Domain-Independent Dynamic Programming for Combinatorial Optimization

J. Christopher Beck & Ryo Kuroiwa

Department of Mechanical & Industrial Engineering

University of Toronto

jcb@mie.utoronto.ca

UNIVERSITY OF TORONTO

Engineering

DPSOLVE Workshop

Toronto, August 27, 2023

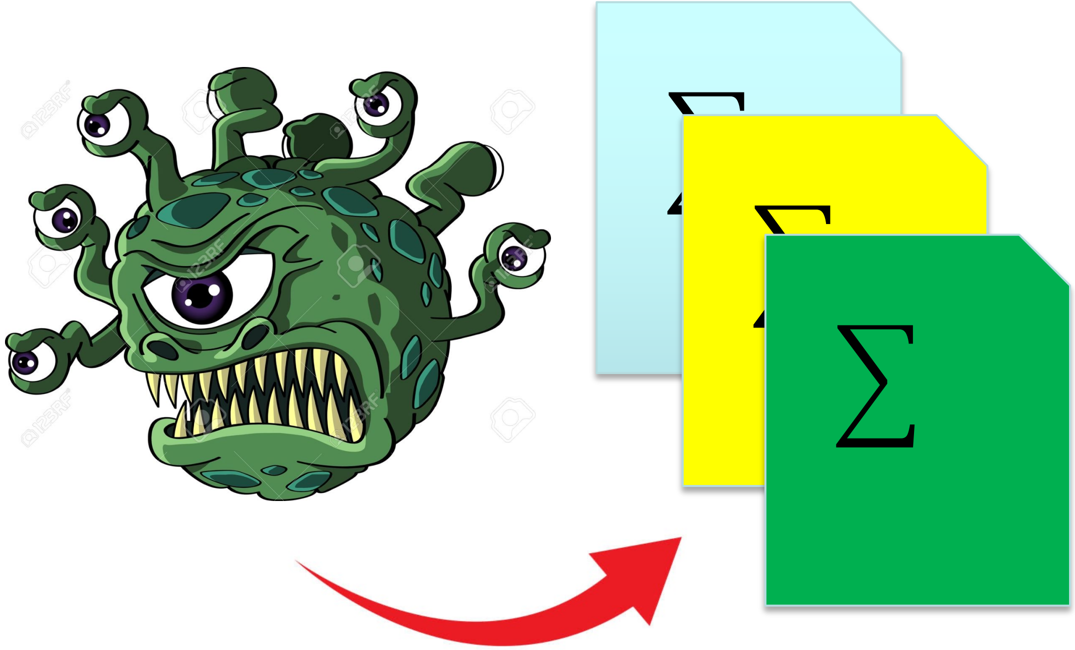
This is not a talk about Decision Diagrams

What it is about

1. A language to model combinatorial optimization problems as dynamic programs
2. A solver that solves such problems using heuristic search

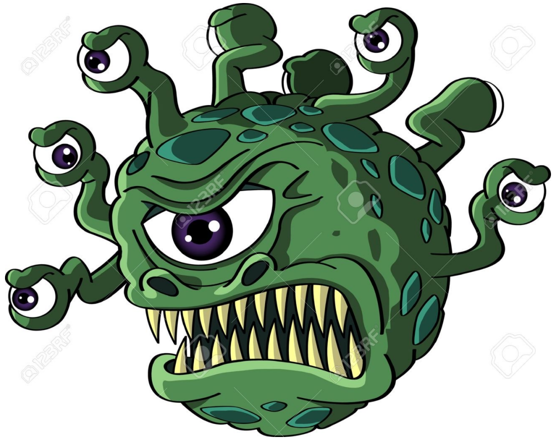
Model-and-Solve

Problem Problem Definition

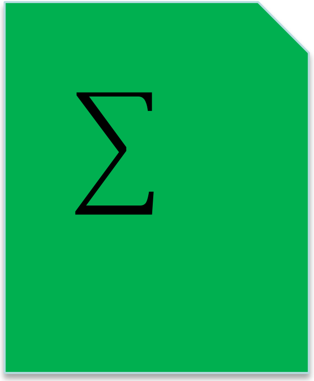


Model-and-Solve

Problem



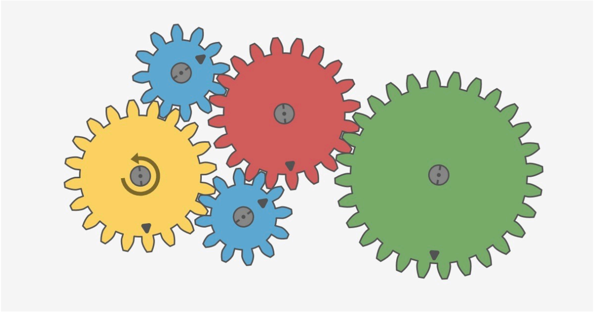
Problem Definition



Models

$$
\begin{array}{lll}
\min & m & \\
\text{s.t.} & \text{Pack}(\{y_j \mid j \in M\}, \{x_i \mid i \in N\}, \{t_i \mid i \in N\}) & \\
& 0 \leq y_j \leq c & \forall j \in M \\
& e_i - 1 \leq x_i \leq m - 1 - l_i & \forall i \in N \\
& x_i + d_{ij} \leq x_j \quad \forall j \in N, \forall i \in \tilde{P}_j, & \\
& \qquad \nexists k \in \tilde{S}_i \cap \tilde{P}_j : d_{ij} \leq d_{ik} + d_{kj} & \\
& m \in \mathbb{Z} & \\
& y_j \in \mathbb{Z} & \forall j \in M \\
& x_i \in \mathbb{Z} & \forall i \in N.
\end{array}
$$

General Purpose Solver



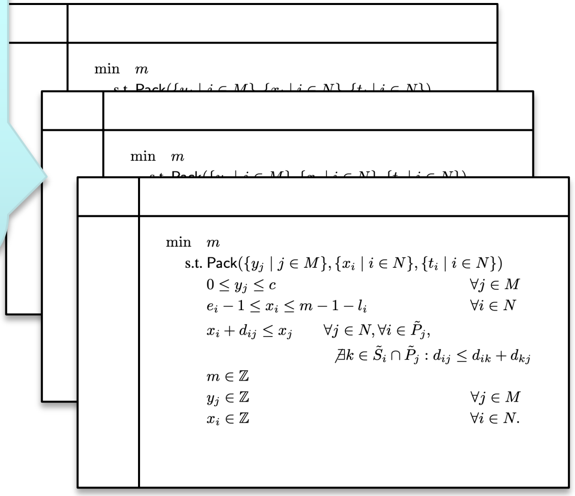
A Solution!

CP, LP, MIP, MINLP,
AI Planning, …

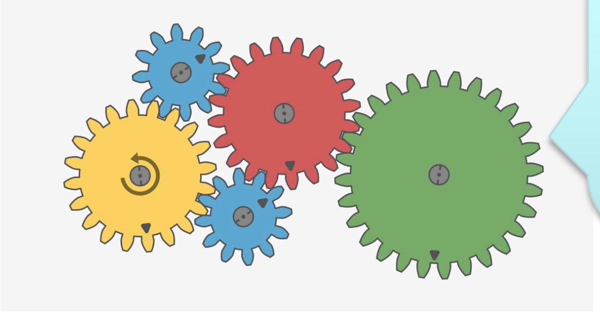
Model-and-Solve for DP

- Domain-independent dynamic programming (DIDP)

Define models using DP transition system



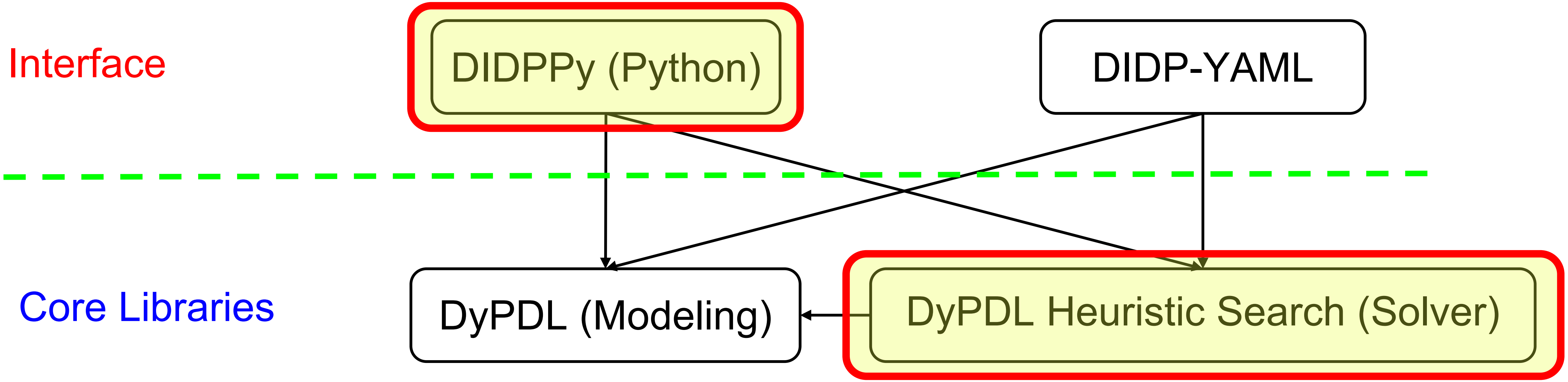
$$
\begin{aligned}
\min \quad & m \\
\text{s.t.} \quad & \text{Pack}(\{y_j \mid j \in M\}, \{x_i \mid i \in N\}, \{t_i \mid i \in N\}) \\
& 0 \leq y_j \leq c \quad && \forall j \in M \\
& e_i - 1 \leq x_i \leq m - 1 - l_i \quad && \forall i \in N \\
& x_i + d_{ij} \leq x_j \quad && \forall j \in N, \forall i \in \tilde{P}_j, \\
& && \nexists k \in \tilde{S}_i \cap \tilde{P}_j : d_{ij} \leq d_{ik} + d_{kj} \\
& m \in \mathbb{Z} \\
& y_j \in \mathbb{Z} \quad && \forall j \in M \\
& x_i \in \mathbb{Z} \quad && \forall i \in N.
\end{aligned}
$$



Solve models using heuristic state-based search

Open Source Software: didp-rs

https://github.com/domain-independent-dp/didp-rs



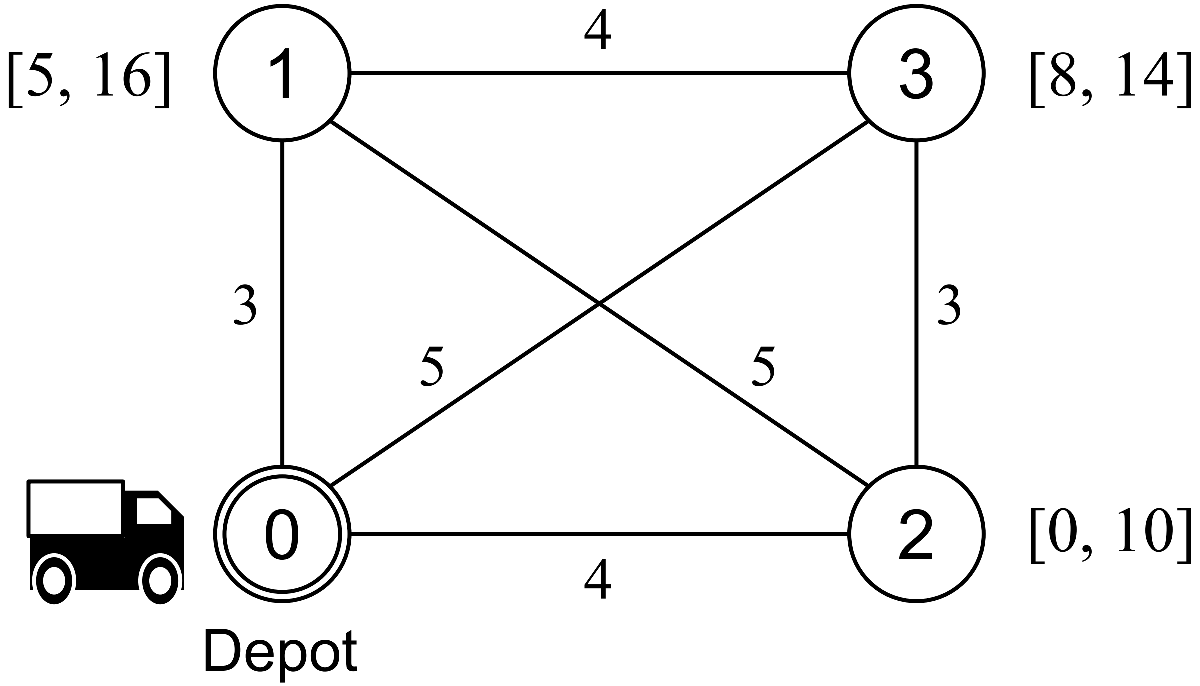
Implemented in Rust

Outline

1. Background
2. Our Modeling Interface: DIDPPy
3. Solving DIDP
4. Anytime DIDP Solvers
5. Ongoing & Future Work

Combinatorial Optimization

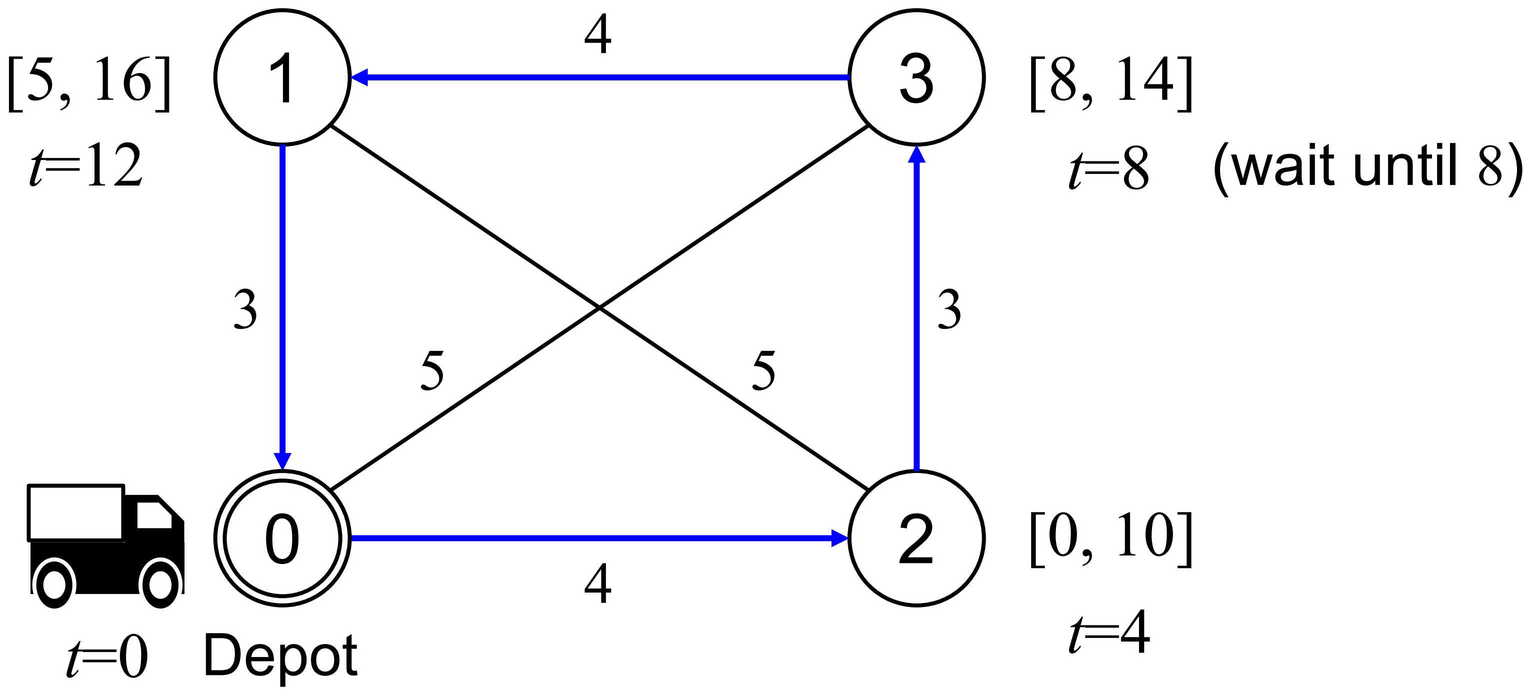
Traveling Salesperson Problem with Time Windows (TSPTW)
Minimize the travel time to visit all customers within time windows



Combinatorial Optimization

Traveling Salesperson Problem with Time Windows (TSPTW)
Minimize the travel time to visit all customers within time windows

Total cost: 14



DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

$$\text{compute } V(N \setminus \{0\}, 0, 0)$$

$$V(U,i,t) = \begin{cases} \min\limits_{j \in U: t + c_{ij} \le b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) & \text{else if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N: all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

compute $V(N \setminus \{0\}, 0, 0)$

$$V(U,i,t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) & \text{else if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N: all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

