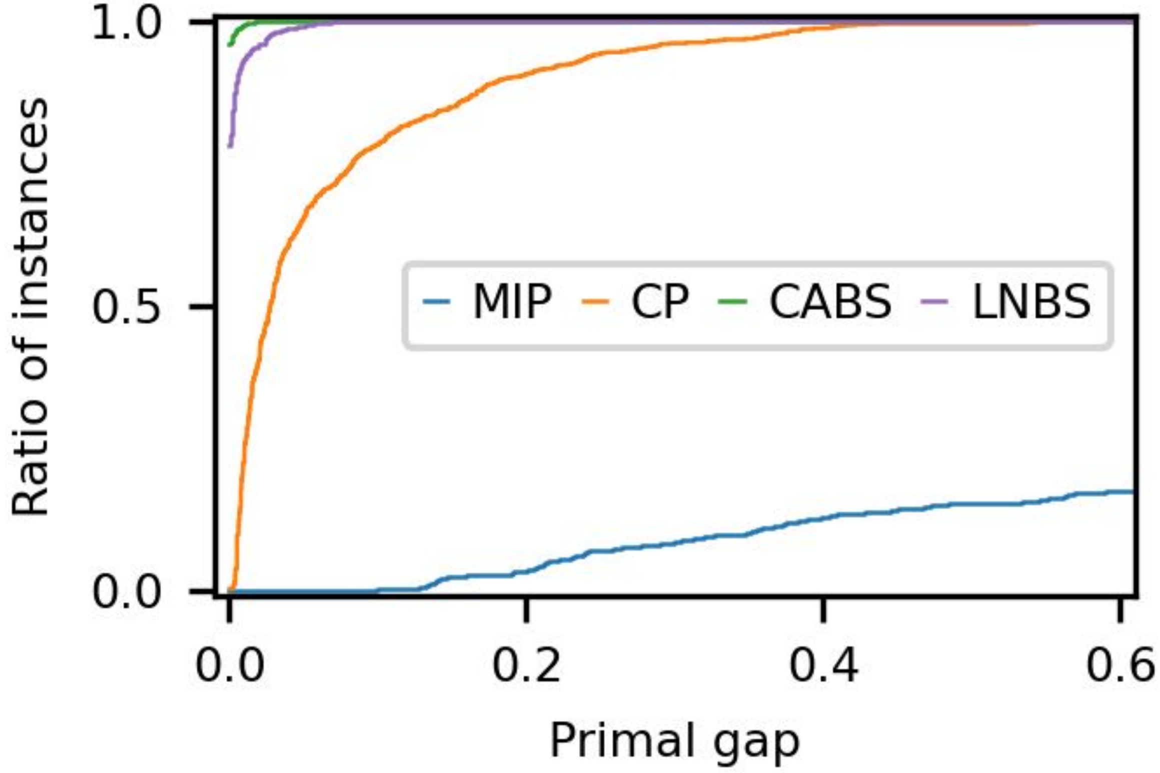
Distribution of Primal Gap in Other Problems

Primal gap: relative gap to the best known solution (smaller is better) achieved with 30 min

Simple Assembly Line Balancing (SALBP-1)

