Domain-Independent Dynamic Programming

国立情報学研究所　黒岩　稜

Joint work with J. Christopher Beck at University of Toronto

Inter-University Research Institute Corporation
Research Organization of Information and Systems
National Institute of Informatics

UNIVERSITY OF TORONTO

黒岩　稜

NII助教（2024年9月ー）

- トロント大学 Department of Mechanical & Industrial Engineering 博士課程（2020/9－2024/8）
- 東京大学教養学部・総合文化研究科修士課程（2014/4－2020/3）

研究分野

- 組合せ最適化ソルバ
- グラフ探索アルゴリズム
- 並列アルゴリズム
- プランニング（人工知能）

発表の流れ

1. はじめに
2. DIDPにおけるモデリング

   [Kuroiwa and Beck ICAPS 2023a; Kuroiwa and Beck AIJ 2025]
3. 状態空間探索によるソルバ

   [Kuroiwa and Beck ICAPS 2023a,b; Kuroiwa and Beck AIJ 2025]
4. 並列ソルバ（時間があれば）

   [Kuroiwa and Beck AAAI 2024]

はじめに

組合せ最適化

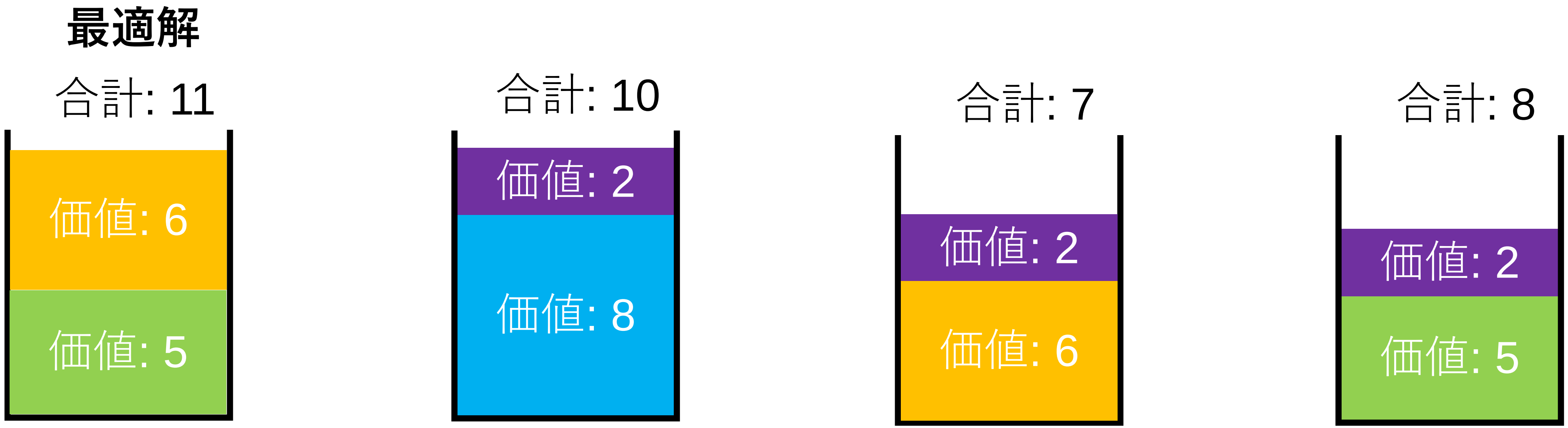
例： ナップザック問題

ナップザックに詰め込まれるアイテムの価値の合計を最大化する

組合せ最適化

例： ナップザック問題

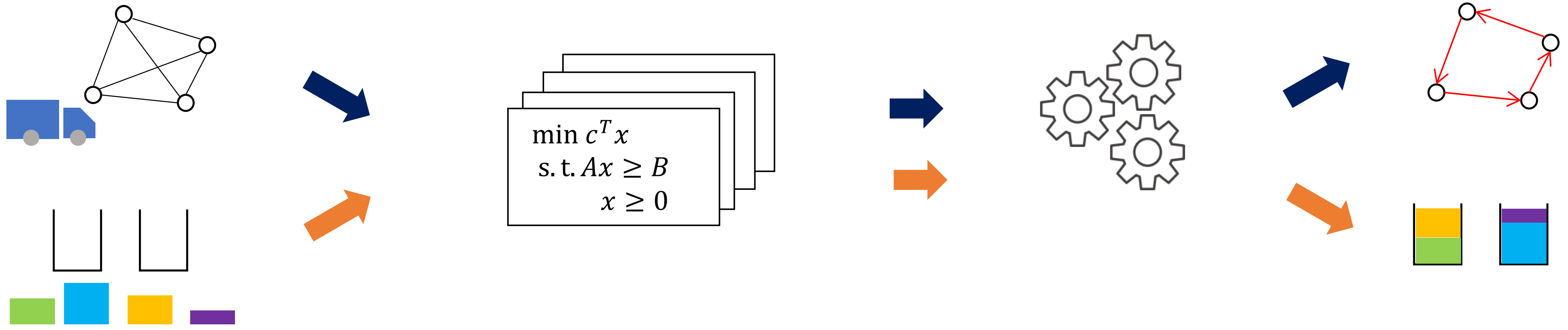
ナップザックに詰め込まれるアイテムの価値の合計を最大化する



数理モデルに基づく汎用ソルバ

- 問題を数理モデルとして定式化し、汎用ソルバで解く
- 混合整数計画法（MIP）、制約プログラミング（CP）など

組合せ最適化問題　**数理モデル**　**汎用ソルバ**　**解**



ナップザック問題を整数線形計画法（ILP）で解く

$$\max \sum_{i=1}^{n} p_i\, x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i\, x_i \leq C$$

$$x_i \in \{0, 1\}, i = 1, \dots, n$$

- C: ナップザックの容量
- w_i: アイテム i の重さ
- p_i: アイテム i の価値
- $x_i = 1$: アイテム i を詰める

ナップザック問題を整数線形計画法（ILP）で解く

$$\max \textstyle\sum_{i=1}^{n} p_i \, x_i$$

$$\text{s.t.} \quad \textstyle\sum_{i=1}^{n} w_i \, x_i \leq C$$

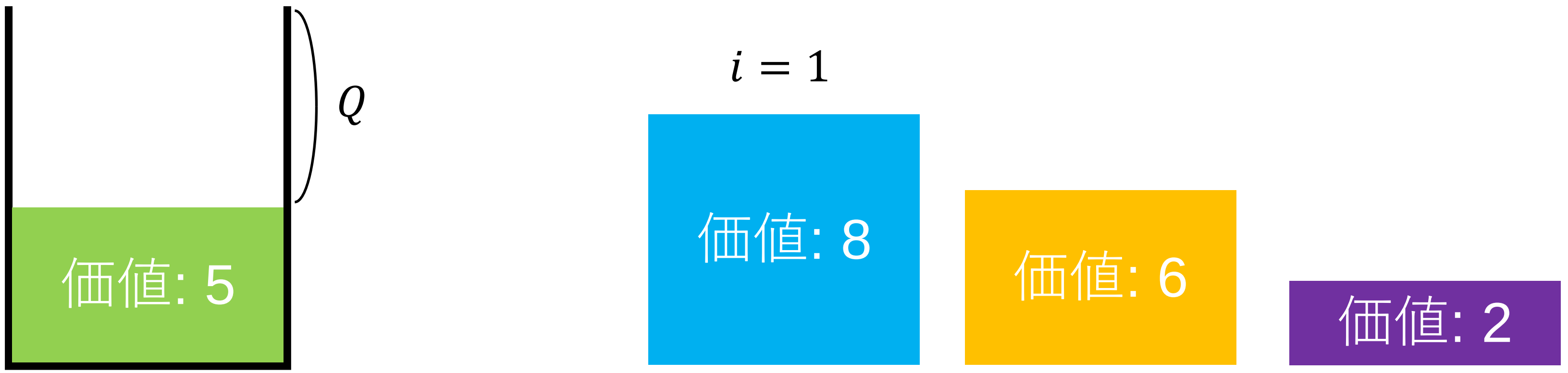
$$x_i \in \{0, 1\}, i = 1, \ldots, n$$

```python
import gurobipy as gp

C = 9
w = [4, 7, 5, 2]
p = [5, 8, 6, 2]

model = gp.Model()
model.ModelSense = gp.GRB.MAXIMIZE
x = model.addVars(len(w), vtype=gp.GRB.BINARY, obj=p)
model.addConstr(gp.quicksum(w[i]*x[i] for i in range(len(w))) <= C)

model.optimize()
```