$\boxed{\text{compute } V(N \setminus \{0\}, 0, 0)}$

$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \le b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) & \text{else if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N : all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

compute $V(N \setminus \{0\}, 0, 0)$

$$V(U,i,t) = \begin{cases} \boxed{\displaystyle\min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) \quad \text{if } U \neq \emptyset} & \text{Visit a customer} \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) \quad \text{else if } i \neq 0 & \text{Return to the depot} \\ 0 \quad \text{otherwise} & \text{Base case} \end{cases}$$

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N: all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

$$\text{compute } V(N \setminus \{0\}, 0, 0)$$

$$V(U,i,t) = \begin{cases} \min\limits_{\boxed{j \in U : t + c_{ij} \le b_j}} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) & \text{else if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N : all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

$$\text{compute } V(N \setminus \{0\}, 0, 0)$$

$$V(U,i,t) = \begin{cases} \min\limits_{j \in U: t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ \boxed{c_{i0} + V(\emptyset, 0, t + c_{i0}) \quad \text{else if } i \neq 0} \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N: all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

$$\text{compute } V(N \setminus \{0\}, 0, 0)$$

$$V(U,i,t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \le b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) & \text{else if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N: all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP for Combinatorial Optimization

Recursive equations for the value function of a state (subproblem)

compute $V(N \setminus \{0\}, 0, 0)$

$$V(U, i, t) = \begin{cases} \min_{j \in U : t + c_{ij} \le b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{if } U \neq \emptyset \\ c_{i0} + V(\emptyset, 0, t + c_{i0}) & \text{else if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Visit a customer

Return to the depot

Base case

State variables:

- U : unvisited customers
- i : current customer
- t : current time

Constants

- N: all customers (0: depot)
- $[a_i, b_i]$: time window for customer i
- c_{ij} : travel time from customer i to j

DP usually solved by problem-specific algorithm implementations

Our Modeling Interface: DIDPPy

Constants and State Variables

```python
import didppy as dp

model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

Constants and State Variables

```python
import didppy as dp
```
Module

```python
model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

Constants and State Variables

```
import didppy as dp

model = dp.Model(maximize=False)
```

Model (minimization)

```
customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

Constants and State Variables

```python
import didppy as dp

model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

Constants

Customers $N = \{0, 1, 2, 3\}$

Ready time a_i

Deadline b_i

Travel time c_{ij}

Constants and State Variables

```
import didppy as dp

model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

To use state variable i for indexing

Constants and State Variables

```
import didppy as dp

model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

State variables

Unvisited $U \subseteq N$

Current $i \in N$

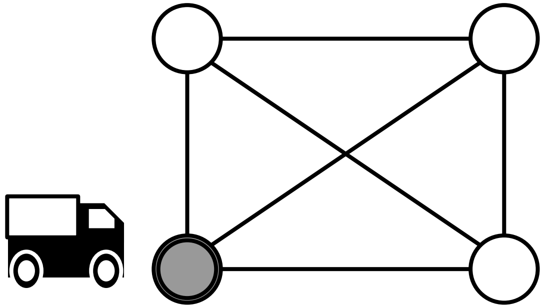
Time $t \in \mathbb{Z}$

Constants and State Variables

```python
import didppy as dp

model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)
a = [0, 5, 0, 8]
b = [100, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])

u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```

Target state

compute $V(N \setminus \{0\}, 0, 0)$



Questions?

Recursive Equation as Transitions

```
for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j), (t, dp.max(t + c[i, j], a[j]))],
        preconditions=[u.contains(j), t + c[i, j] <= b[j]],
    )
    model.add_transition(visit)
```

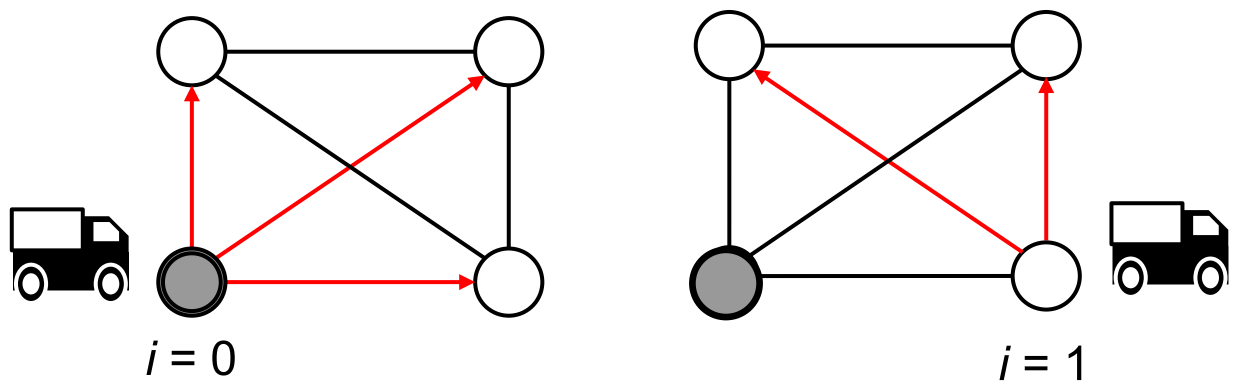
$$V(U, i, t) = \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\})$$

Recursive Equation as Transitions

```
for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j), (t, dp.max(t + c[i, j], a[j]))],
        preconditions=[u.contains(j), t + c[i, j] <= b[j]],
    )
    model.add_transition(visit)
```

$$V(U, i, t) = \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\})$$

How to compute V



… for each value of $i \in N$

Recursive Equation as Transitions

```
for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j), (t, dp.max(t + c[i, j], a[j]))],
        preconditions=[u.contains(j), t + c[i, j] <= b[j]],
    )
    model.add_transition(visit)
```

$$V(U,i,t) = \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\})$$

How to compute the next state

Recursive Equation as Transitions

```
for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j), (t, dp.max(t + c[i, j], a[j]))],
        preconditions=[u.contains(j), t + c[i, j] <= b[j]],
    )
    model.add_transition(visit)
```

$$V(U, i, t) = \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\})$$

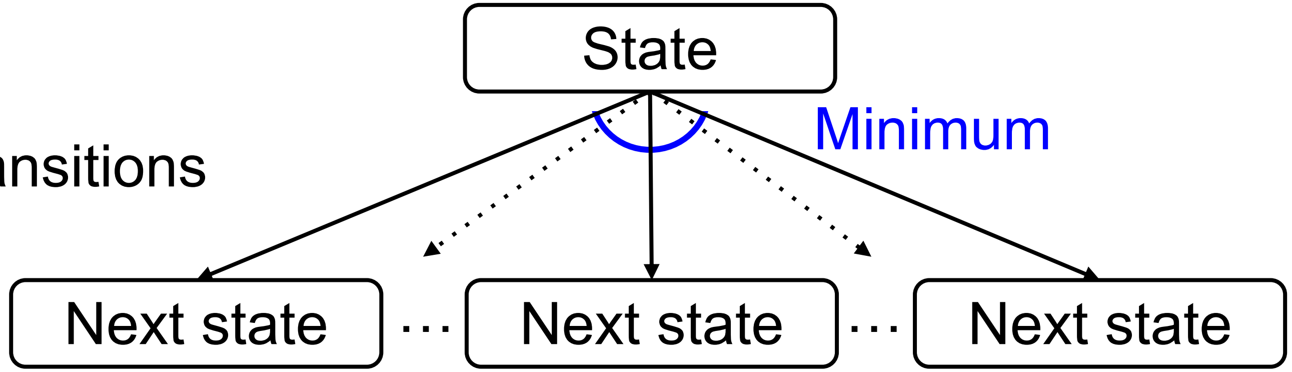
When the transition is applicable

Recursive Equation as Transitions

```python
for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j), (t, dp.max(t + c[i, j], a[j]))],
        preconditions=[u.contains(j), t + c[i, j] <= b[j]],
    )
    model.add_transition(visit)
```

$$V(U, i, t) = \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\})$$

Applicable transitions

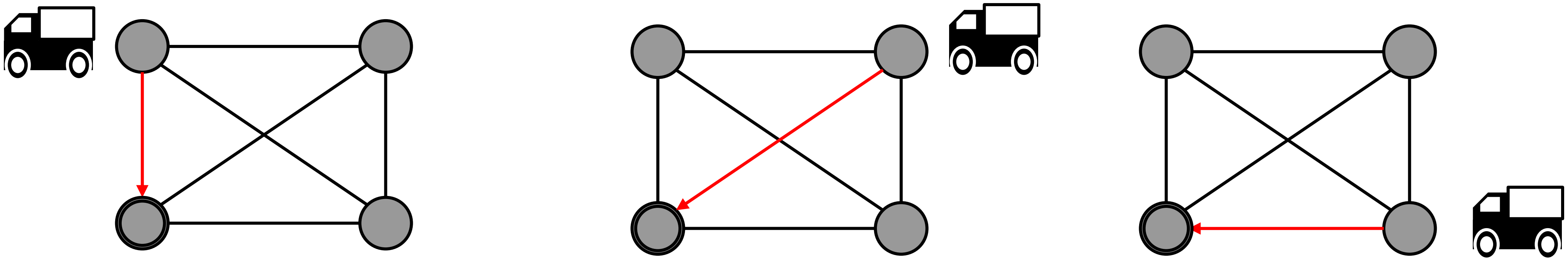


Value of the current state: minimum cost over all applicable transitions
(infinity if no applicable transitions)

Recursive Equation as Transitions

```
return_to_depot = dp.Transition(
    name="return",
    cost=c[i, 0] + dp.IntExpr.state_cost(),
    effects=[(i, 0), (t, t + c[i, 0])],
    preconditions=[u.is_empty(), i != 0],
)
model.add_transition(return_to_depot)
```

$$V(U, i, t) = c_{i0} + V(\emptyset, 0, t + c_{i0}) \quad \text{if } U = \emptyset \wedge i \neq 0$$

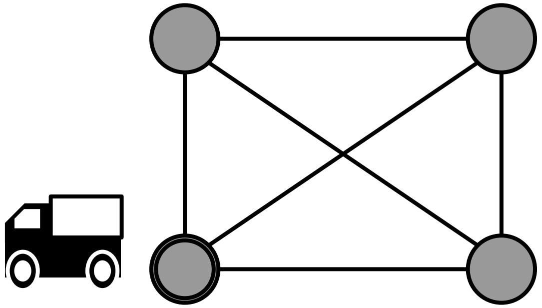


Base Cases: When to Stop Recursion

```
model.add_base_case([u.is_empty(), i == 0], cost=0)
```

$V(U, i, t) = 0 \quad \text{if } U = \emptyset \land i = 0$

End of recursion on V



Better Model with Redundant Information

Explicitly modeling implications of the problem definition (similar to redundant constraints in MIP)

Dominance based on **resource variables** $V(U, i, t) \leq V(U, i, t')$ if $t \leq t'$

```
t = model.add_int_resource_var(target=0, less_is_better=True)
```

Dual bound function (LB in minimization) $V(U, i, t) \geq 0$

```
model.add_dual_bound(0)
```

A dual bound is defined for a state

Other features not detailed here:

- State constraints: conditions that a state must satisfy
- Forced transitions: sometimes transition can be inferred

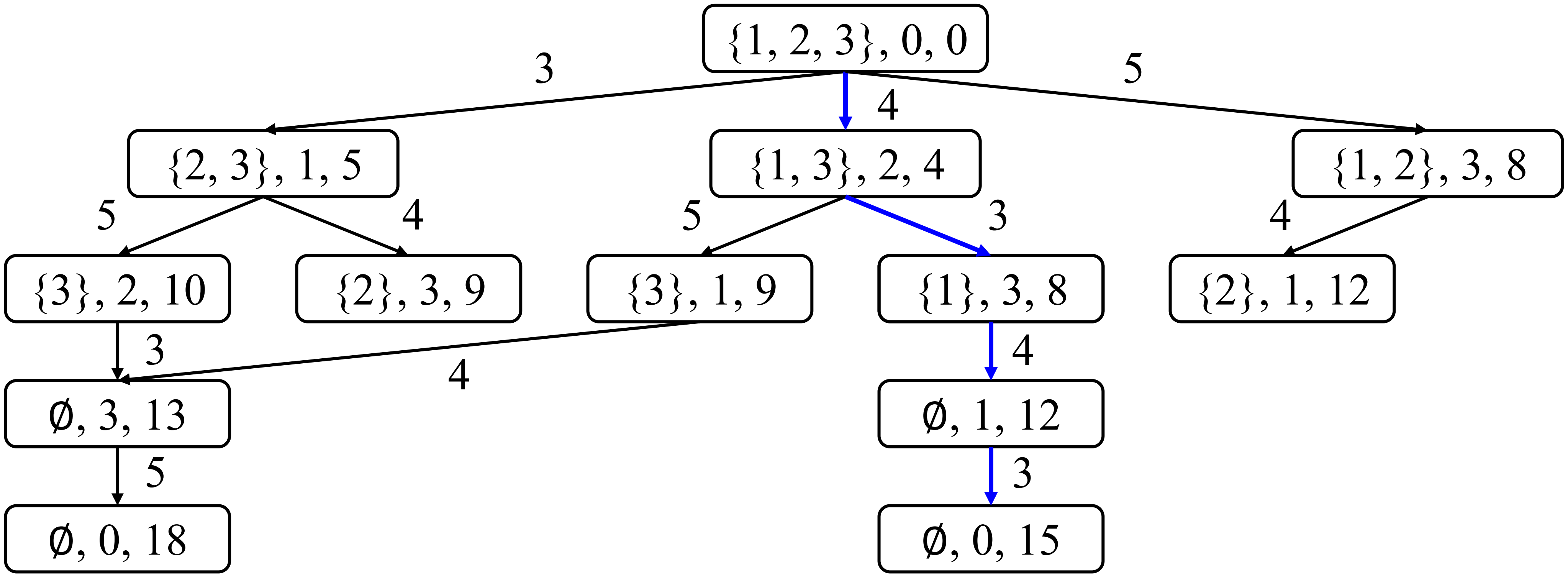
Solving DIDP

Solving

```python
solver = dp.CABS(model)
solution = solver.search()

if solution.is_optimal:
    print("Optimal cost: {}".format(solution.cost))
elif solution.is_feasible:
    print("Infeasible")
else:
    print("Best cost: {}".format(solution.cost))
    print("Best bound: {}".format(solution.best_bound))

print("Solution:")