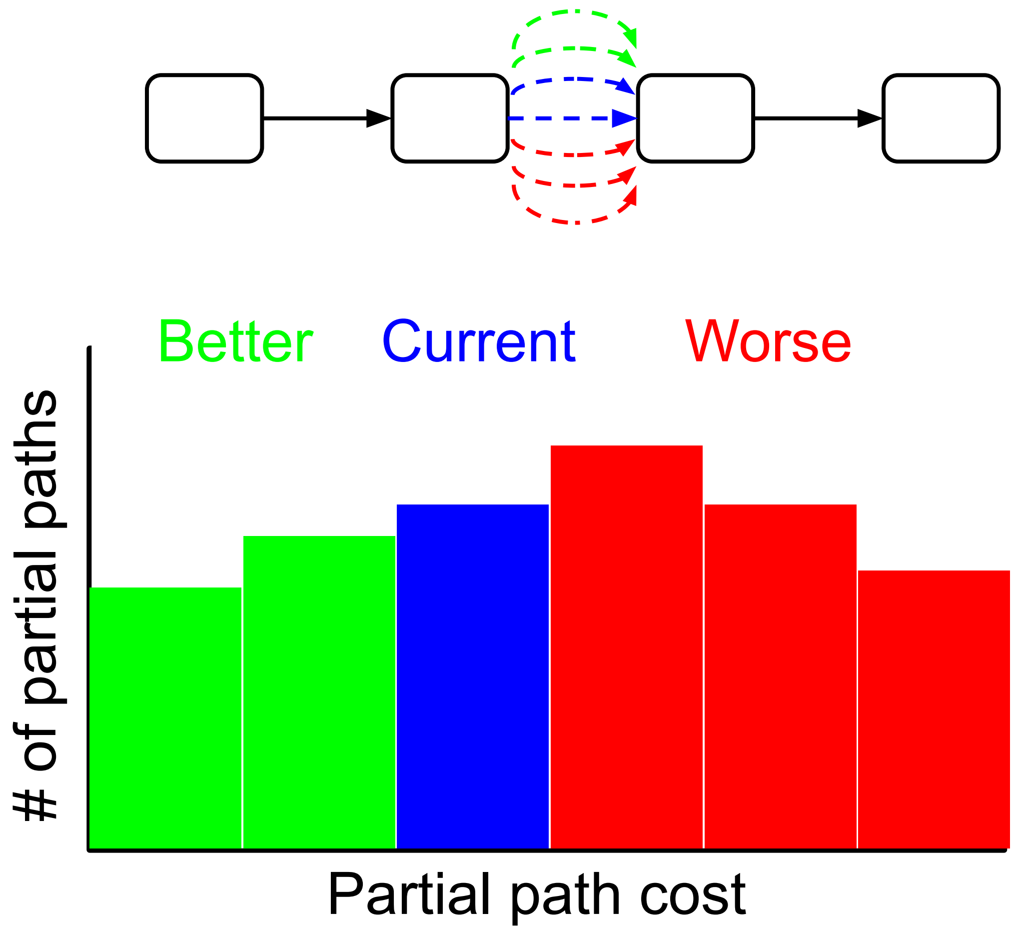


Minimization of Open Stacks (MOSP)

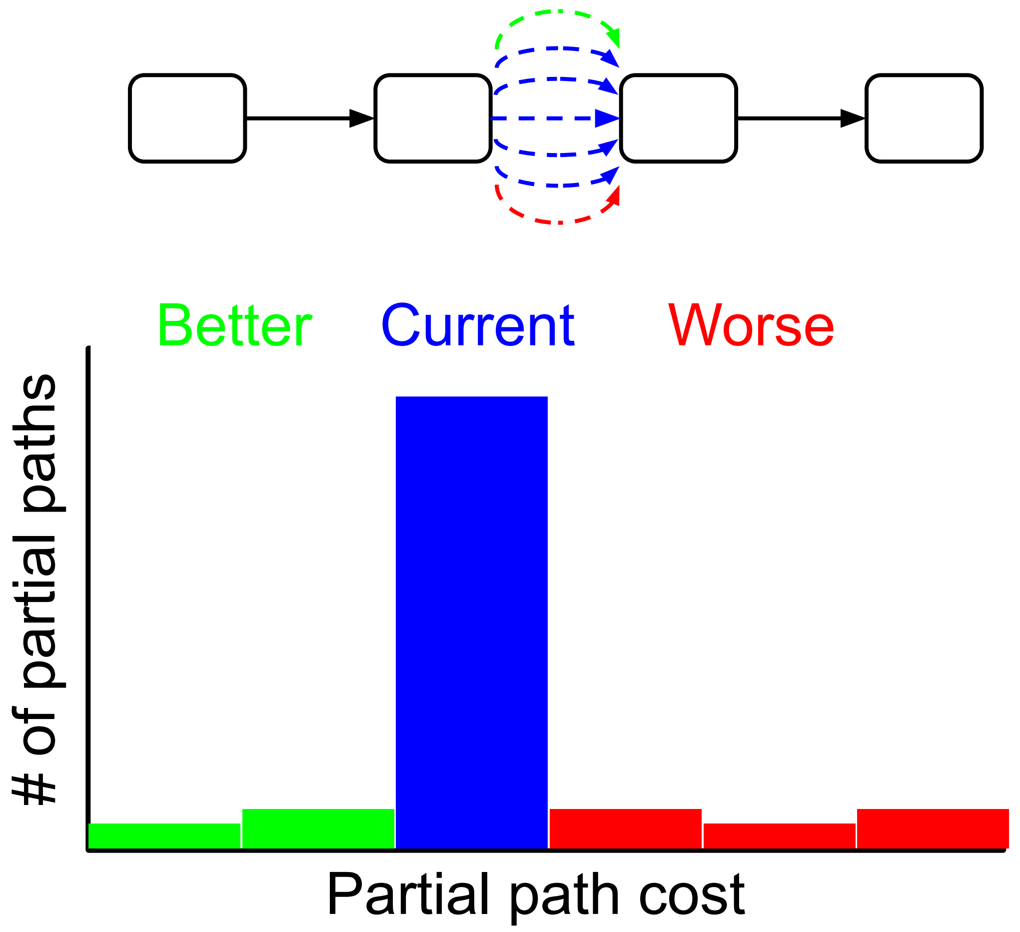


Why LNBS is Worse in Some Problems?