ナップザック問題を動的計画法（DP）で解く

- 残りの容量 Q と現在考慮しているアイテム i で**状態**を表す
- $V(Q,i)$ で現在の状態から達成できる最大の合計価値を表す

$$V(Q,i)=\begin{cases}\max\{p_i+V(Q-w_i,i+1),V(Q,i+1)\} & \text{if } i\le n \text{ and } Q\ge w_i\\ V(Q,i+1) & \text{if } i\le n \text{ and } Q< w_i\\ 0 & \text{if } i=n+1\end{cases}$$

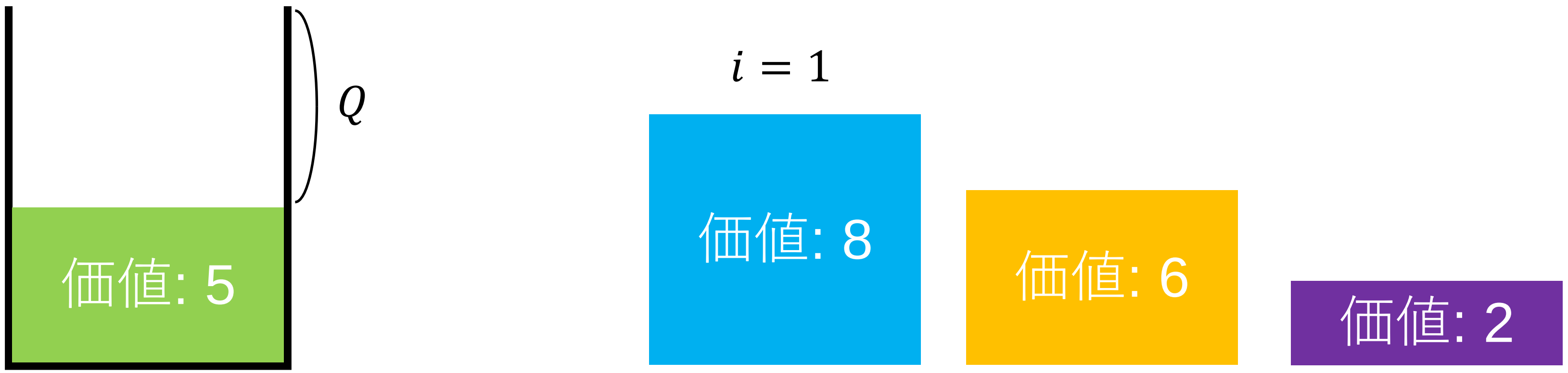


ナップザック問題を動的計画法（DP）で解く

- 残りの容量 Q と現在考慮しているアイテム i で**状態**を表す
- $V(Q,i)$ で現在の状態から達成できる最大の合計価値を表す

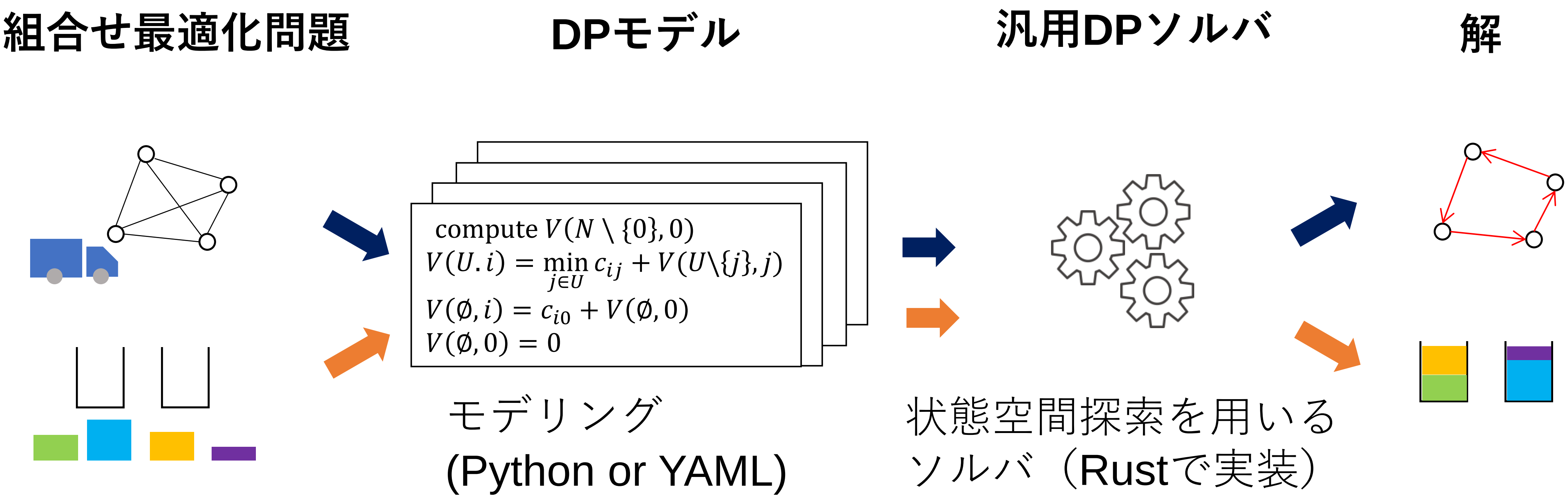
$$V(Q,i)=\begin{cases}\max\{p_i+V(Q-w_i,i+1),V(Q,i+1)\} & \text{if } i\le n \text{ and } Q\ge w_i\\ V(Q,i+1) & \text{if } i\le n \text{ and } Q< w_i\\ 0 & \text{if } i=n+1\end{cases}$$

Gurobiのような汎用ソルバを作れないか？

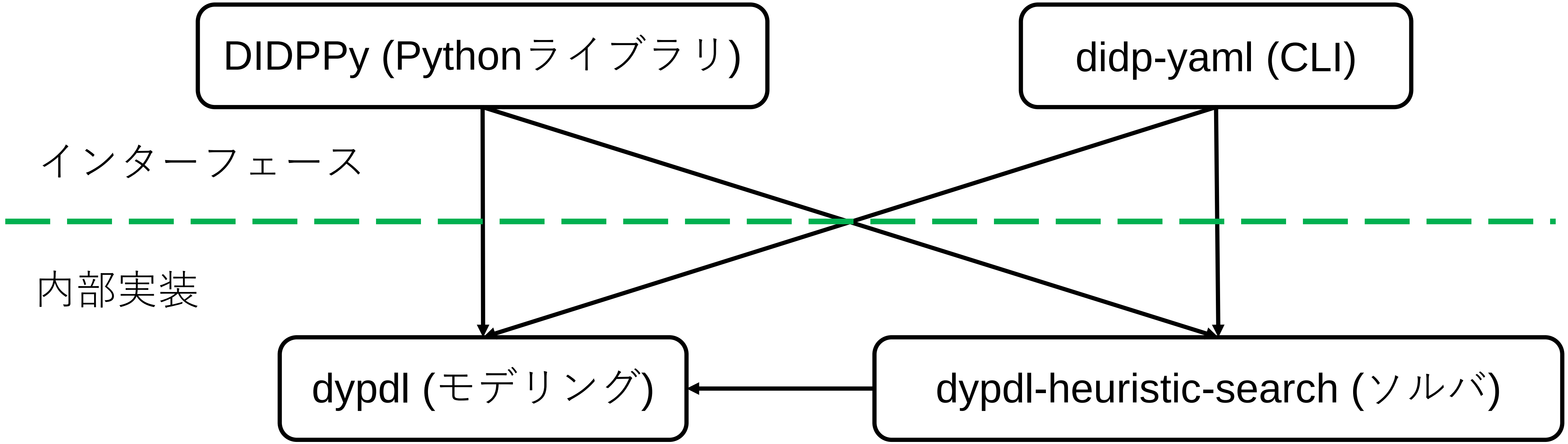


Domain-Independent Dynamic Programming (DIDP)