for transition in solution.transitions:
    print(transition.name)
```

DP as a Shortest Path Problem

- Optimal solution: the shortest path in a state space graph
- Nodes: states, edges: transitions, weights: travel times



CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)

$\boxed{\{1, 2, 3\}, 0, 0}$

CAASDy: Prototype Solver for DIDP

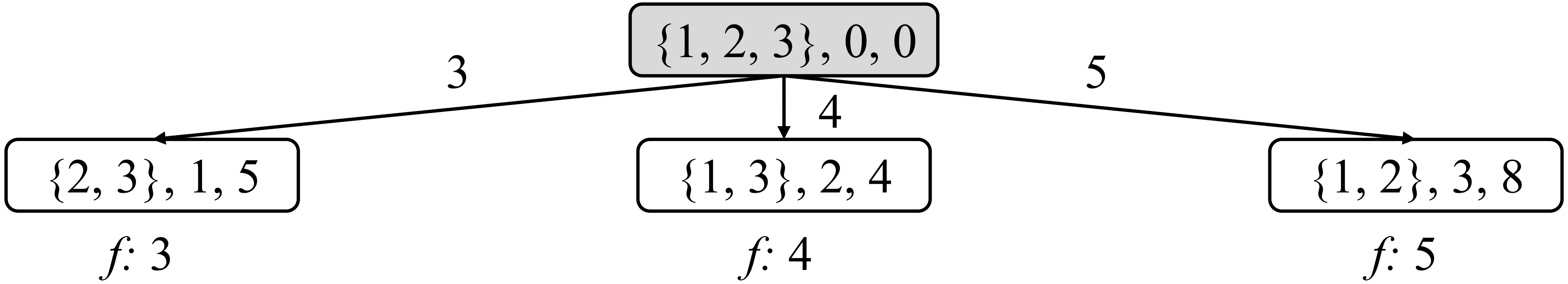
- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)

$$\boxed{\{1, 2, 3\}, 0, 0}$$

$V(U, i, t) \geq 0$ $\quad$ f: 0

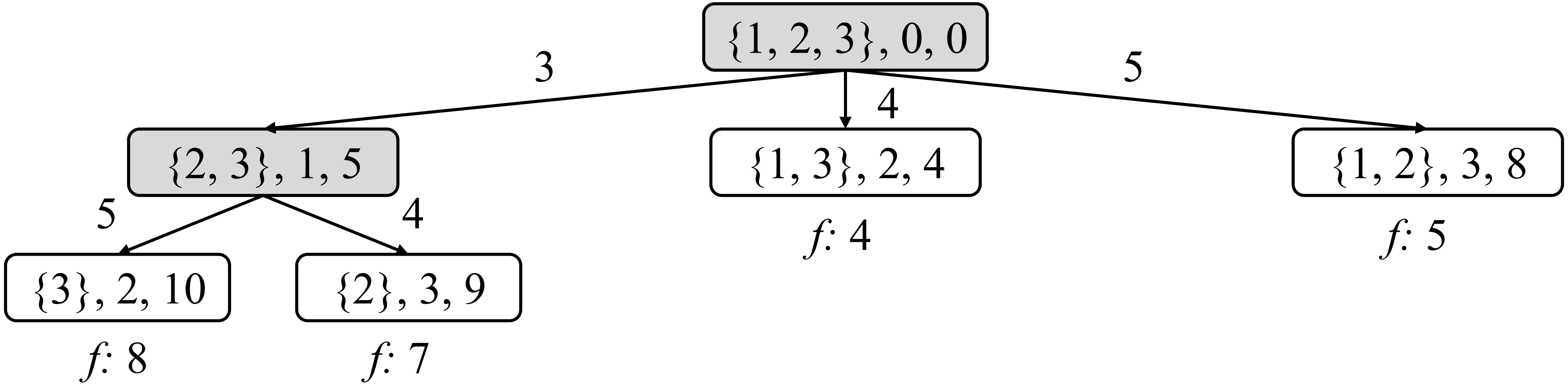
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



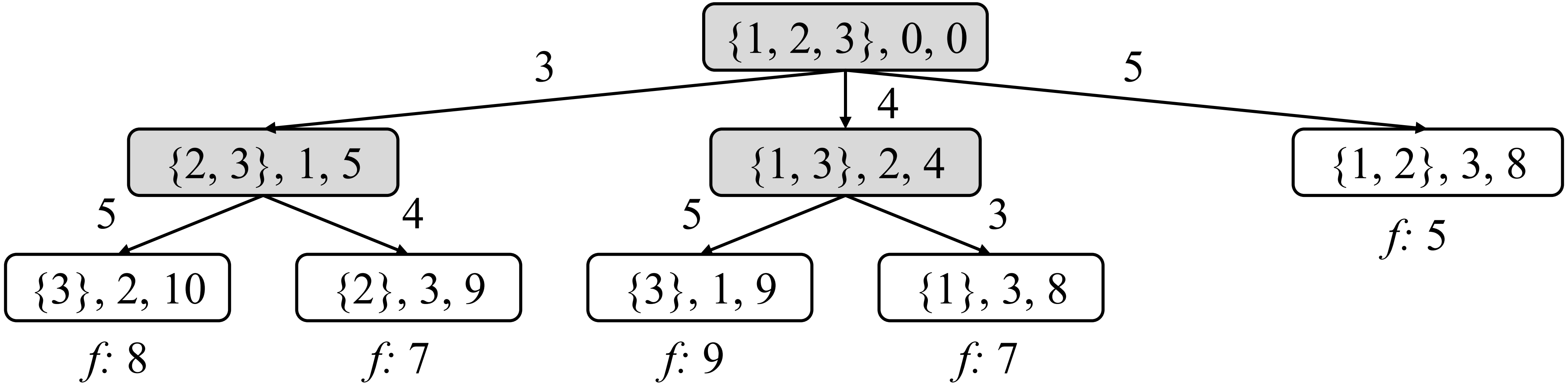
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



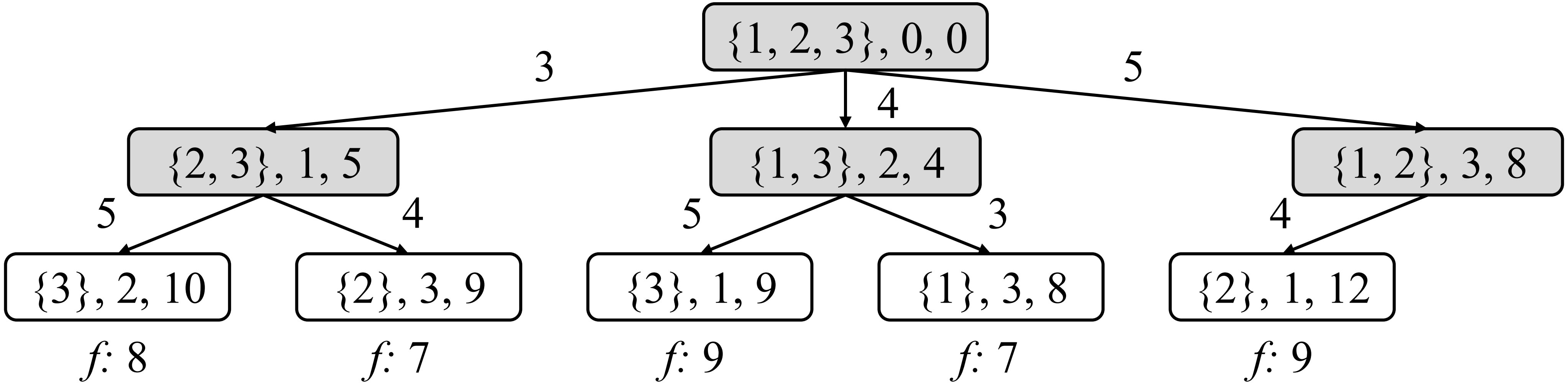
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



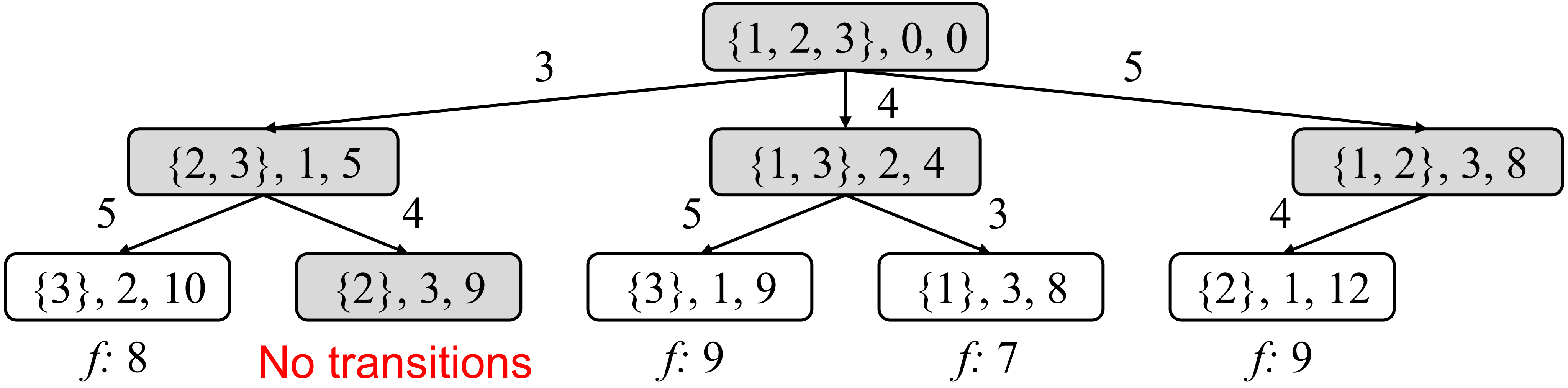
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of *f* (path cost + dual bound of a state)



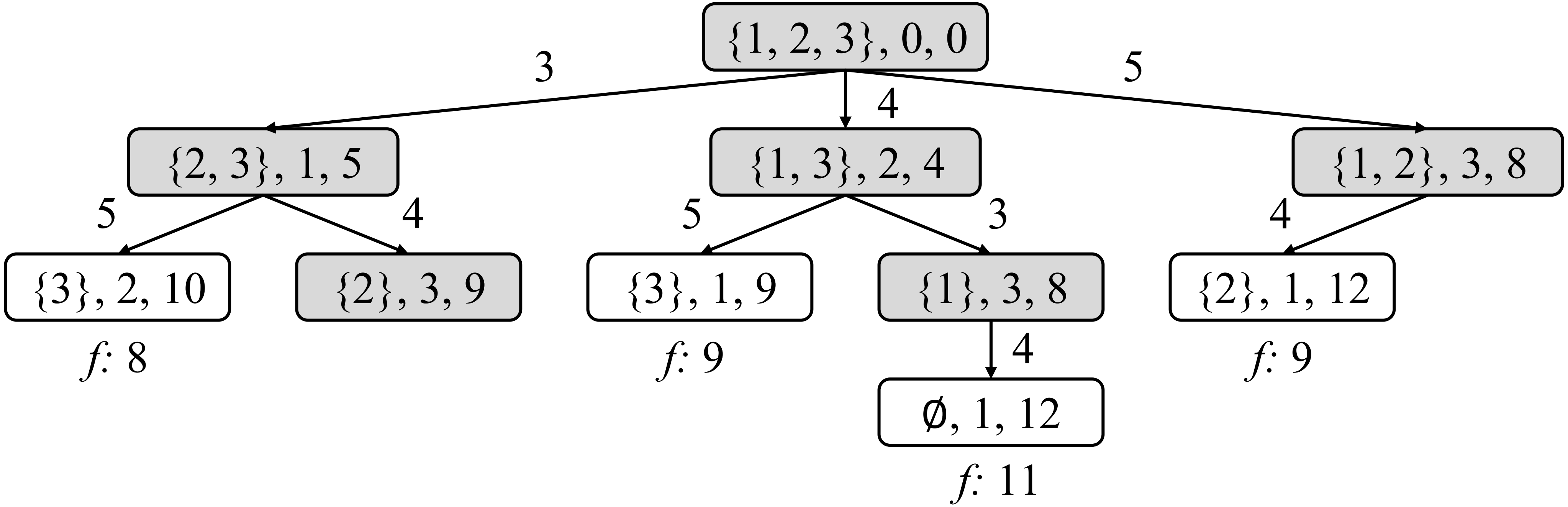
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



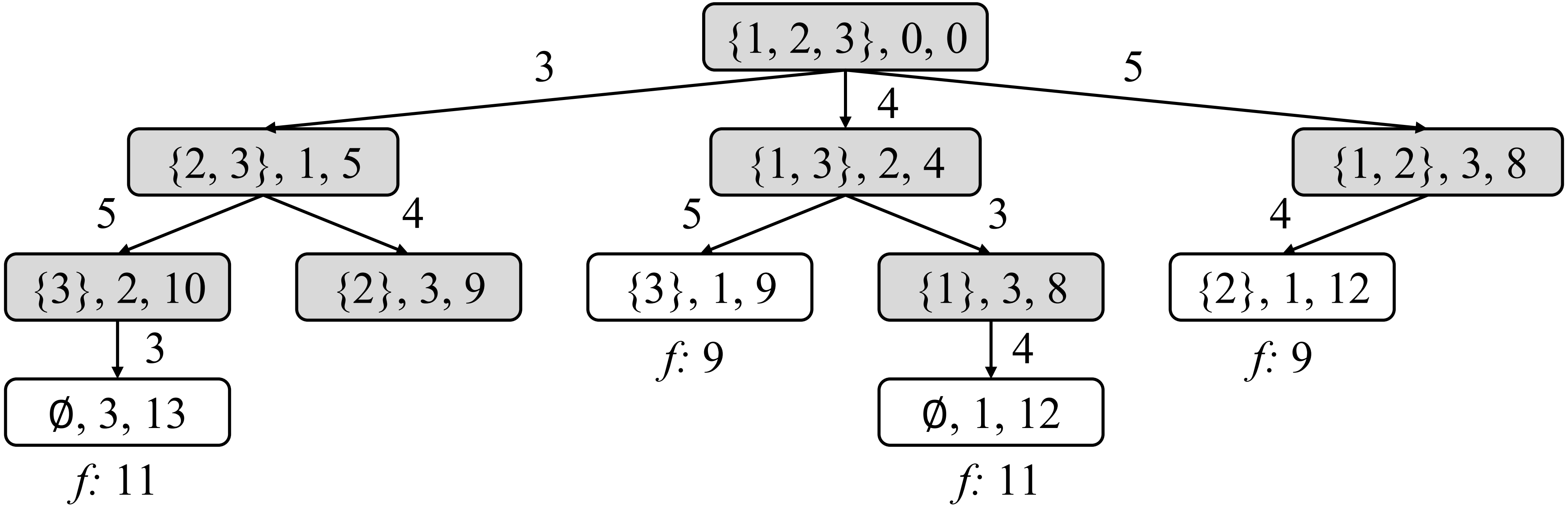
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of *f* (path cost + dual bound of a state)



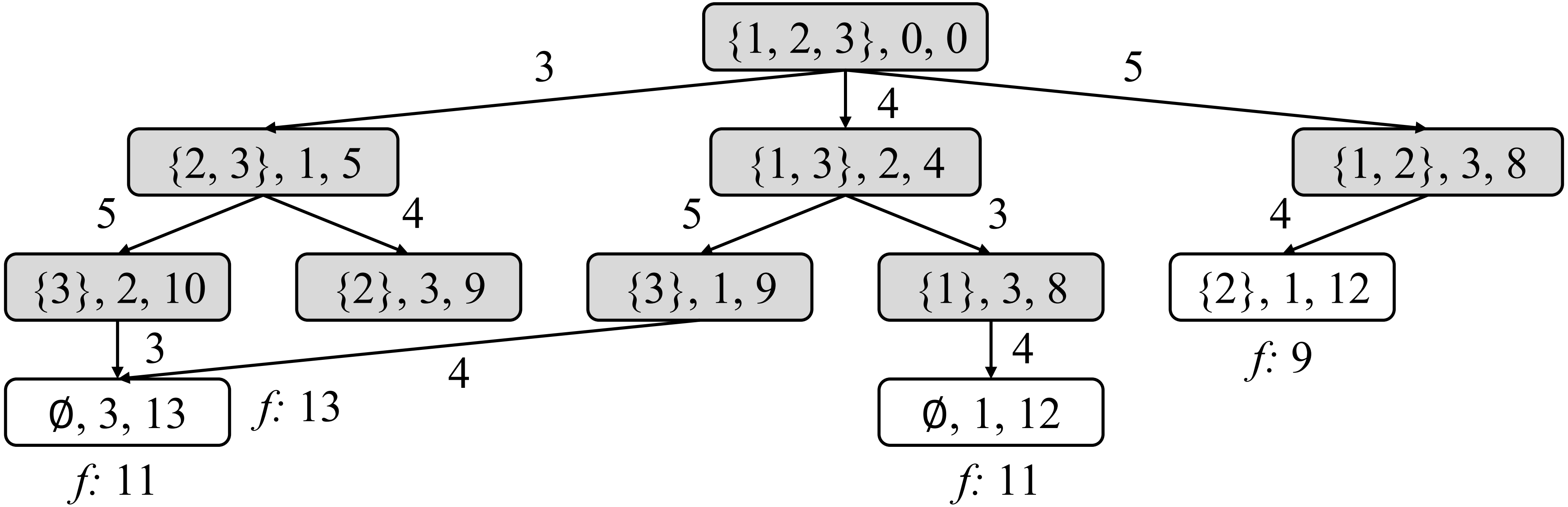
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of *f* (path cost + dual bound of a state)



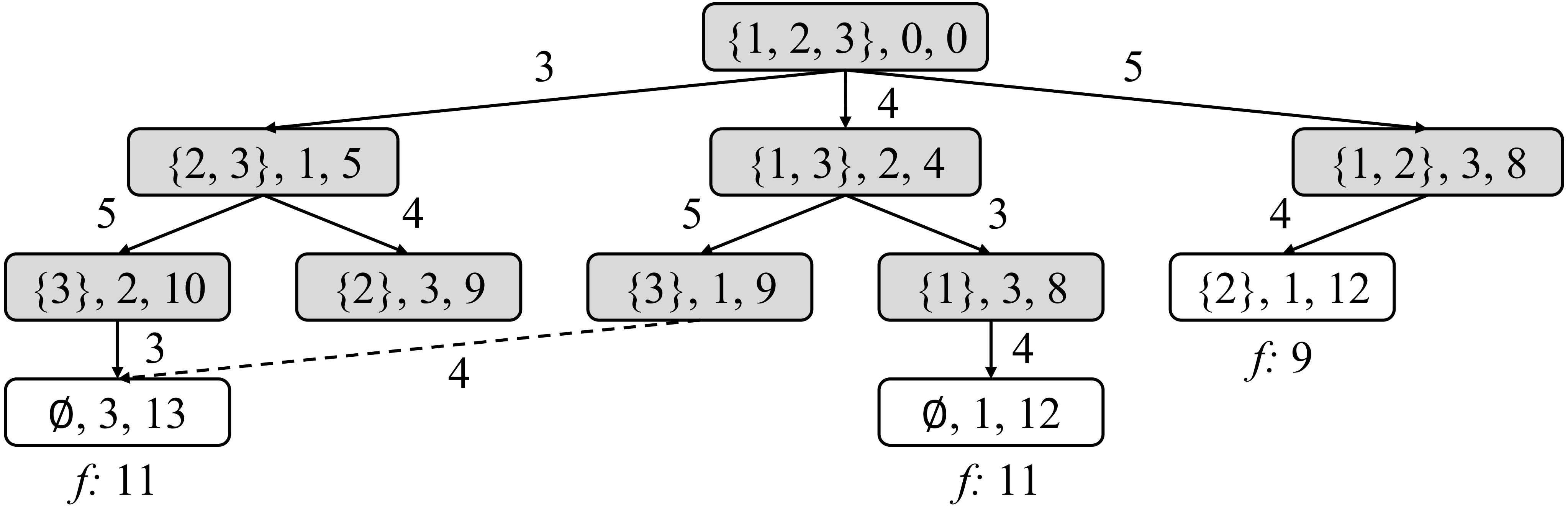
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



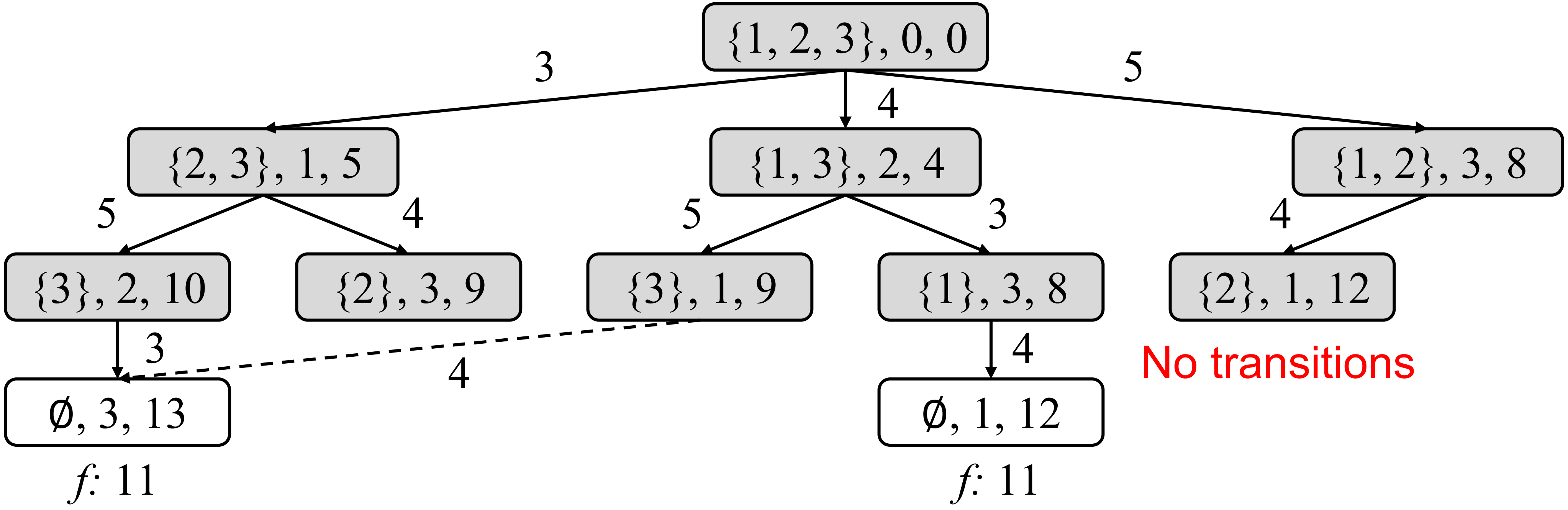
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of *f* (path cost + dual bound of a state)



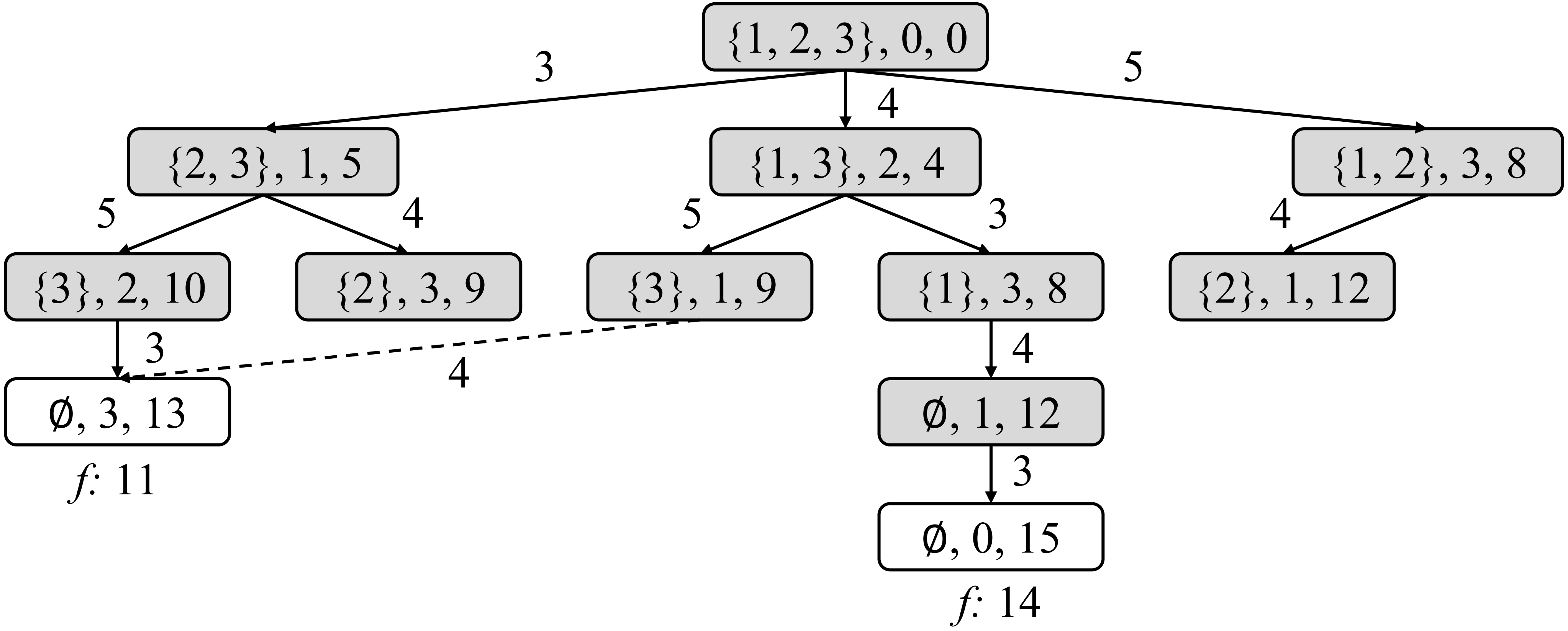
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



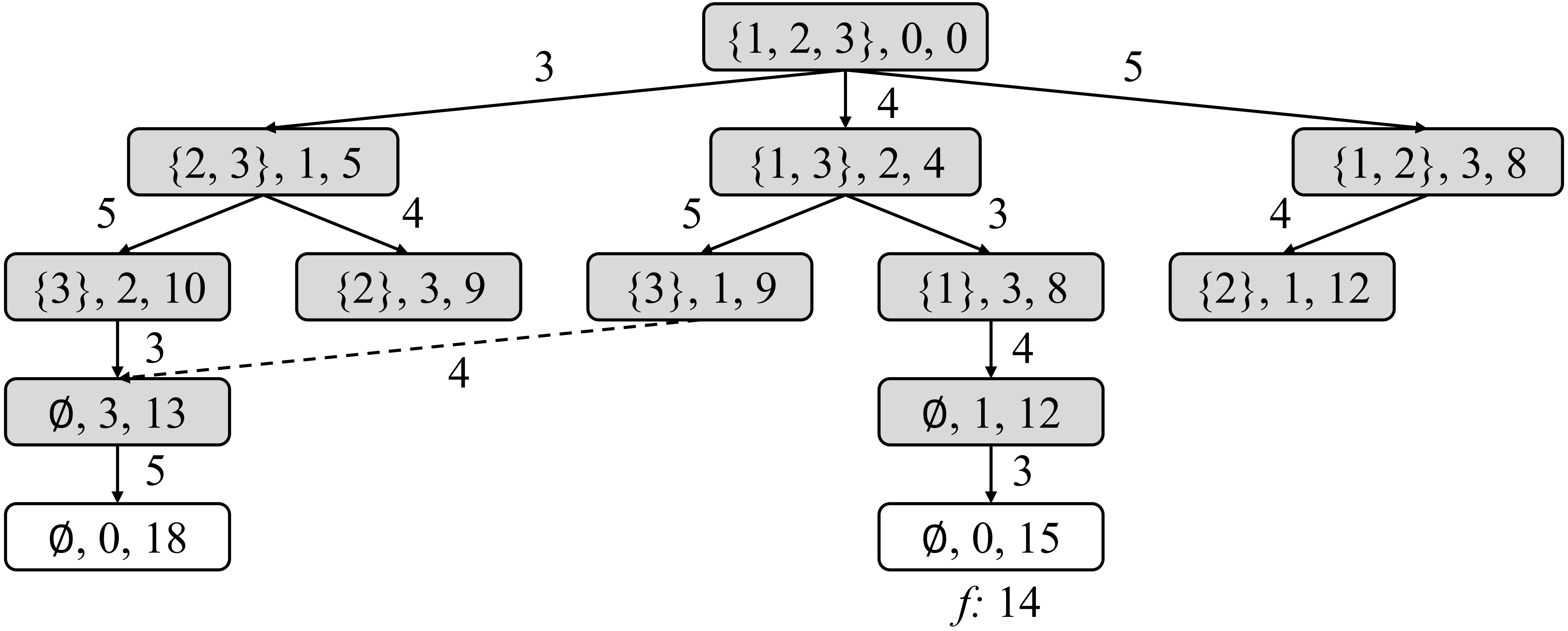
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of *f* (path cost + dual bound of a state)



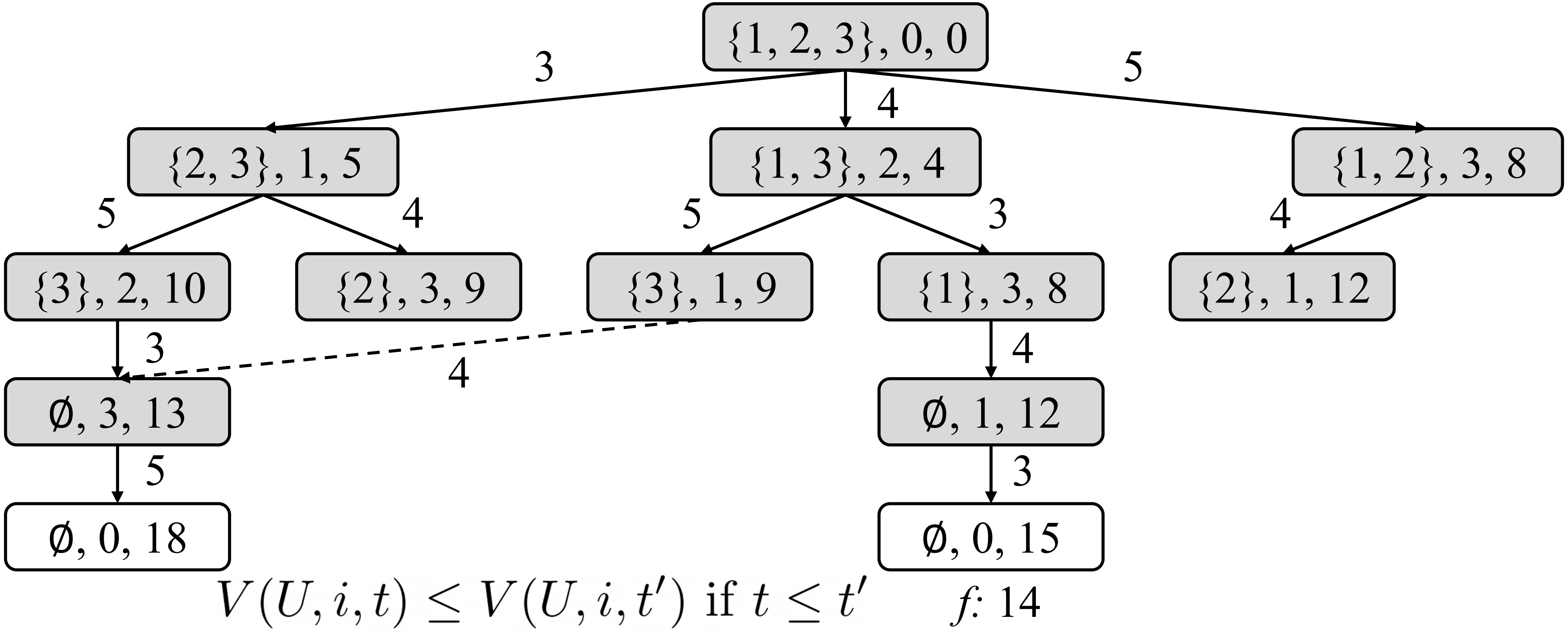