Diverse => easy to find a better one?



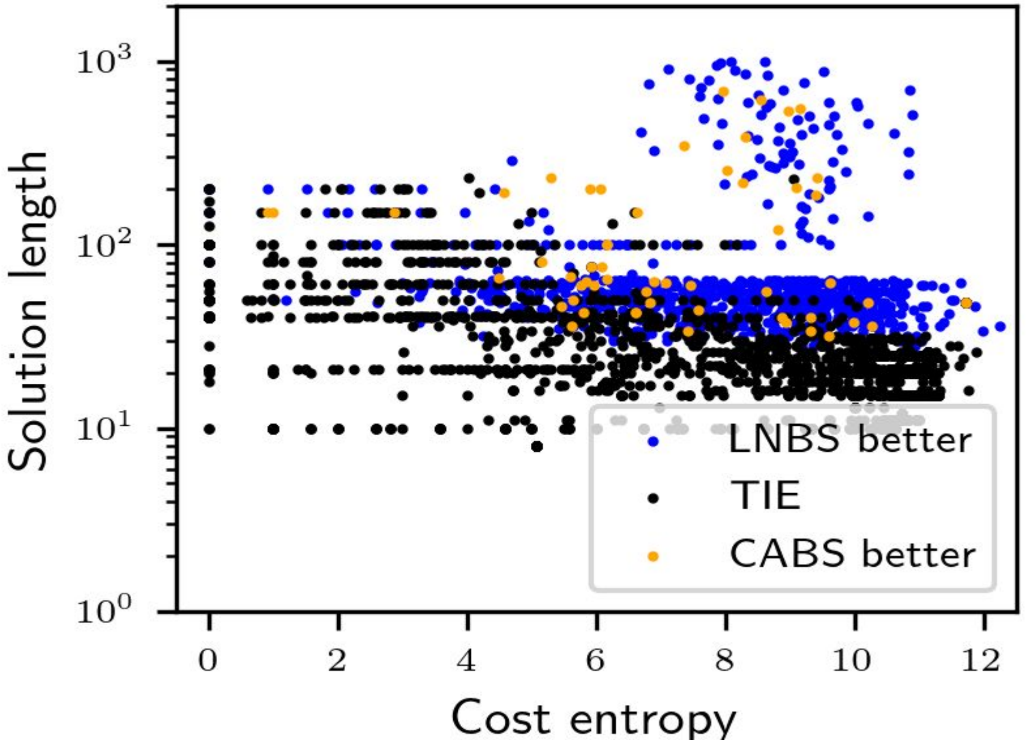
Not diverse => difficult?



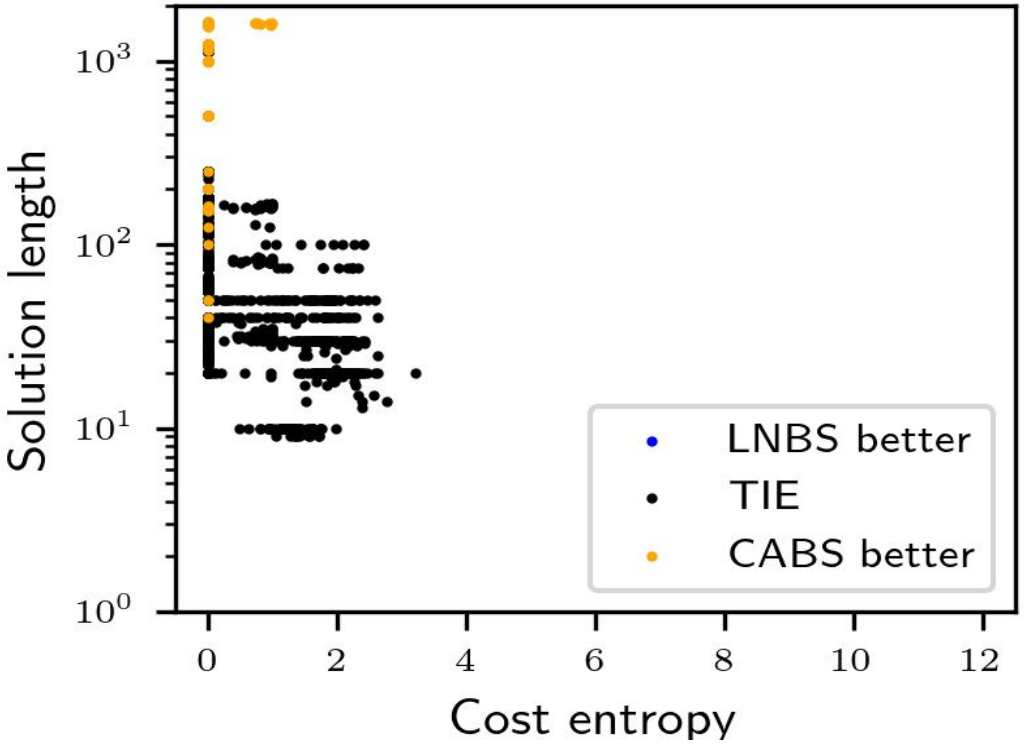
Why LNBS is Worse in Some Problems?