問題をDPモデルとして定式化し、汎用ソルバで解く

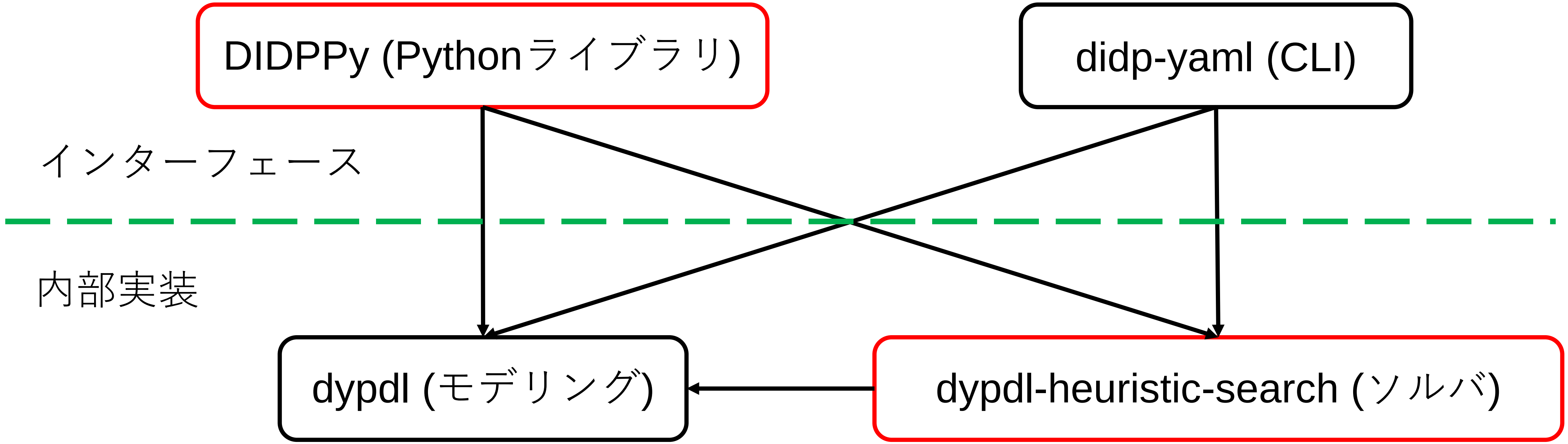


オープンソースソフトウェア：didp-rs



全てRustで実装されている

オープンソースソフトウェア：didp-rs



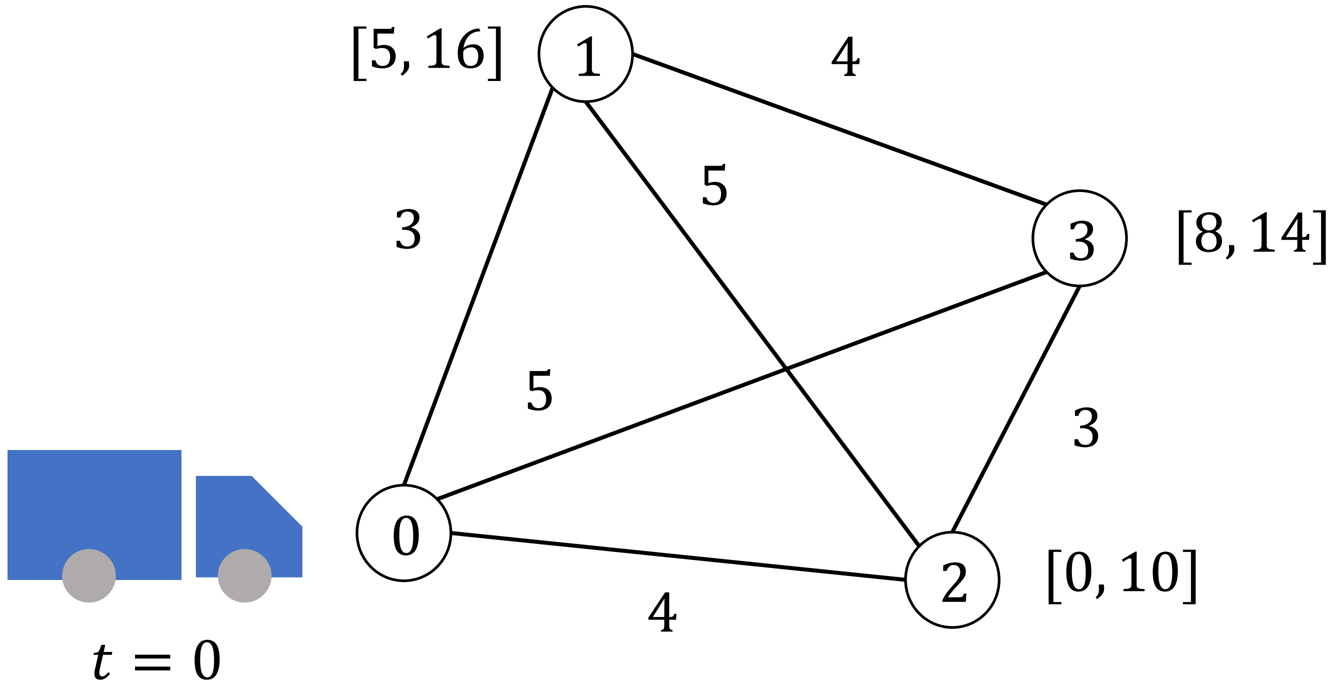
全てRustで実装されている

DIDPにおけるモデリング

[Kuroiwa and Beck ICAPS 2023a; Kuroiwa and Beck AIJ 2025]