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



CAASDy: Prototype Solver for DIDP

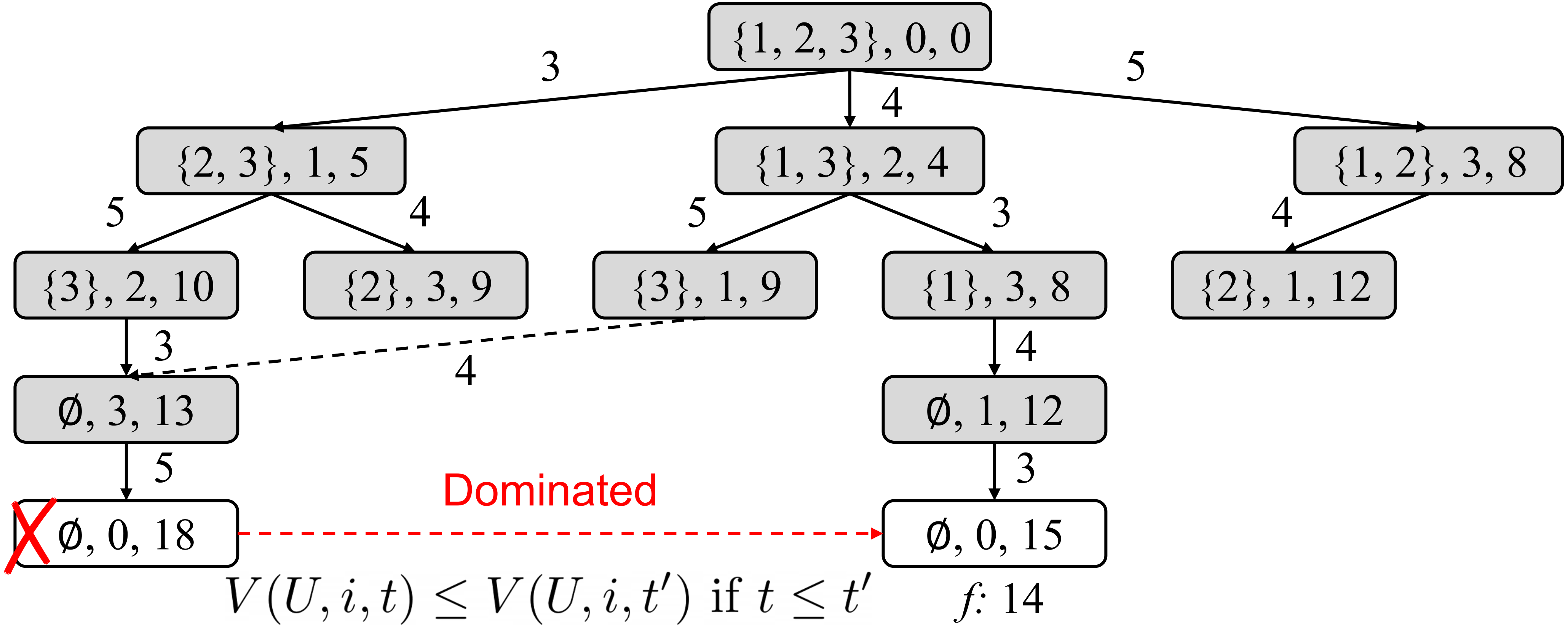
- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



$V(U, i, t) \leq V(U, i, t')$ if $t \leq t'$

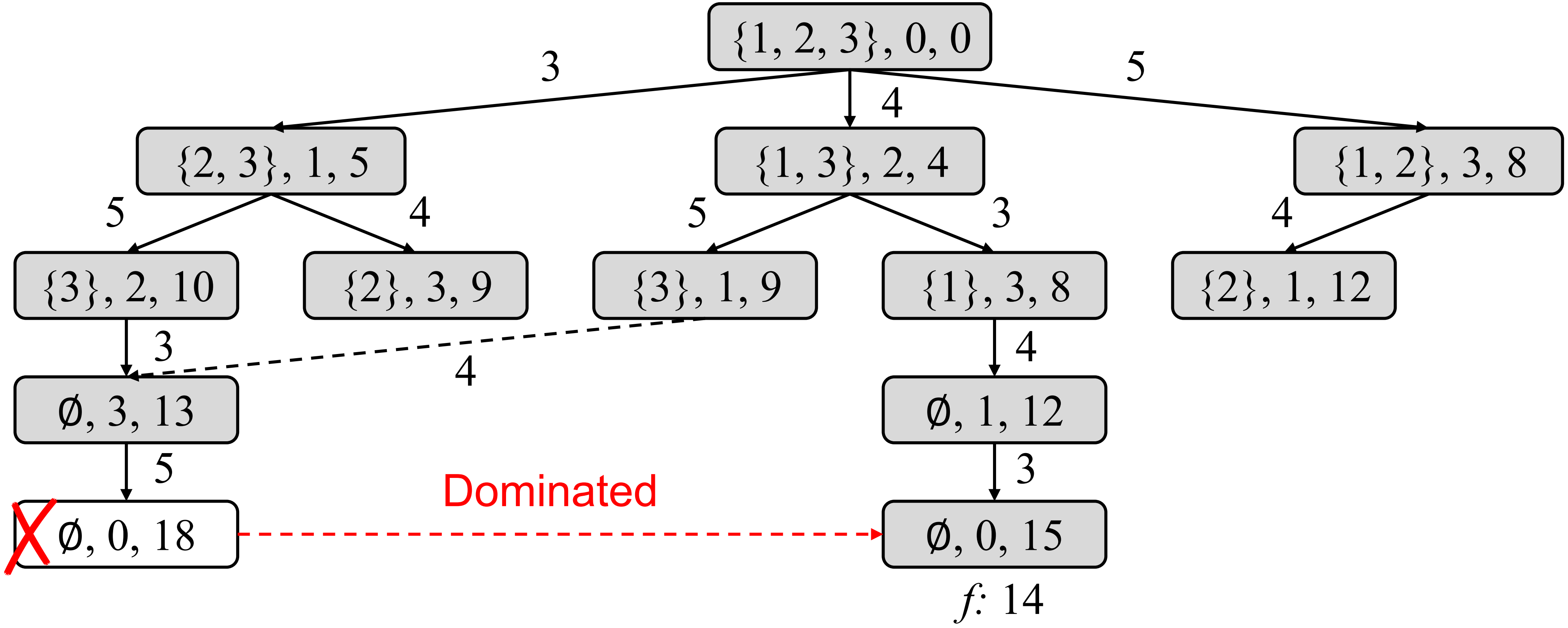
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



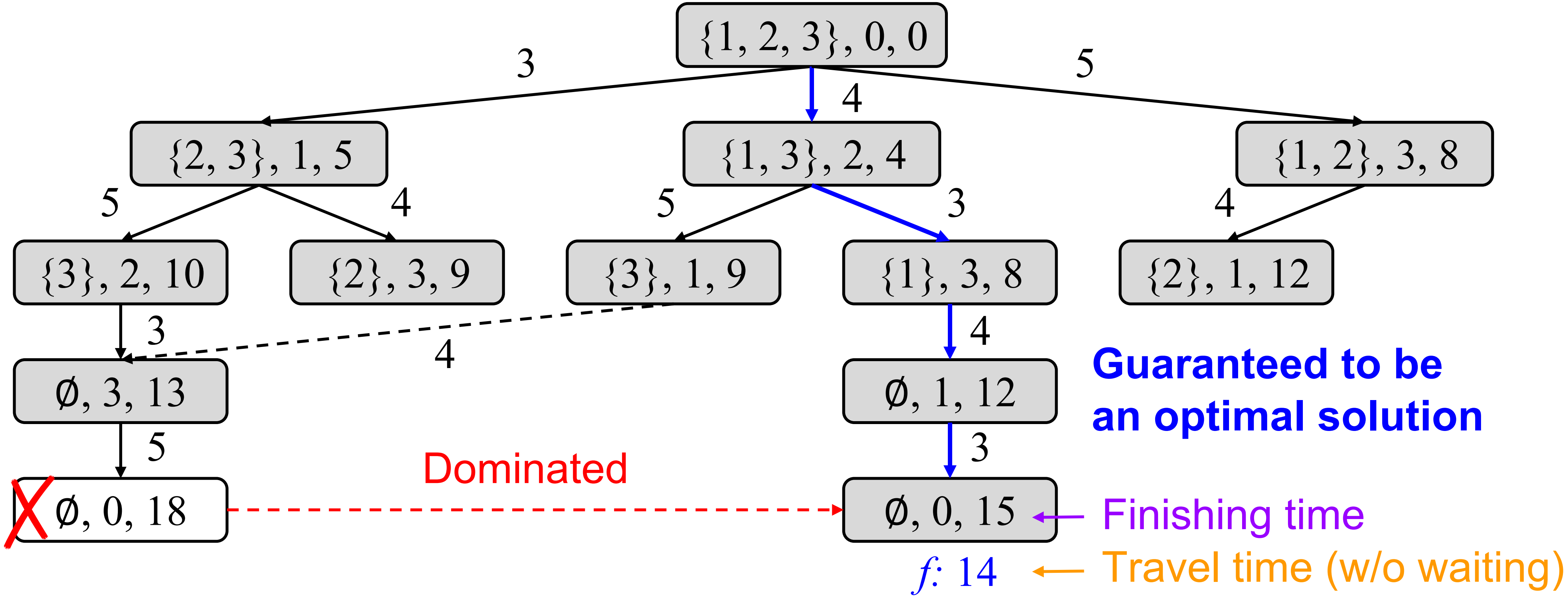
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of f (path cost + dual bound of a state)



Comparison of MIP, CP, and DIDP

Problem	Description	MIP (Gurobi)	CP (CP Optimizer)	DIDP
TSPTW (340)	TSP with time	227	47	**257**
CVRP (207)	vehicle routing	**26**	0	5
SALBP-1 (2100)	assembly line	1357	1584	**1653**
Bin Packing (1615)	bin packing	1157	**1234**	922
MOSP (570)	manufacturing	225	437	**483**
Graph-Clear (135)	building security	24	4	**76**

# of optimally solved instances with 8 GB and 30 minutes

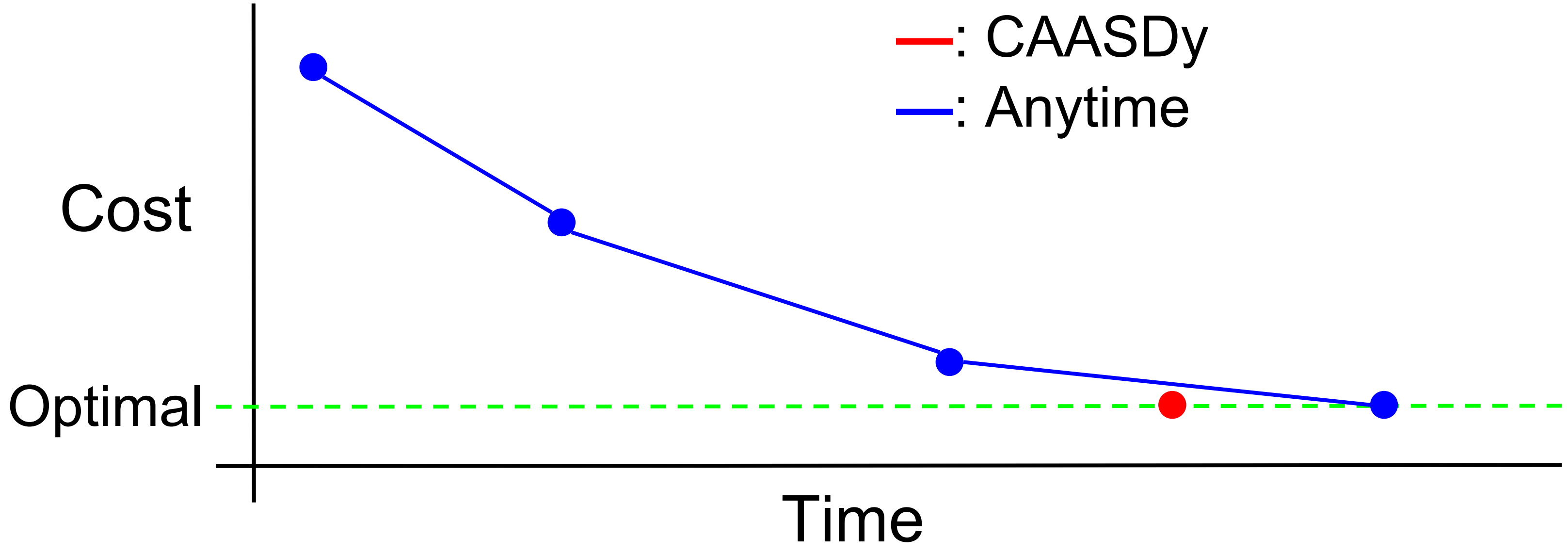
ICAPS 2023 [Kuroiwa and Beck 2023a]

Anytime DIDP Solvers

Anytime Solvers

- Quickly find a solution and continuously improve it
- Standard in OR (e.g., MIP and CP)

Can we develop **anytime solvers for DIDP?**



Anytime State Space Search Algorithms

Algorithm	Description	Reference
Depth First Branch-and-Bound (DFBnB)	DFS	
Cyclic Best-First Search (CBFS)	Hybrid of DFS and best-first search	Kao et al. 2009
Anytime Column Progressive Search (ACPS)	Hybrid of DFS and beam search	Vadlamudi et al. 2012
Anytime Pack Progressive Search (APPS)	Hybrid of DFS and beam search	Vadlamudi et al. 2016
Discrepancy-Bounded DFS (DBDFS)	Discrepancy-based	Beck and Perron 2000
Complete Anytime Beam Search (CABS)	Iterative beam search	Zhang 1998

Implemented in https://github.com/domain-independent-dp/didp-rs

Anytime State Space Search Algorithms

Algorithm	Description	Reference
Depth First Branch-and-Bound (DFBnB)	DFS	