- Hypothesis: when partial path costs are not diverse (low entropy), finding a better solution in a partial state space is difficult
- Not much difference if a problem is easy (the solution length is small)

Routing and scheduling problems



Other problems

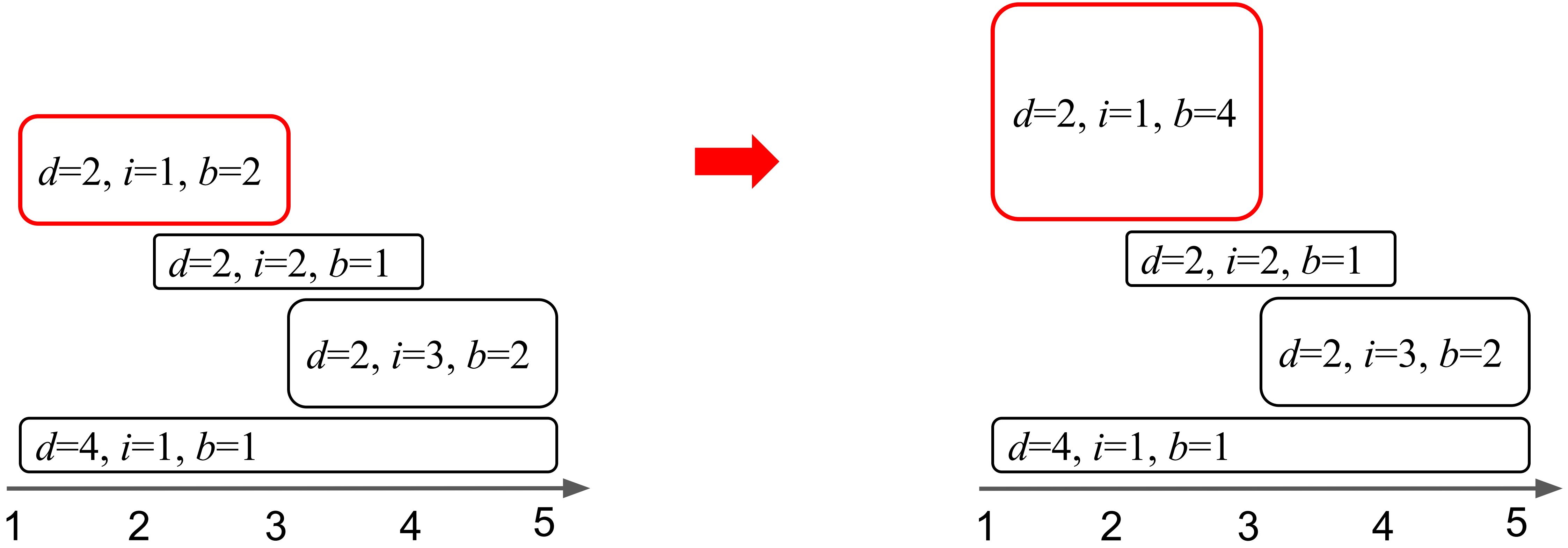


Conclusion

- DIDP: a model & solve paradigm based on DP
- LNBS is effective particularly in routing and scheduling problems such as TSPTW, m-PDTSP, and talent scheduling, which seems to be related to the diversity of partial path costs
- Start DIDP with Python: `pip install didppy`
  Tutorials and API Reference: https://didppy.rtfd.io

Beam Width Selection

Double beam width b_{di} for length d and start i after each beam search starting from $b_{di}=1$



Definition of Entropy

$$H(Y) = \sum_{c \in C} \frac{|\{y \in Y \mid \mathsf{cost}_y = c\}|}{|Y|} \log_2 \frac{|\{y \in Y \mid \mathsf{cost}_y = c\}|}{|Y|}$$

Y : set of partial paths
C : set of partial path costs

We enumerate all feasible prefixes for an initial solution where first eight transitions are removed