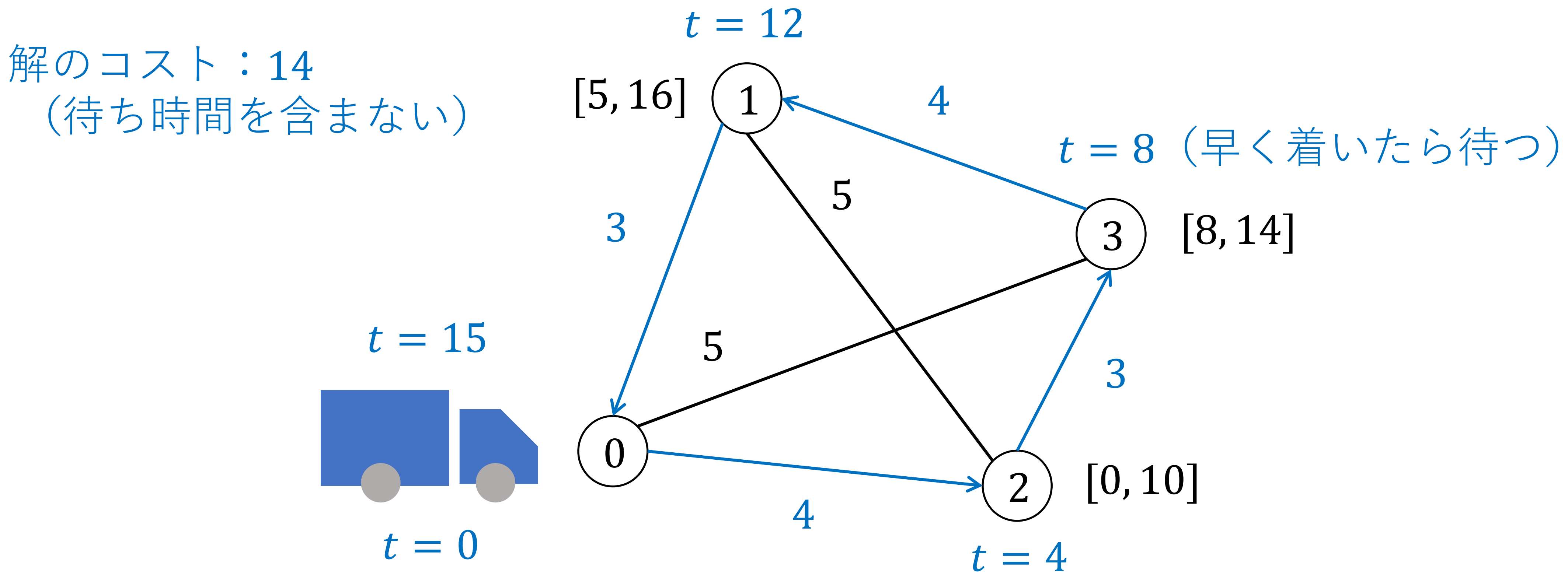
例：時間枠付き巡回セールスパーソン（TSPTW）

デポ（0）を $t = 0$ に出発して全ての顧客を時間枠内に訪問して戻る巡回路の合計の移動時間を最小化する



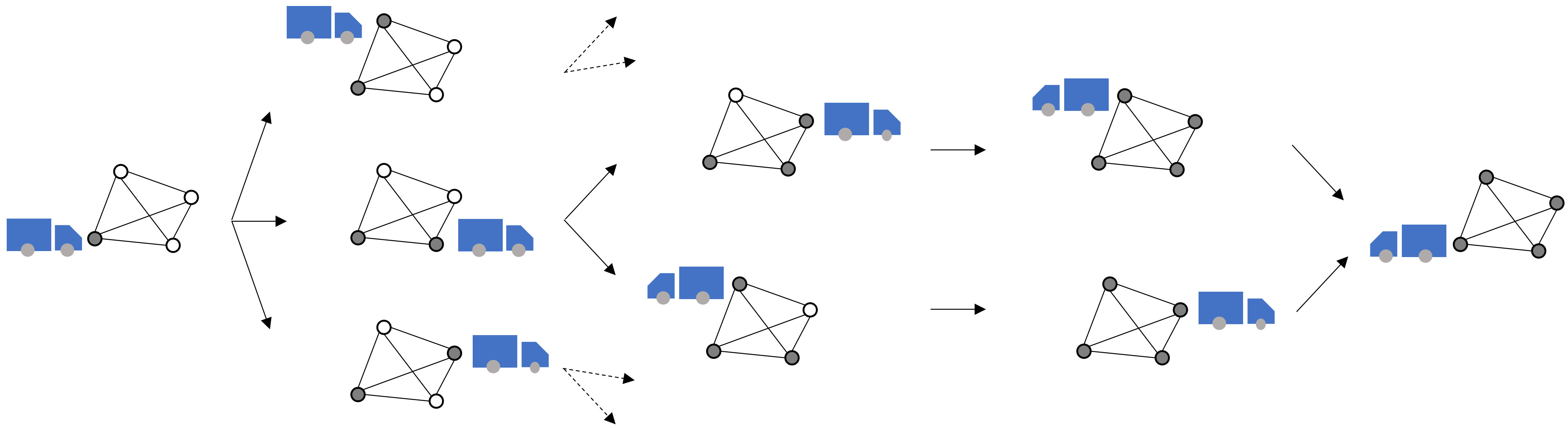
例：時間枠付き巡回セールスパーソン（TSPTW）

デポ（0）を $t=0$ に出発して全ての顧客を時間枠内に訪問して戻る巡回路の合計の移動時間を最小化する



TSPTWのDPモデル

顧客を1つずつ訪問する



TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

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

j を訪問する

デポに戻る

Base case

状態変数：
- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：
- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

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

j を訪問する

デポに戻る

Base case

状態変数：
- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：
- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

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

j を訪問する

デポに戻る

Base case

状態変数：

- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：

- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

$$V(U,i,t) = \begin{cases} \boxed{\min\limits_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\left\{t + c_{ij}, a_j\right\}\right) \quad \text{if } U \neq \emptyset} & j \text{ を訪問する} \\ c_{i0} + V(U, 0, t + c_{i0}) \quad \text{if } U = \emptyset, i \neq 0 & \text{デポに戻る} \\ 0 \quad \text{if } U = \emptyset, i = 0 & \text{Base case} \end{cases}$$

状態変数：

- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：

- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

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

j を訪問する

デポに戻る

Base case

状態変数：

- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：

- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

$$V(U, i, t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \quad j\text{ を訪問する} \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \quad \text{デポに戻る} \\ \boxed{0 \quad \text{if } U = \emptyset, i = 0} & \text{Base case} \end{cases}$$

状態変数：
- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：
- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

TSPTWのDPモデル

状態価値関数 V を再帰的に定義する

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

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

- 1行目: j を訪問する
- 2行目: デポに戻る
- 3行目: Base case

状態変数：

- U：未訪問の顧客
- i：現在の位置
- t：現在の時刻

定数：

- N：顧客の集合（0：デポ）
- $[a_j, b_j]$：顧客 j の時間枠
- c_{ij}：i から j への移動時間

これまでの研究では個別にアルゴリズムを実装することで解かれてきた

DIDPPyによるモデリング

```
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)
```

DIDPPyによるモデリング

```python
import didppy as dp
```

ライブラリをインポート

```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)
```

DIDPPyによるモデリング

```
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)
```

状態価値関数を最小化するモデル

定数の定義

```
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)
```

定数

顧客の集合 $N = \{0,1,2,3\}$

時間枠 $[a_j, b_j]$

移動時間 c_{ij}

定数の定義

```
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)
```

Table：状態変数 i をインデックスとして使える

状態変数の定義

```
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)
```

未訪問の顧客 U
現在の位置 i
現在の時刻 t

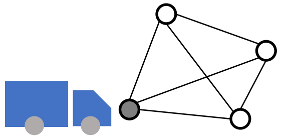
Tagret Stateの定義

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

```
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]])
```

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

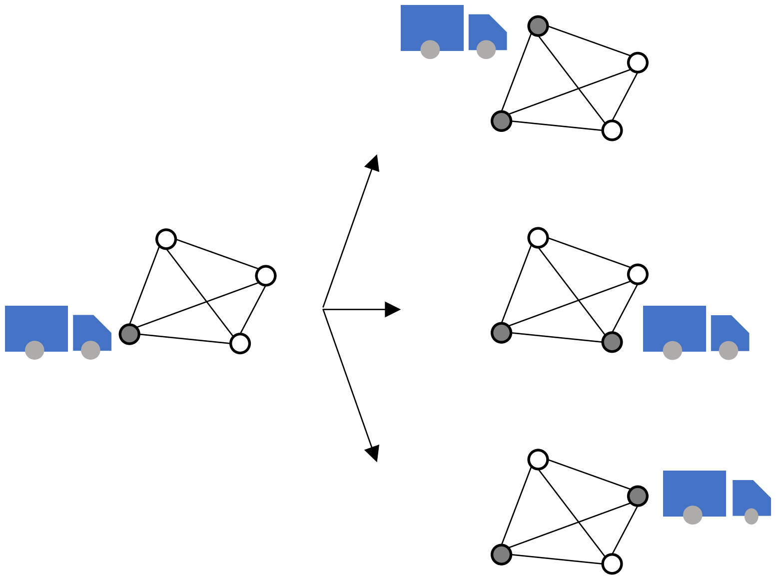
```
u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_var(target=0)
```



状態遷移の定義

```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)
```

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

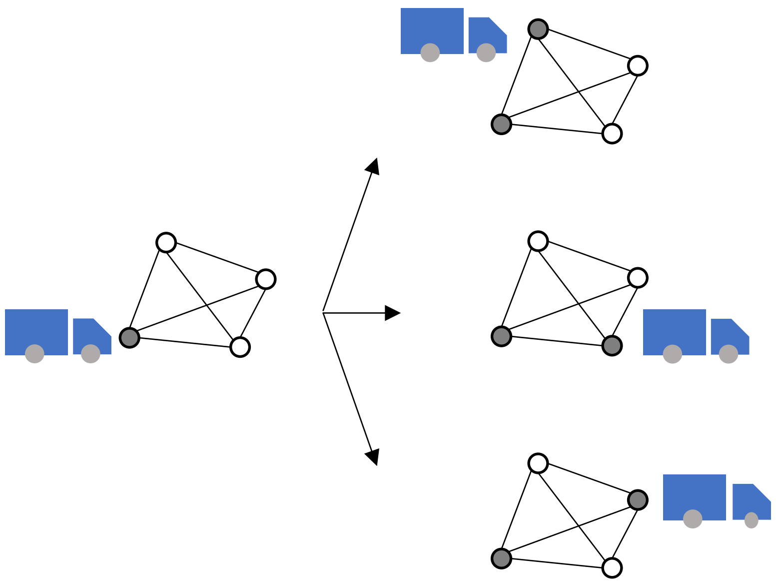


状態遷移の定義

```
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)
```

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

状態価値関数 V の計算方法



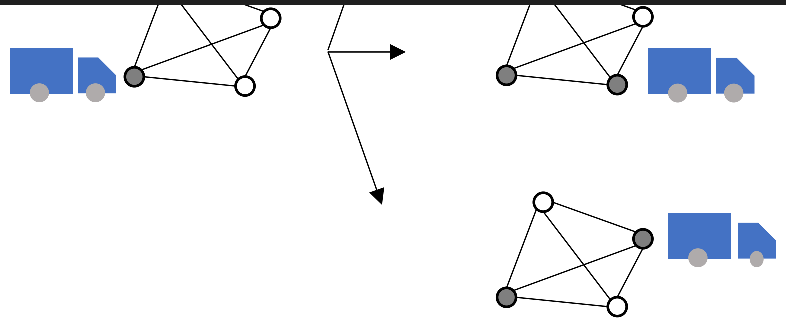
状態遷移の定義

```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)
```