Cyclic Best-First Search (CBFS)	Hybrid of DFS and best-first search	Kao et al. 2009
Anytime Column Progressive Search (ACPS)	Hybrid of DFS and beam search	Vadlamudi et al. 2012
Anytime Pack Progressive Search (APPS)	Hybrid of DFS and beam search	Vadlamudi et al. 2016
Discrepancy-Bounded DFS (DBDFS)	Discrepancy-based	Beck and Perron 2000
Complete Anytime Beam Search (CABS)	Iterative beam search	Zhang 1998

Implemented in https://github.com/domain-independent-dp/didp-rs

Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality

$b = 2$

$\boxed{\{1, 2, 3\}, 0, 0}$

f: 0

Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality

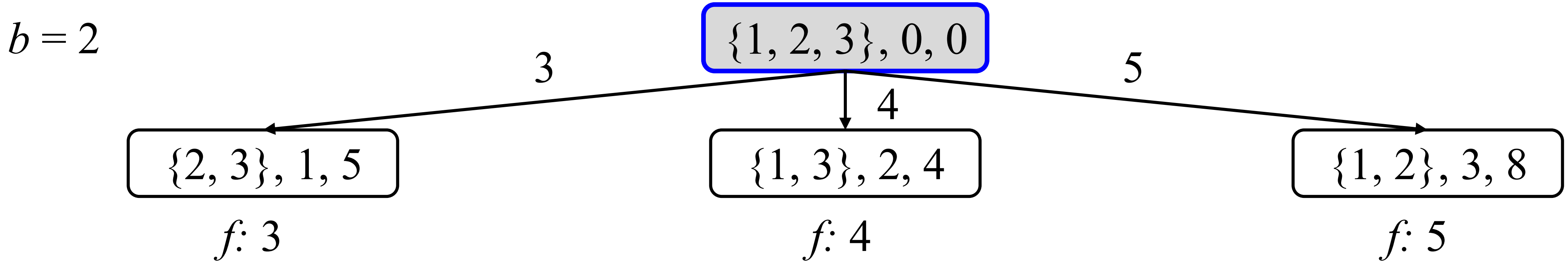
$b = 2$

$$\boxed{\{1, 2, 3\}, 0, 0}$$

f: 0

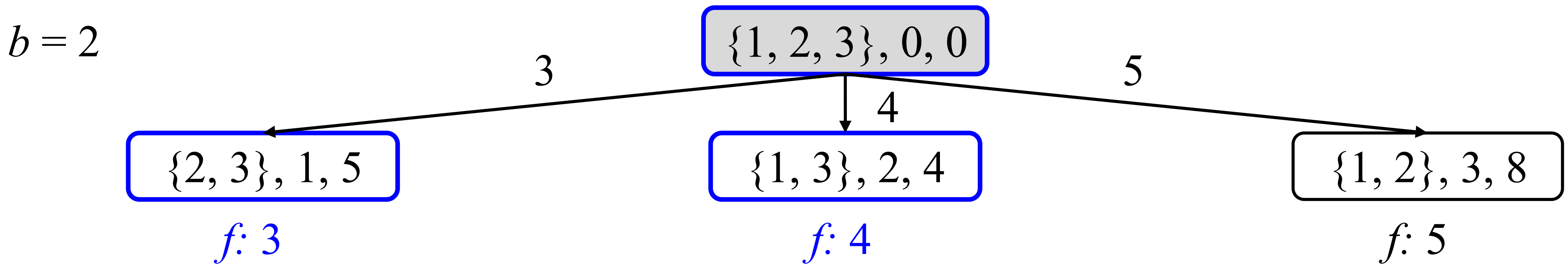
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



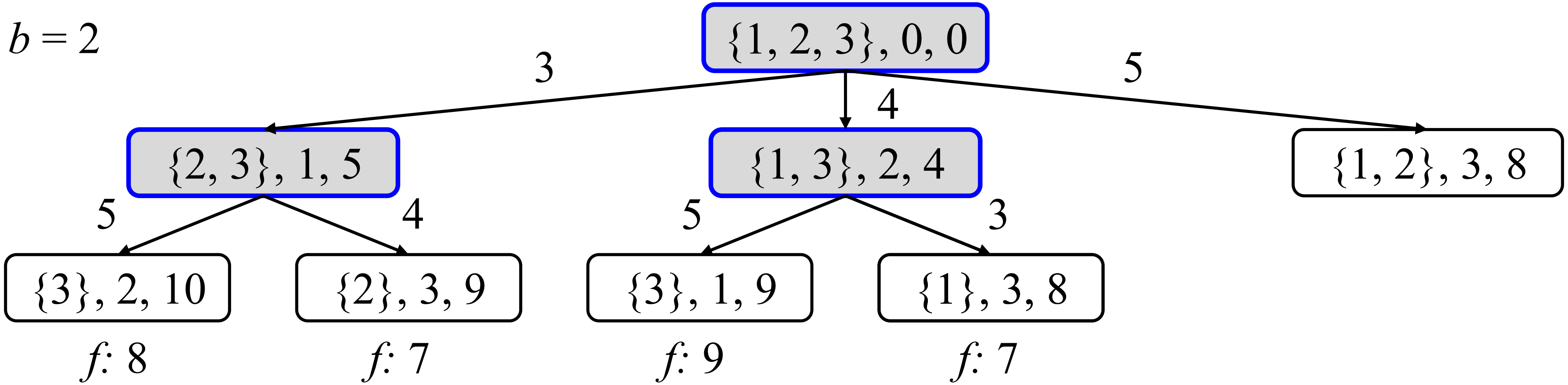
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



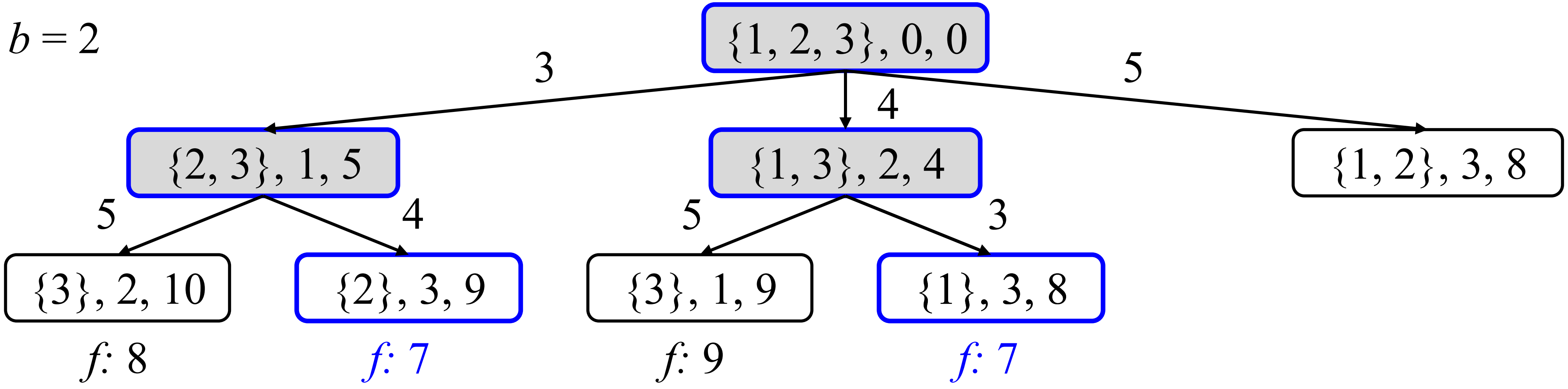
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



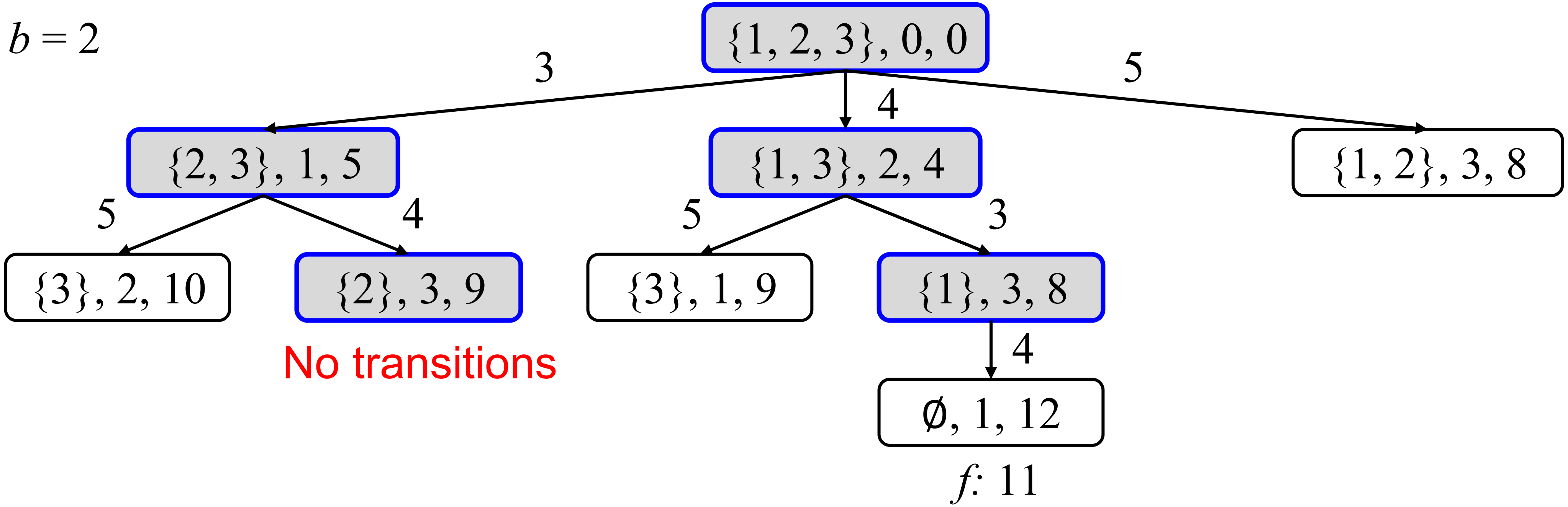
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



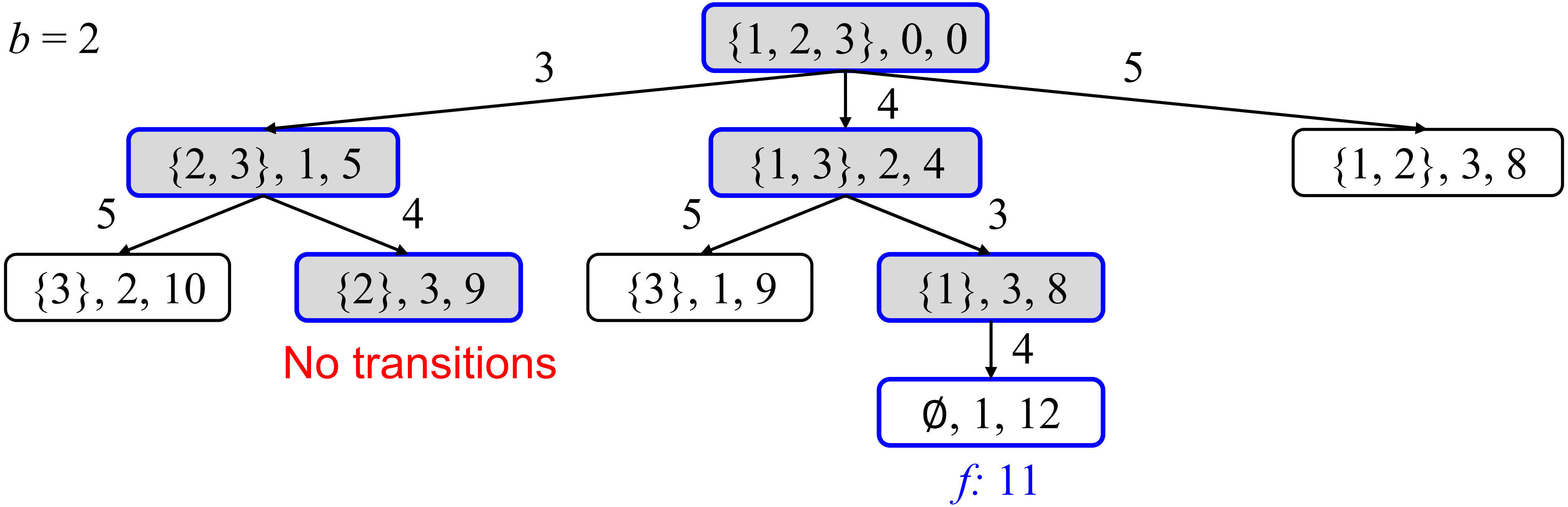
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



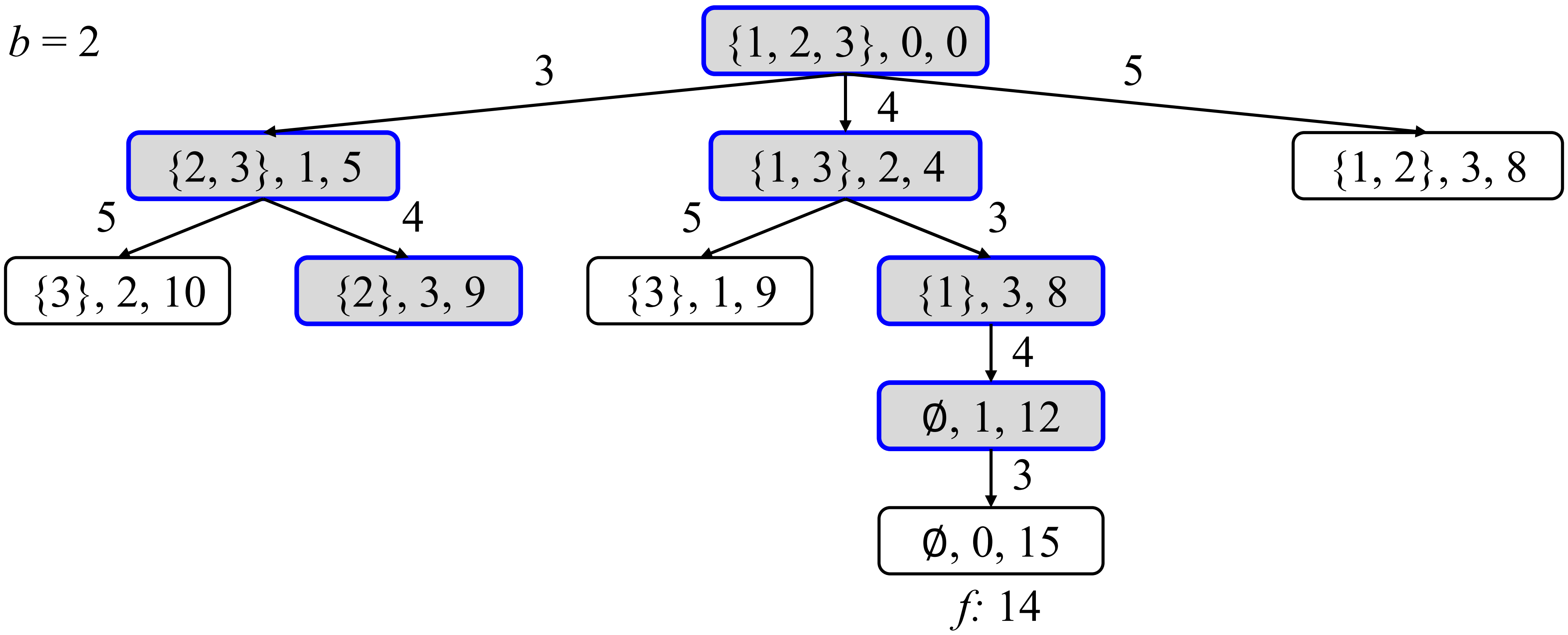
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



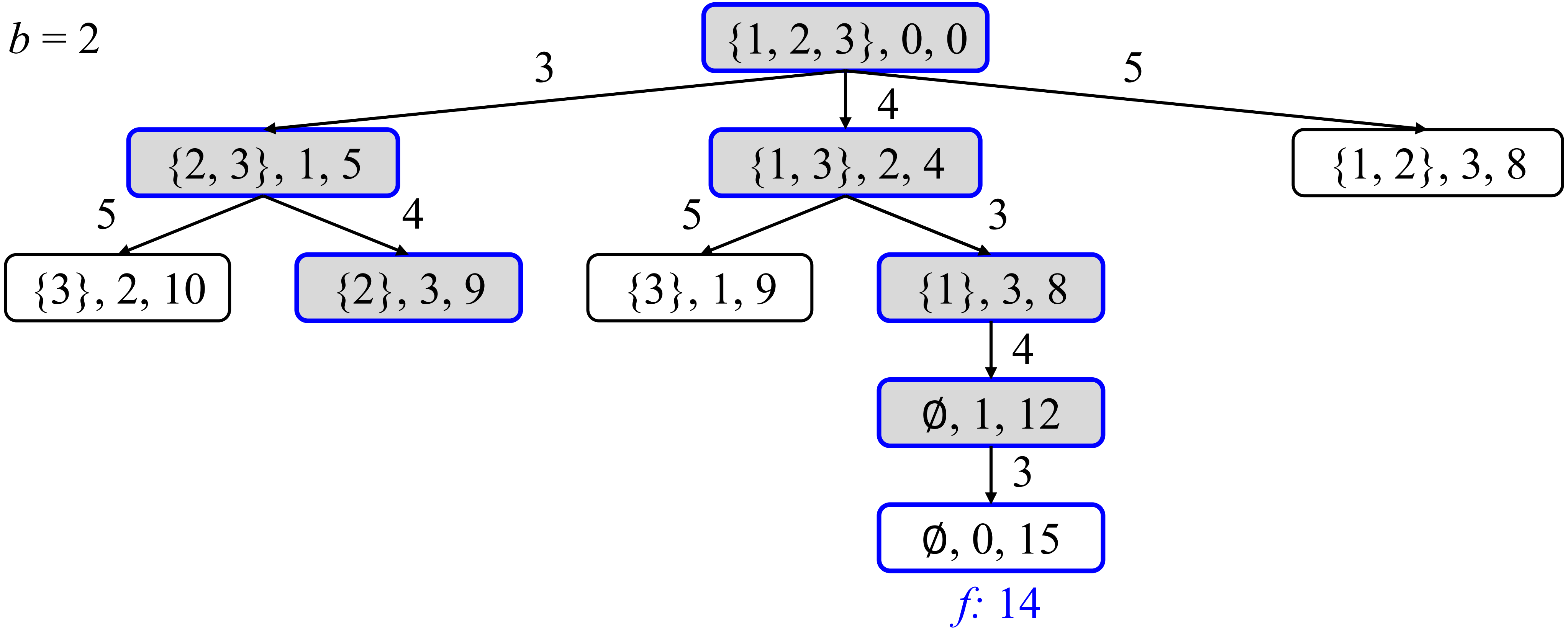
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



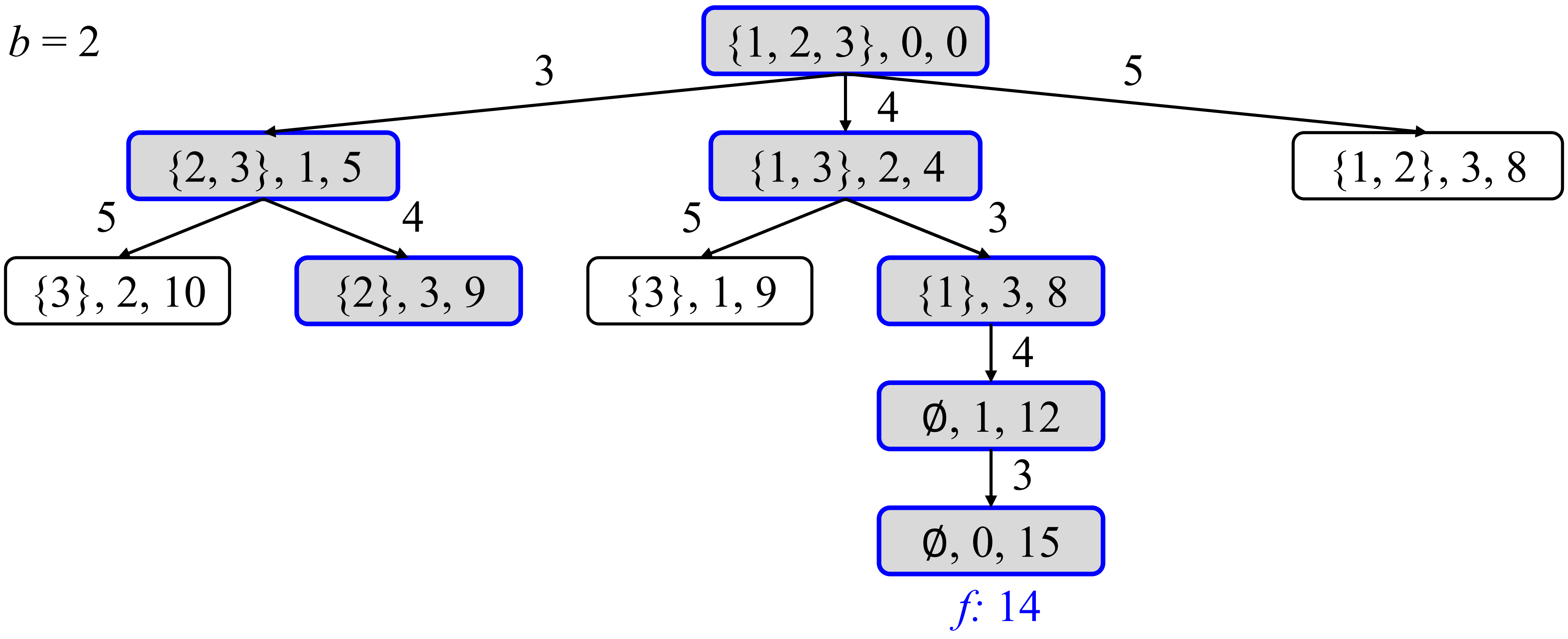
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



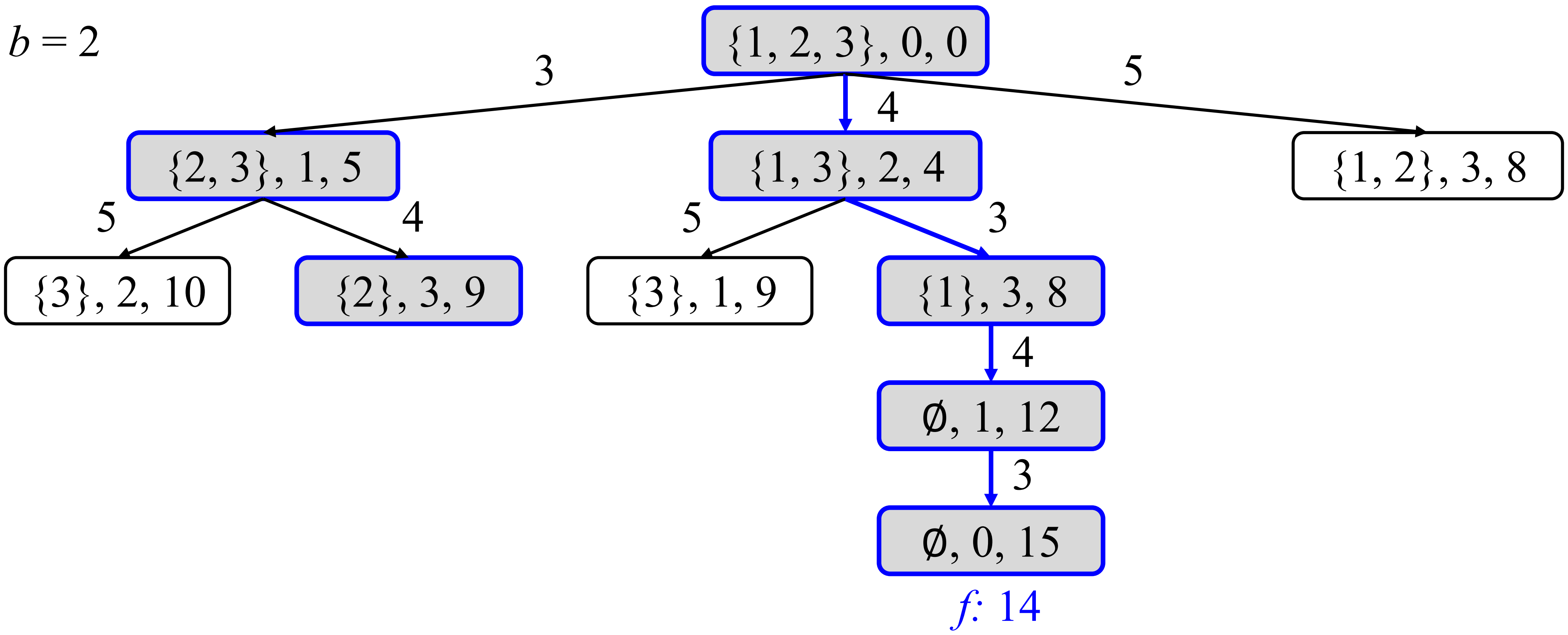
Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



Beam Search

- Keep b best states using the f-value at each layer
- No guarantee of completeness nor optimality



Complete Anytime Beam Search (CABS)

- Beam search with $b = 1, 2, 4, 8, 16, \ldots$ until states are exhausted