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

状態価値関数 V の計算方法

```python
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]])
```

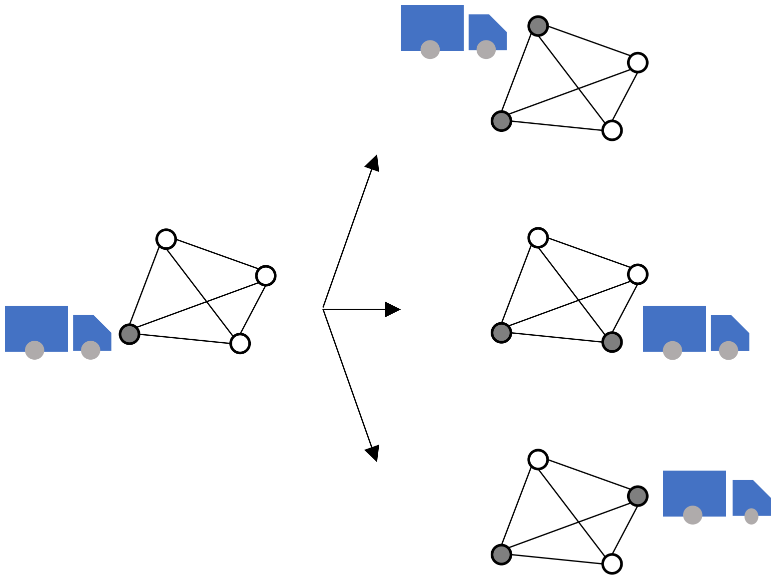


状態遷移の定義

```
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)
```

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

遷移先の状態の計算方法

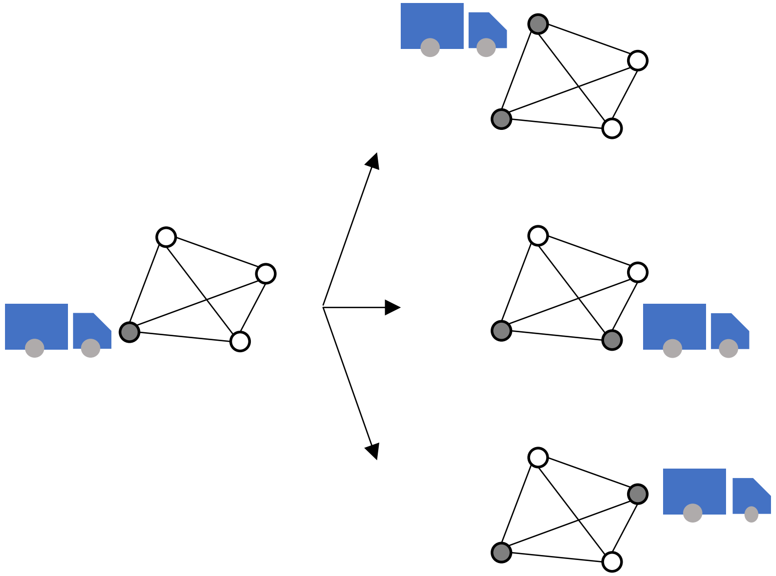


状態遷移の定義

```
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)
```

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

状態遷移を適用するための条件

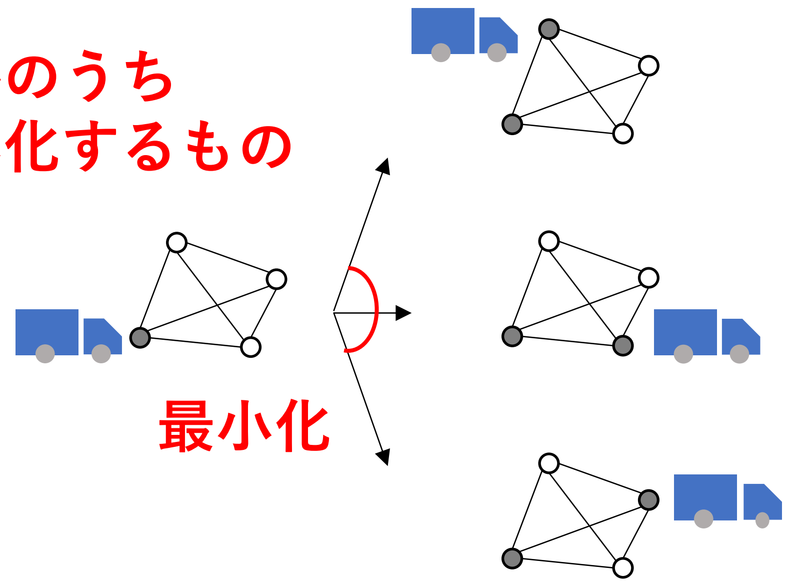


状態遷移の定義

```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)
```

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

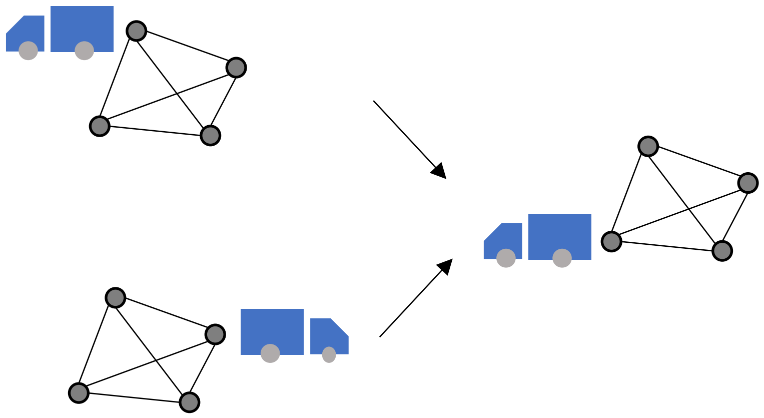
適用可能な状態遷移のうち状態価値関数を最小化するものを選ぶ



状態遷移の定義

```
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)
```

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



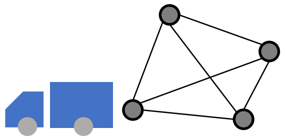
Base Case

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

Base Case

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

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



冗長な情報を使ったより良いモデルの定式化

問題の定義に含意されている**冗長な情報を明示的に定義する**

- **状態制約**：$V(U,i,t) = \infty \text{ if } \exists j \in U, t + c_{ij} > b_j$

```
for j in range(1, 4):
    model.add_state_constr(~u.contains(j) | (t + c[i, j] <= b[j]))
```

冗長な情報を使ったより良いモデルの定式化

問題の定義に含意されている**冗長な情報を明示的に定義する**

- **状態制約**：$V(U,i,t) = \infty \text{ if } \exists j \in U, t + c_{ij} > b_j$

```
for j in range(1, 4):
    model.add_state_constr(~u.contains(j) | (t + c[i, j] <= b[j]))
```

- **リソース変数**による不等式：$V(U,i,t) \leq V(U,i,t') \text{ if } t \leq t'$

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

冗長な情報を使ったより良いモデルの定式化

問題の定義に含意されている**冗長な情報を明示的に定義する**

- **状態制約**：$V(U,i,t) = \infty \text{ if } \exists j \in U, t + c_{ij} > b_j$

```
for j in range(1, 4):
    model.add_state_constr(~u.contains(j) | (t + c[i, j] <= b[j]))
```

- **リソース変数**による不等式：$V(U,i,t) \leq V(U,i,t') \text{ if } t \leq t'$

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

- **双対限界** (下界) 関数：$V(U,i,t) \geq \sum_{j\in U} \min_{k\in N} c_{kj}$

```
cmin = model.add_int_table([3, 3, 3, 3])
model.add_dual_bound(cmin[u])
```

定式化された問題をソルバで解く

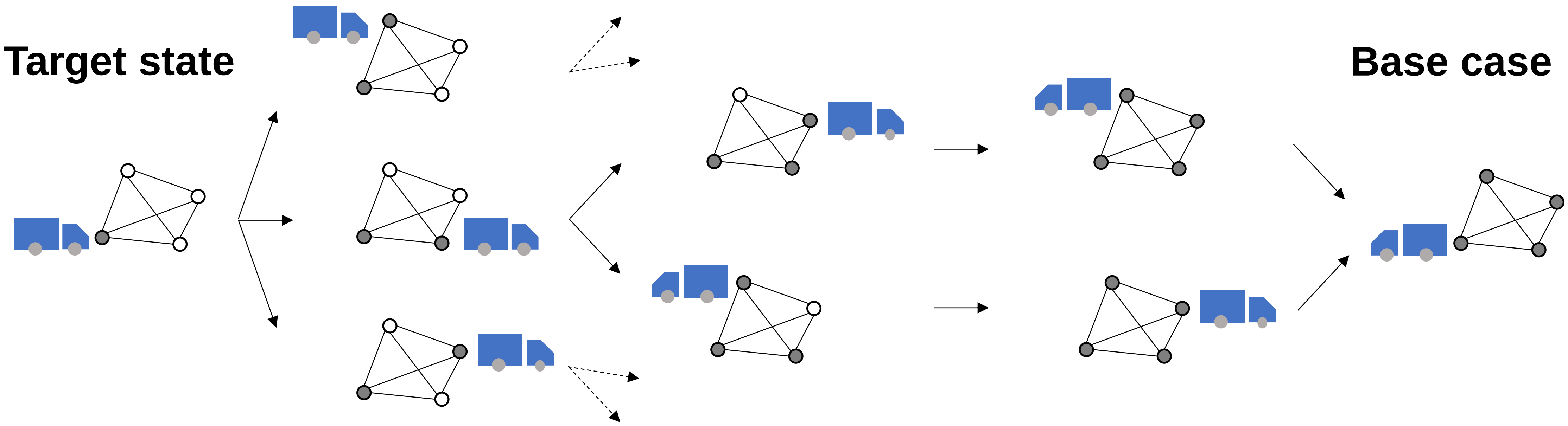
```
solver = dp.CABS(model)
solution = solver.search()

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)
```

状態空間探索によるソルバ

[Kuroiwa and Beck ICAPS 2023a,b; Kuroiwa and Beck AIJ 2025]

DPを最短経路問題として解く

DPモデルを状態遷移グラフ上の経路を探すことで解く



TSPTWのDPモデルの状態遷移グラフ

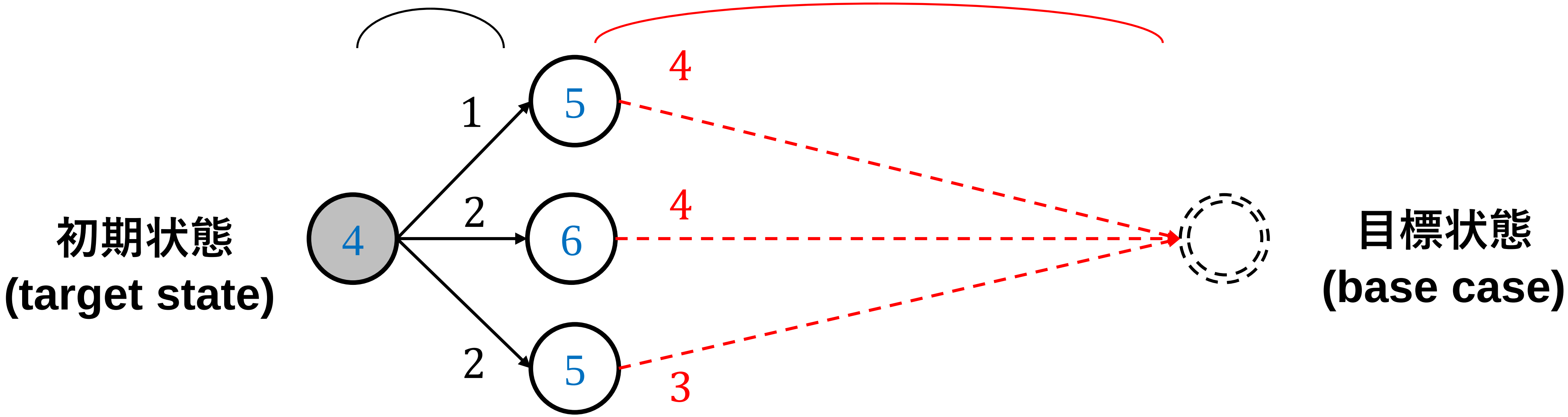
状態空間探索（ヒューリスティック探索）でDPを解く

目標状態が見つかるまで状態を展開することで経路を探す

f 値：展開する際の優先度、 $g+h$（その状態を通る経路コストの下界）

g 値：初期状態からその状態への経路コスト

h 値：その状態から目標状態への経路コストの下界
（現状のDIDPではモデルに定義された双対限界関数で計算）



CAASDy：A*でDPを解く

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

g: 0

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

Target state

A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

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

双対限界関数

$$V(\{1,2,3\}, 0, 0) \ge \sum_{j \in \{1,2,3\}} \min_{k \in N} c_{kj} = 9$$

g: 0

h: 9

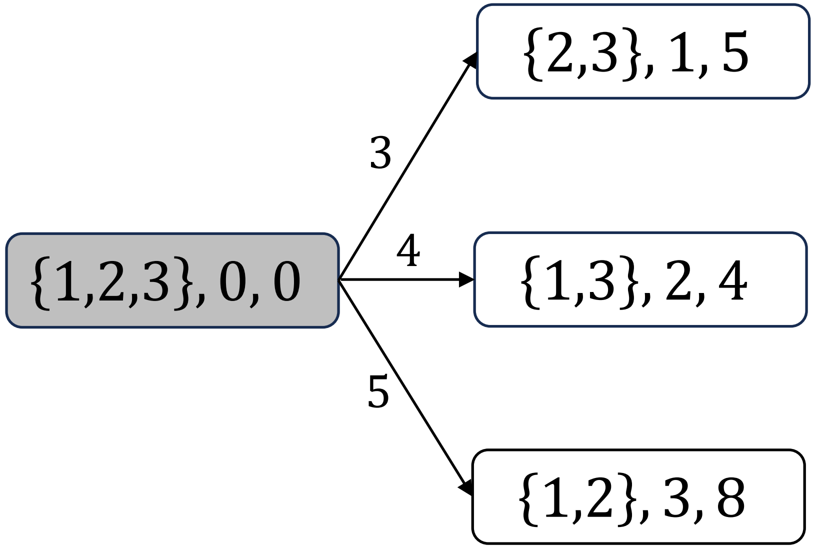
f: 9 $\boxed{\{1,2,3\}, 0, 0}$ ⟶ $\emptyset, 0, t^*$

Target state

A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

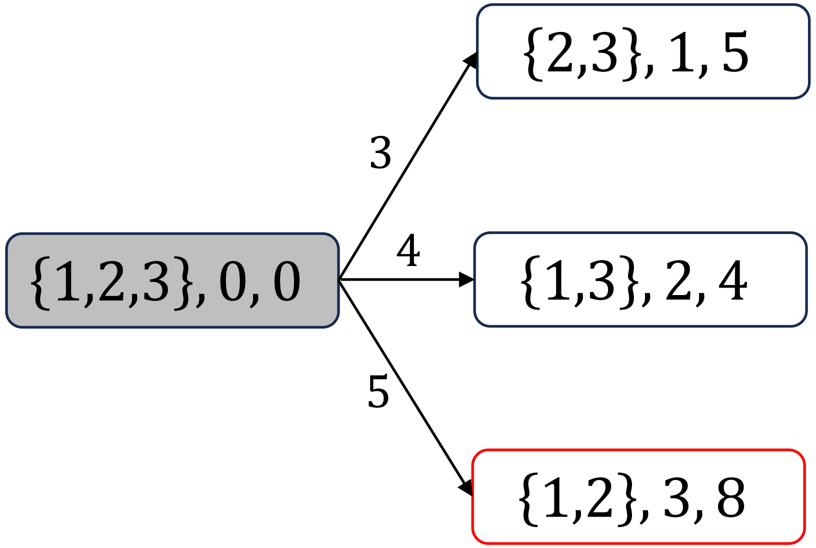
$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \leq b_j} \boxed{c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right)} & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