- Prune a state s if $f(s) \geq$ the incumbent solution cost

$b = 8$, incumbent: 14

$\boxed{\{1, 2, 3\}, 0, 0}$

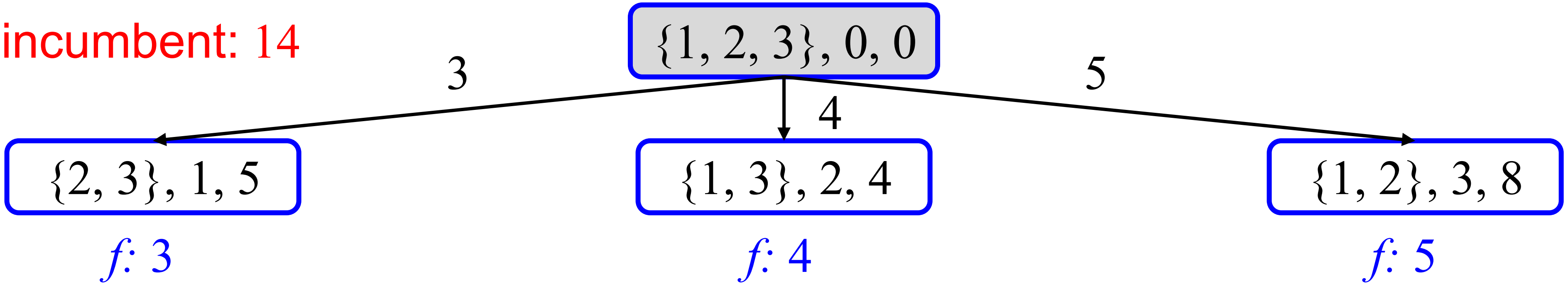
f: 0

[Zhang 1998]

Complete Anytime Beam Search (CABS)

- Beam search with $b = 1, 2, 4, 8, 16, \dots$ until states are exhausted
- Prune a state s if $f(s) \geq$ the incumbent solution cost

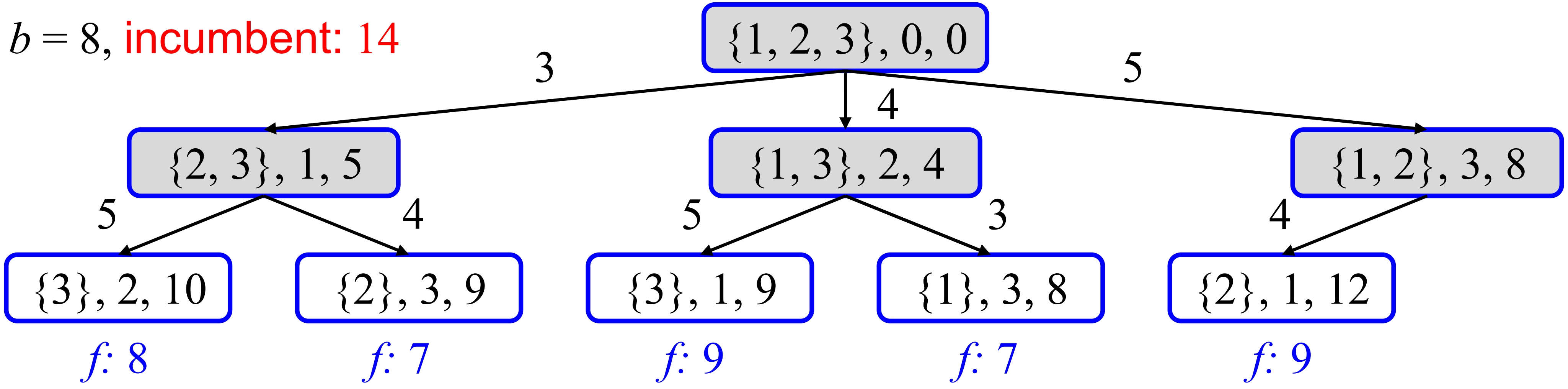
$b = 8$, incumbent: 14



[Zhang 1998]

Complete Anytime Beam Search (CABS)

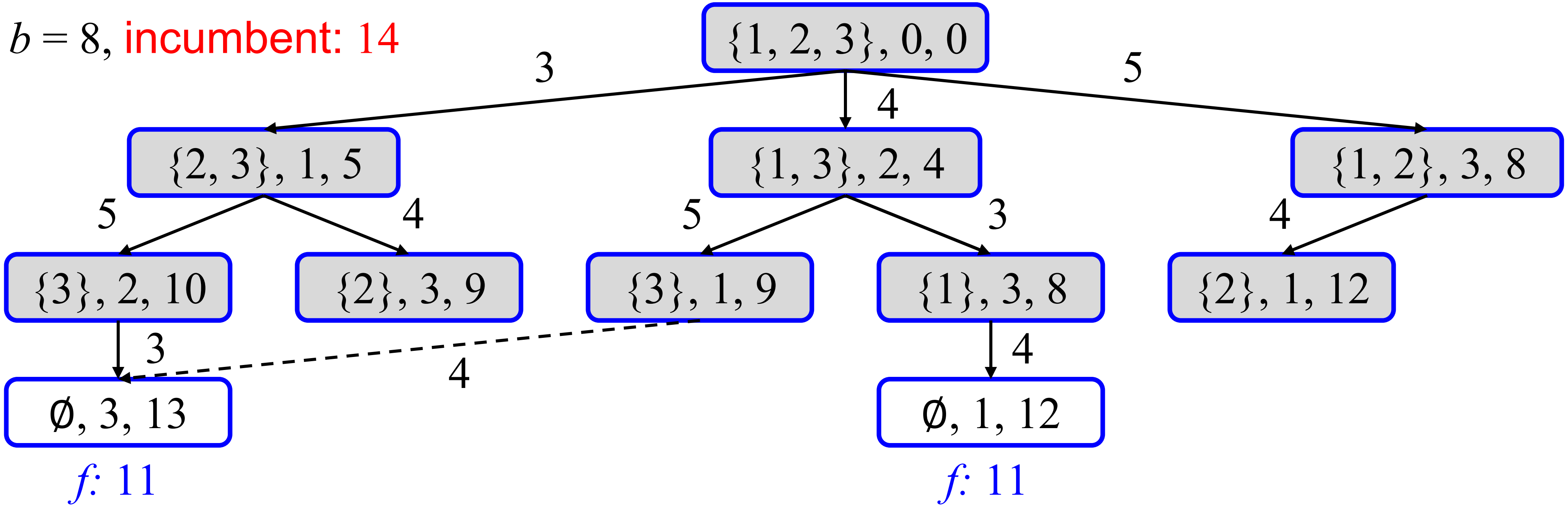
- Beam search with $b = 1, 2, 4, 8, 16, \ldots$ until states are exhausted
- Prune a state s if $f(s) \geq$ the incumbent solution cost



[Zhang 1998]

Complete Anytime Beam Search (CABS)

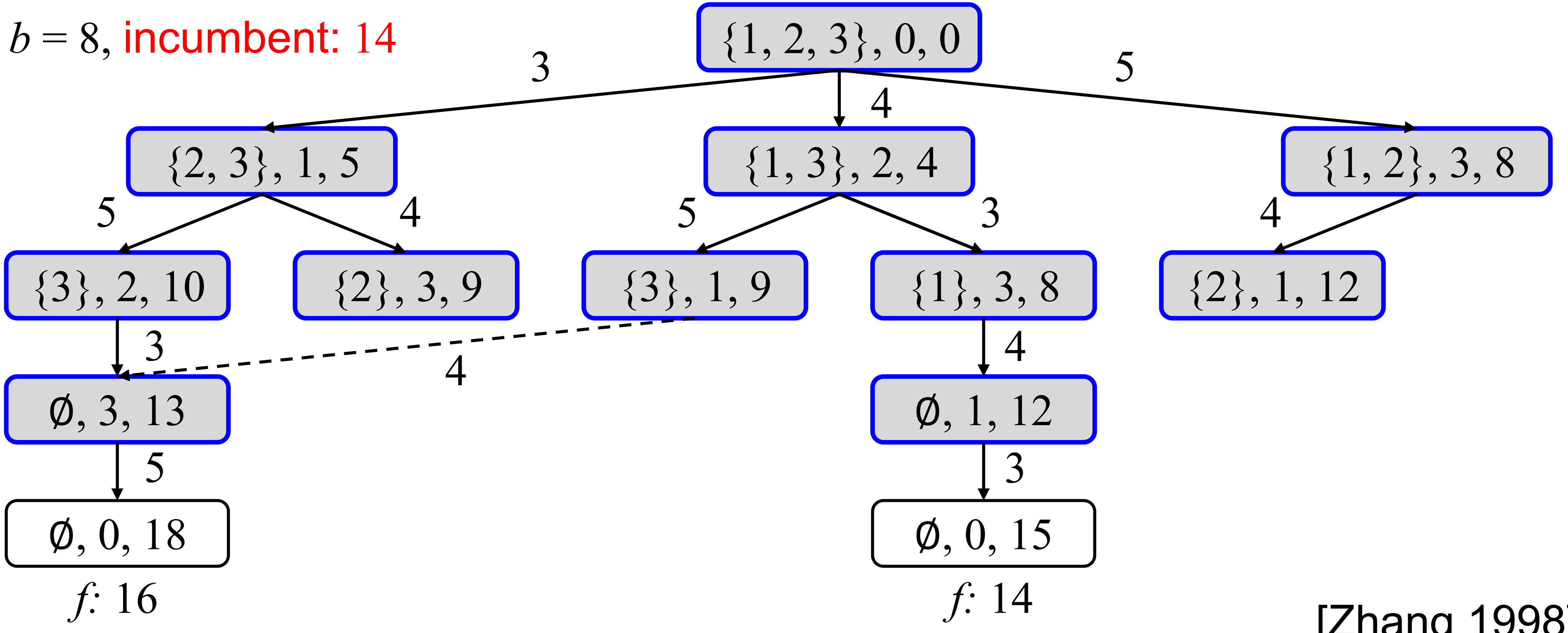
- Beam search with $b = 1, 2, 4, 8, 16, \ldots$ until states are exhausted
- Prune a state s if $f(s) \geq$ the incumbent solution cost



[Zhang 1998]

Complete Anytime Beam Search (CABS)

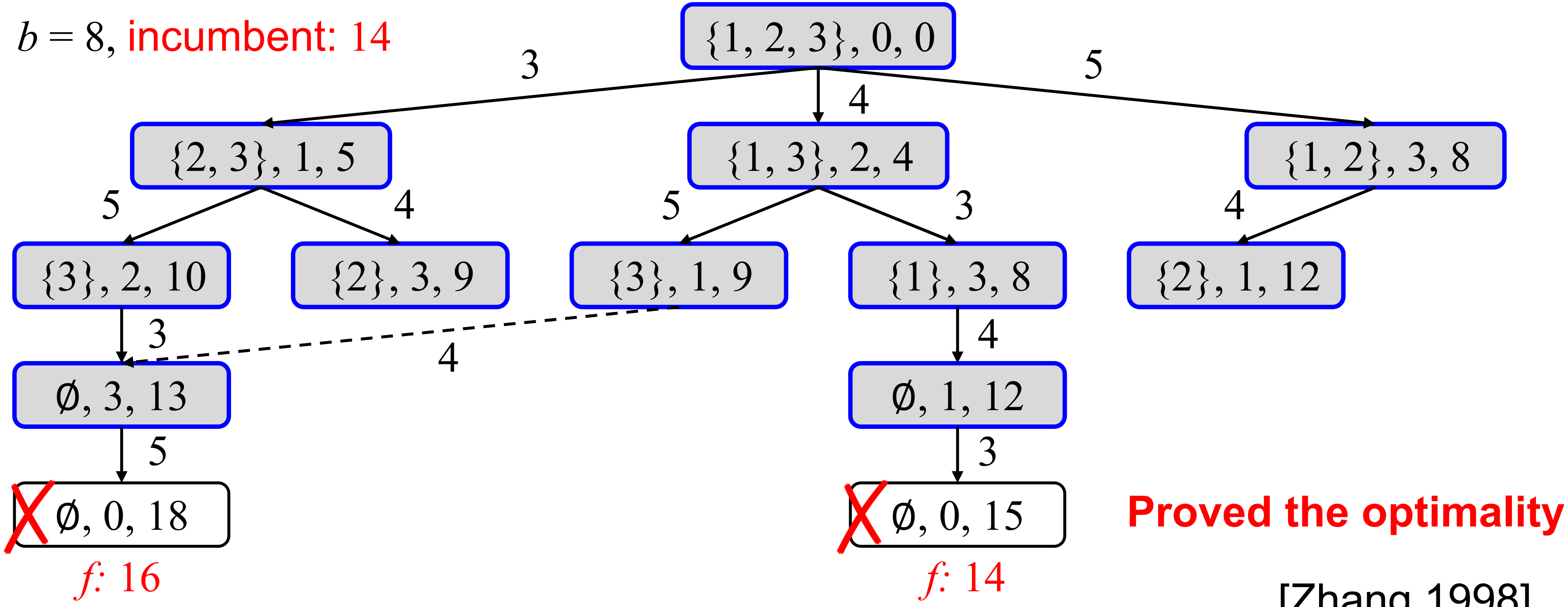
- Beam search with $b = 1, 2, 4, 8, 16, \ldots$ until states are exhausted
- Prune a state s if $f(s) \geq$ the incumbent solution cost



[Zhang 1998]

Complete Anytime Beam Search (CABS)

- Beam search with $b = 1, 2, 4, 8, 16, \ldots$ until states are exhausted
- Prune a state s if $f(s) \geq$ the incumbent solution cost



[Zhang 1998]

Experimental Evaluation of CABS

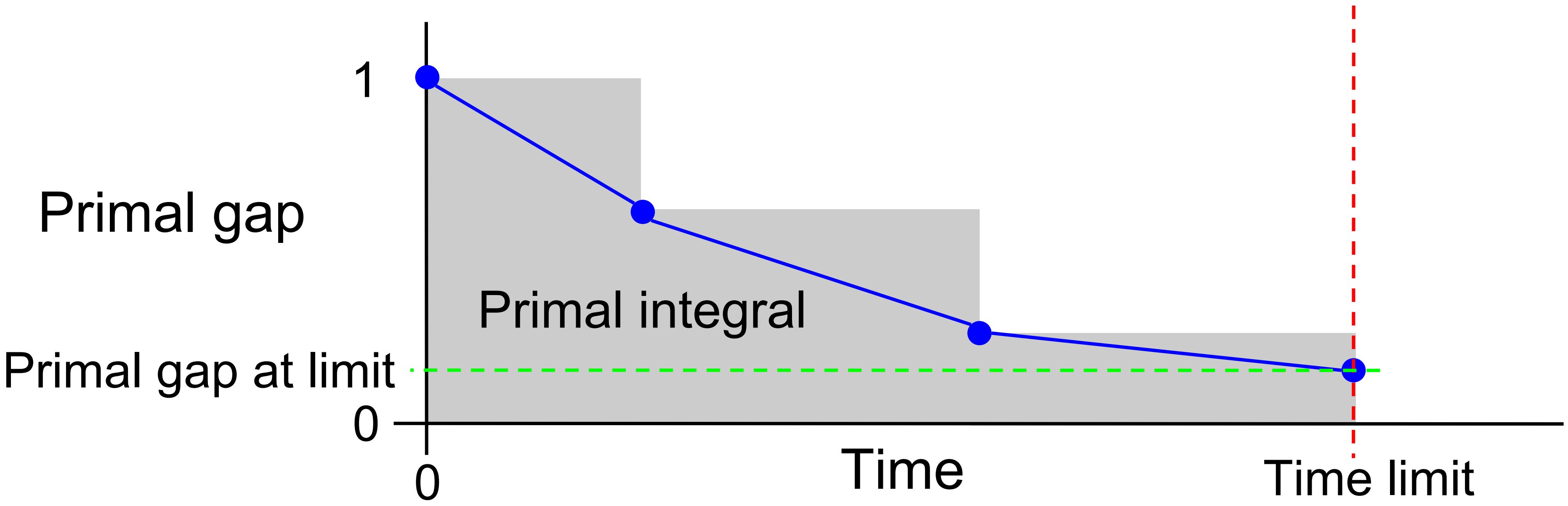
of Optimally Solved by CABS

	Description	MIP	CP	CAASDy	CABS
TSPTW (340)	TSP with time	227	47	257	**259**
CVRP (207)	vehicle routing	**26**	0	5	6
SALBP-1 (2100)	assembly line	1357	1584	1653	**1801**
Bin Packing (1615)	bin packing	1157	**1234**	922	1163
MOSP (570)	manufacturing	225	437	483	**527**
Graph-Clear (135)	building security	24	4	76	**103**
Talent Scheduling (1000)	scheduling actors	6	7	224	**253**
m-PDTSP (1117)	pick up & delivery	945	**1049**	947	1035