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



状態制約

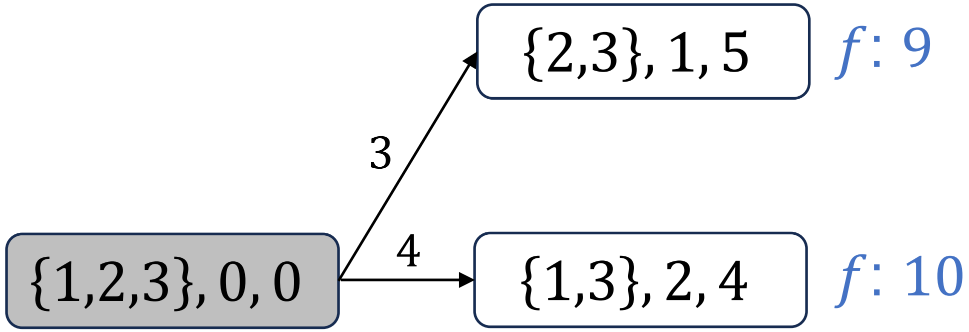
$$V(U,i,t) = \infty \text{ if } \exists j \in U, t + c_{ij} > b$$

$$V(\{1,2\},3,8) = \infty \leftarrow 2 \in \{1,2\}, t + c_{32} = 8 + 3 > b_2 = 10$$

A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

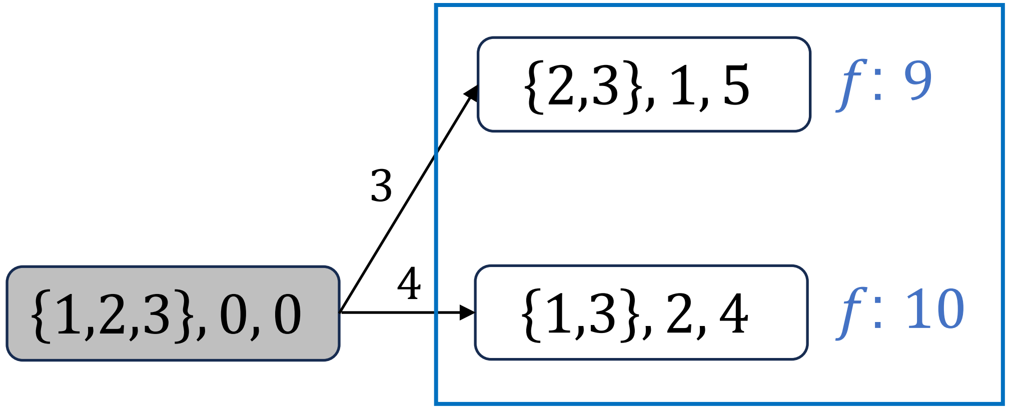
$$V(U,i,t) = \begin{cases} \min\limits_{j \in U: t + c_{ij} \le b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

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

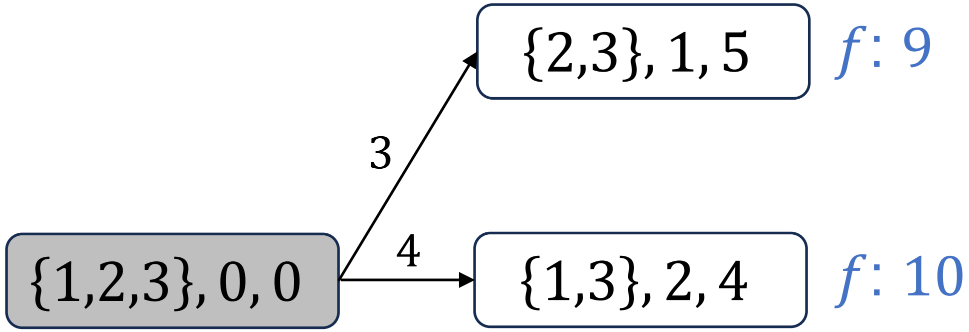


最短経路コストの下界：9

A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

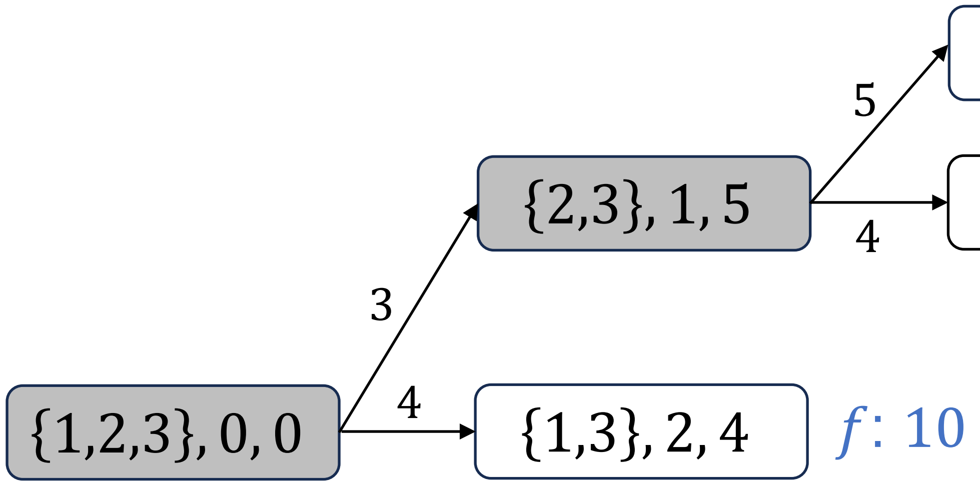
$$V(U, i, t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

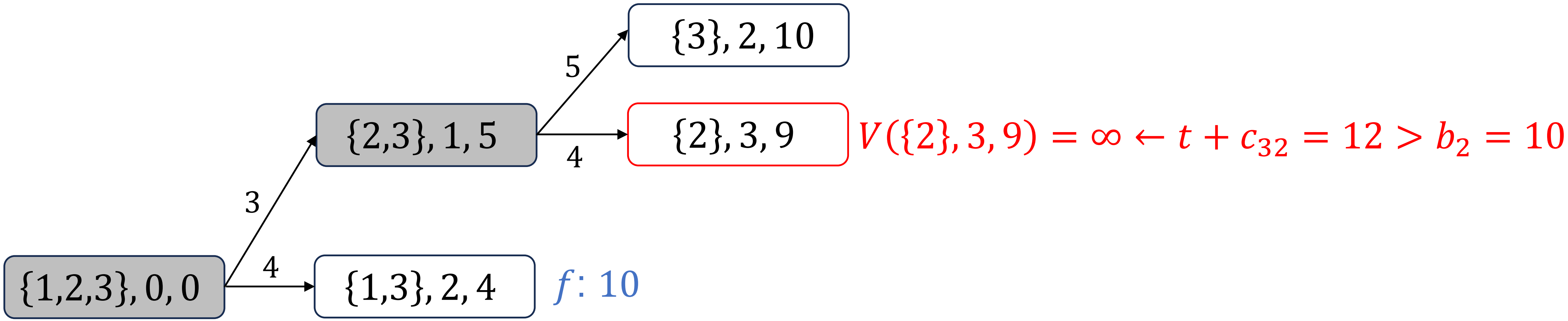
$$V(U,i,t)=\begin{cases}\min\limits_{j\in U:t+c_{ij}\le b_j}\boxed{c_{ij}+V\big(U\setminus\{j\},j,\max\{t+c_{ij},a_j\}\big)} & \text{if } U\neq\emptyset\\ c_{i0}+V(U,0,t+c_{i0}) & \text{if } U=\emptyset, i\neq 0\\ 0 & \text{if } U=\emptyset, i=0\end{cases}$$



A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

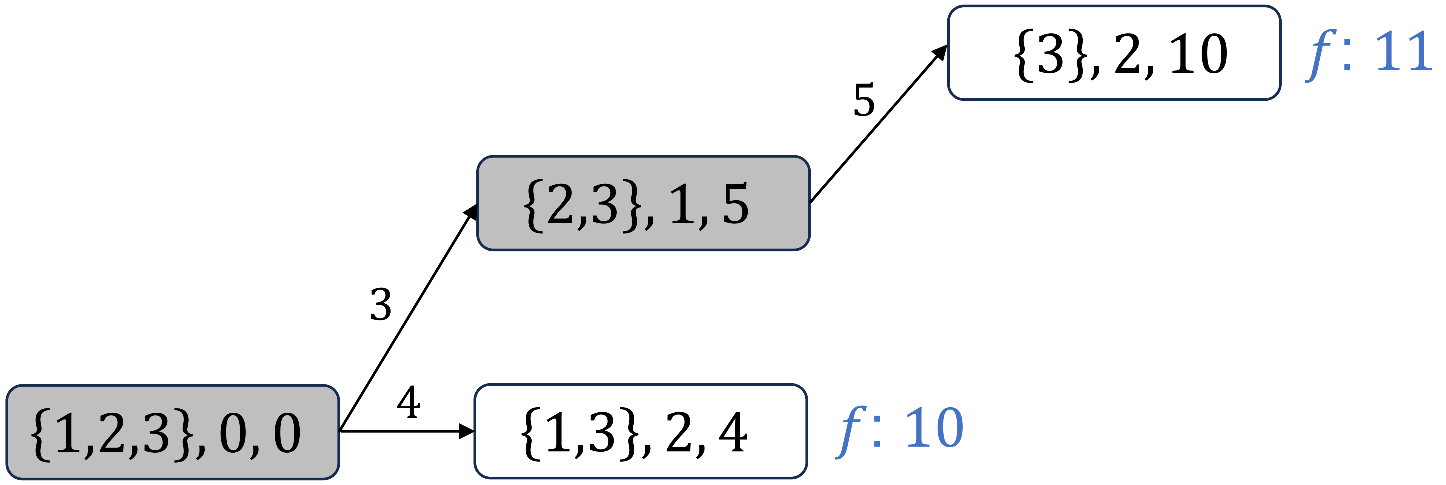
$$V(U,i,t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