$1\|\|\sum w_i T_i$ (375)	job scheduling	109	150	270	**285**

# of optimally solved instances with 8 GB and 30 minutes

Primal Integral

Primal gap: $\frac{\text{solution cost} - \text{best known cost}}{\text{solution cost}}$ (1 if no solution found)



[Berthold 2013]

Mean Primal Gap/Primal Integral

	Description	MIP	CP	CABS
TSPTW (340)	TSP with time	0.227/484.1	0.026/49.0	**0.003**/**9.0**
CVRP (207)	vehicle routing	0.585/1157.4	0.317/601.2	**0.185**/**351.2**
SALBP-1 (2100)	assembly line	0.345/634.6	0.005/28.5	**0.000**/**1.9**
Bin Packing (1615)	bin packing	0.039/86.2	**0.002**/8.0	**0.002**/**5.3**
MOSP (570)	manufacturing	0.039/100.4	0.004/13.0	**0.000**/**0.4**
Graph-Clear (135)	building security	0.110/311.8	0.015/44.3	**0.000**/**0.5**
Talent Scheduling (1000)	scheduling actors	0.051/142.7	**0.002**/**18.1**	0.011/26.4
m-PDTSP (1178)	pick up & delivery	0.078/180.0	0.013/26.0	**0.002**/**5.3**
$1\|\|\sum w_i T_i$ (375)	job scheduling	0.018/74.6	**0.000**/**2.3**	0.034/73.6

Best paper runner-up at ICAPS 2023 [Kuroiwa and Beck 2023b]

Current & Future Work

DIDP Papers at CP

Tuesday, August 29th

16:00-16:50 Session 14B

Applications 3

16:00 *Arnoosh Golestanian*, *Giovanni Lo Bianco*, *Chengyu Tao* and *J. Christopher Beck*
Optimization models for pickup and delivery problems with reconfigurable capacities (abstract)

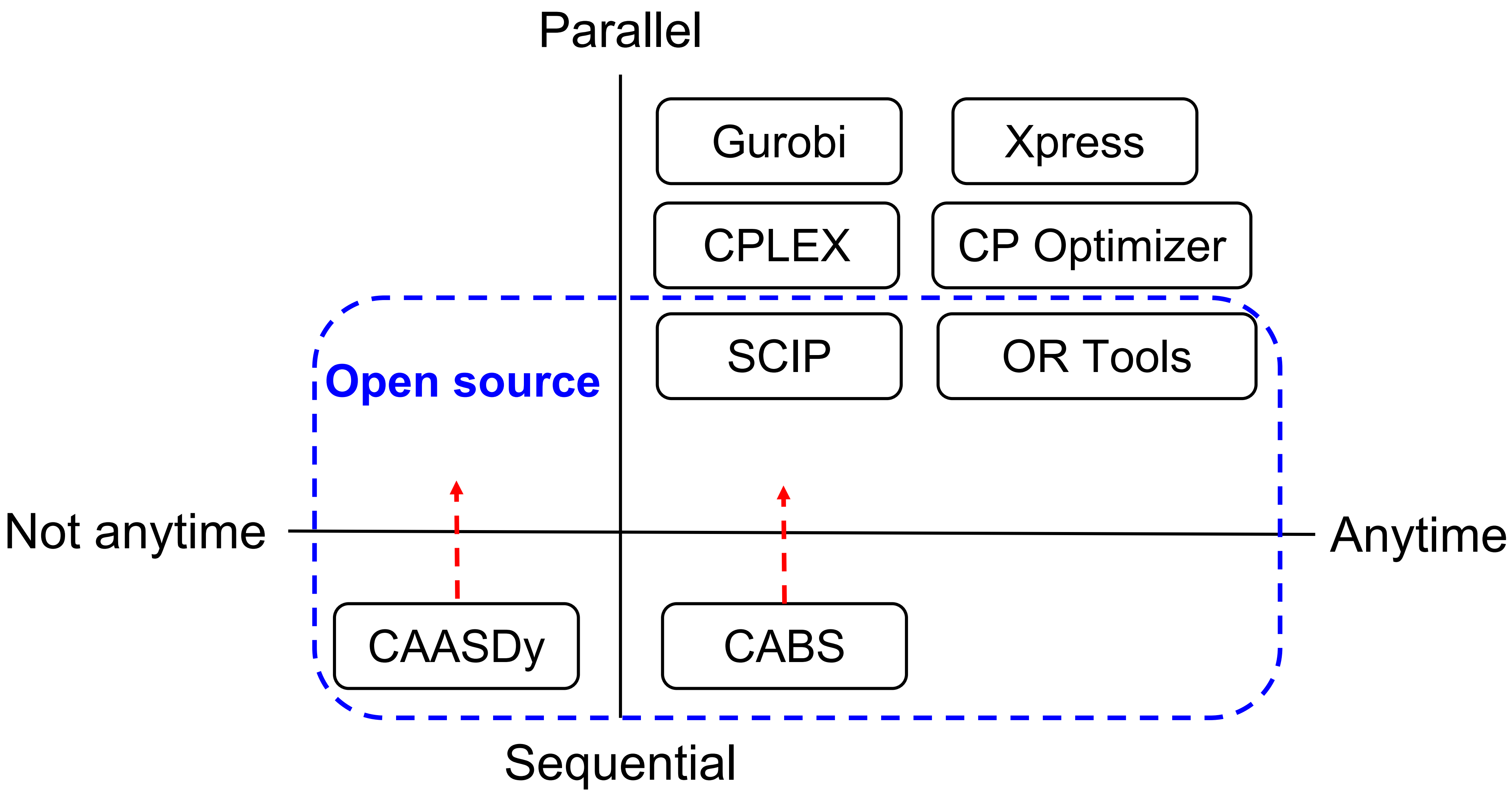
Thursday, August 31st

13:20-14:20 Session 27A

Search 3

13:50 *Ryo Kuroiwa* and *J. Christopher Beck*
Large Neighborhood Beam Search for Domain-Independent Dynamic Programming (abstract)

Building a Parallel Solver



Comparison of Solvers with 32 Threads

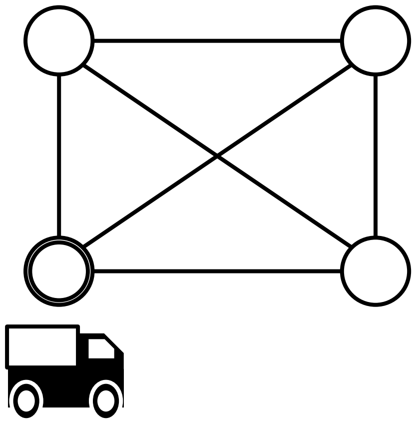
Problem	Description	Gurobi	CP Optimizer	CABS	**Prototype**
TSPTW (340)	TSP with time	239/4.2	27/0.1	235/-	**262**/**13.3**
CVRP (207)	vehicle routing	**29**/5.3	0/-	5/-	8/**9.3**
SALBP-1 (2100)	assembly line	1351/1.3	1581/1.4	1714/-	**1824**/**18.8**
Bin Packing (1615)	bin packing	1192/6.4	**1251**/9.2	1110/-	1239/**39.6**
MOSP (570)	manufacturing	238/3.1	397/0.3	507/-	**531**/**9.0**
Graph-Clear (135)	building security	16/2.0	3/3.2	92/-	**113**/**10.3**

# of optimally solved instances/speedup with 32 threads, 19 2GB, and 5 minutes

What Makes a Good Model?

What DP models are good/bad?

Problem



Model

DP Model 1

compute $V(N \setminus \{0\}, 0)$

$$V(U, i) = \min_{j \in U} c_{ij} + V(U \setminus \{i\}, j)$$

$$V(\emptyset, i) = c_{i0}.$$

DP Model 2

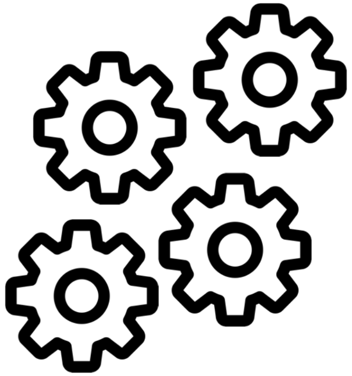
compute $V(\emptyset, 0)$

$$V(U, i) = \min_{j \notin U} c_{ji} + V(U \cup \{i\}, j)$$

$$V(N \setminus \{0\}, i) = c_{0i}$$

Solve

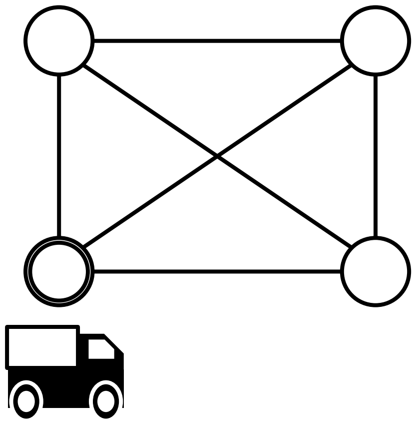
DIDP solver



Domain-Independent Dual Bound Function

Can we automatically derive a dual bound function from a model?

Problem



Model

DP Model

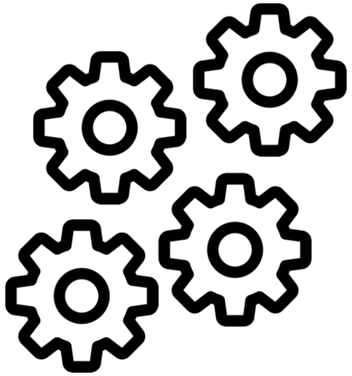
compute $V(N \setminus \{0\}, 0)$

$$V(U,i) = \min_{j \in U} c_{ij} + V(U \setminus \{i\}, j)$$

$$V(\emptyset, i) = c_{i0}.$$

Solve

DIDP solver



Derive

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

Dual bound function

Empirical Analysis of DIDP Search

- What properties of a problem do make DP efficient/inefficient?
- Apply empirical analysis conducted in SAT and CSP
- E.g., does randomized restart help?

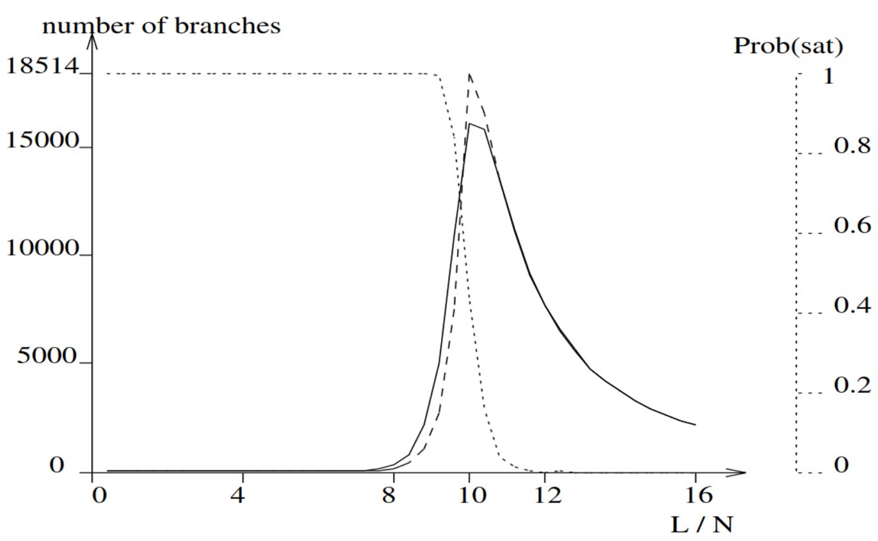
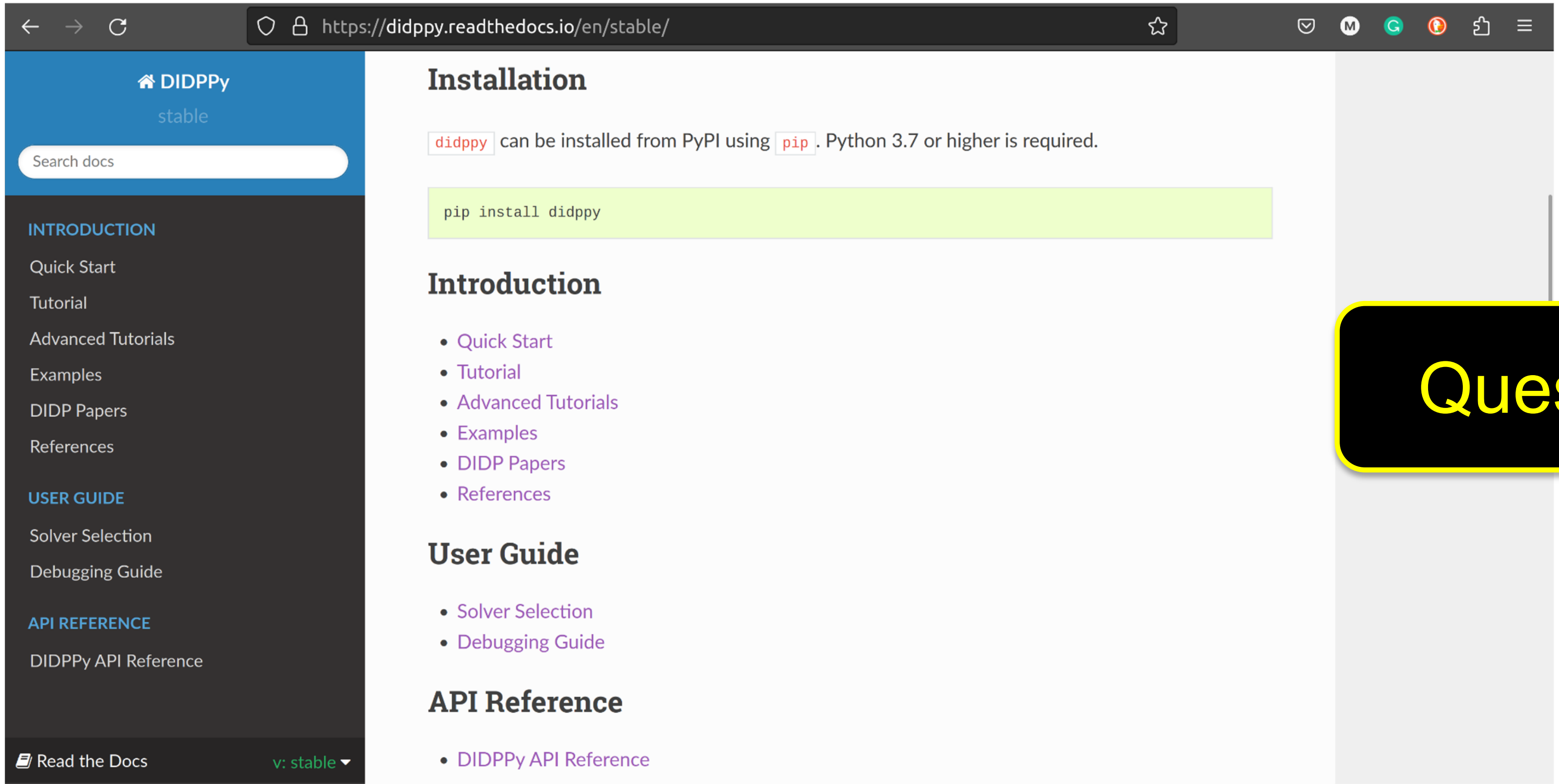


Figure 1. Random 4-SAT *problems, tested using ASAT, mean (solid), median (dashed) branches,* $N = 75$

[Gent and Walsh 1994]

Please Use DIDP on Your Problems!

- Visit our website: https://didp.ai
- Start DIDP with Python: `pip install didppy`
  Tutorials and API Reference: https://didppy.rtfd.io



Questions?