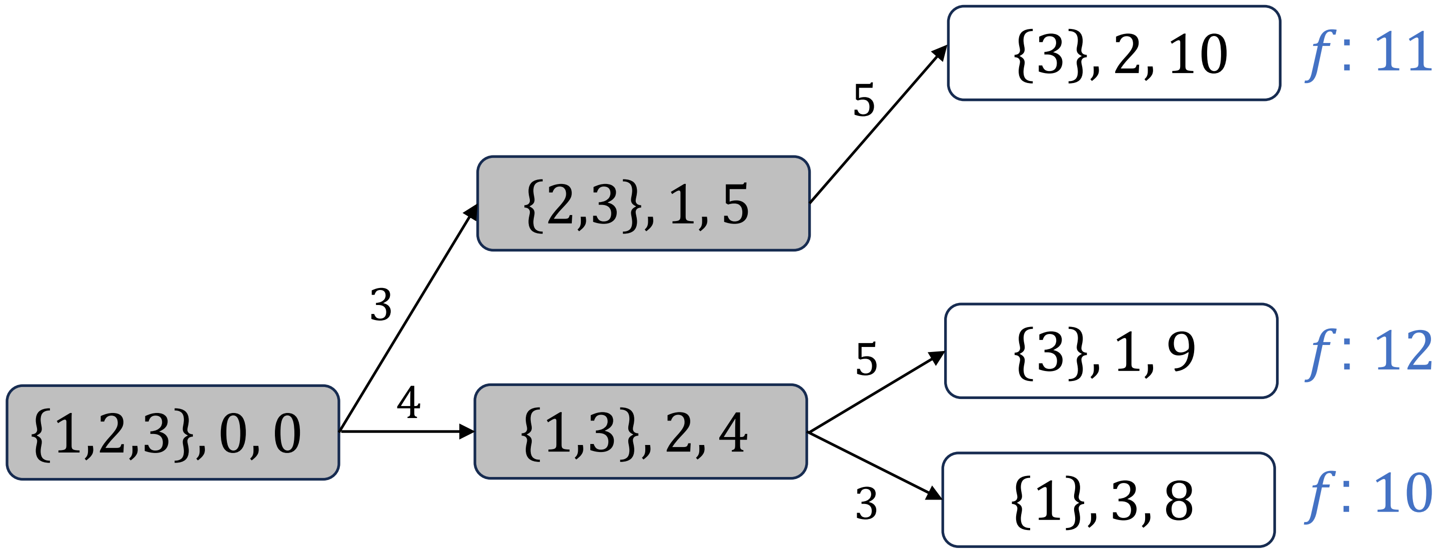
$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

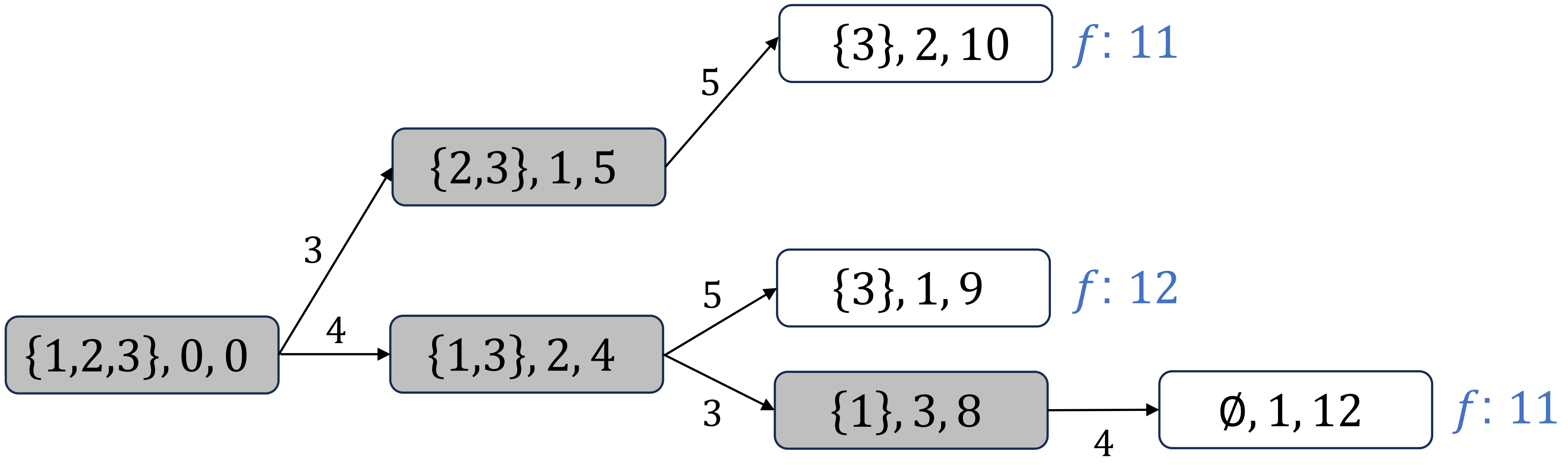
$$V(U,i,t)=\begin{cases}\min_{j\in U:t+c_{ij}\le b_j}\boxed{c_{ij}+V\left(U\setminus\{j\},j,\max\{t+c_{ij},a_j\}\right)} & \text{if } U\neq\emptyset \\ c_{i0}+V(U,0,t+c_{i0}) & \text{if } U=\emptyset, i\neq 0 \\ 0 & \text{if } U=\emptyset, i=0\end{cases}$$



A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

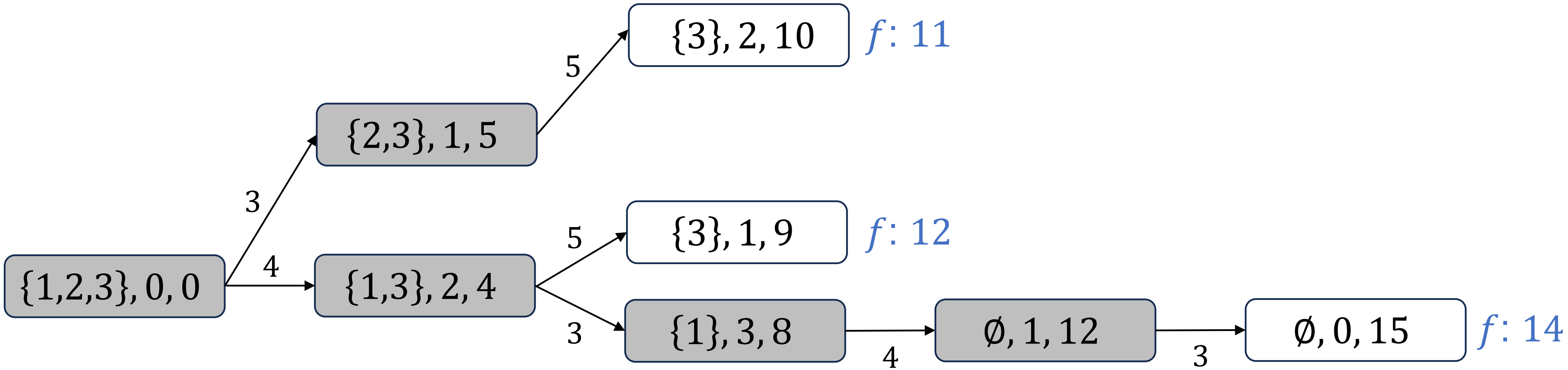
$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \le b_j} \boxed{c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right)} & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

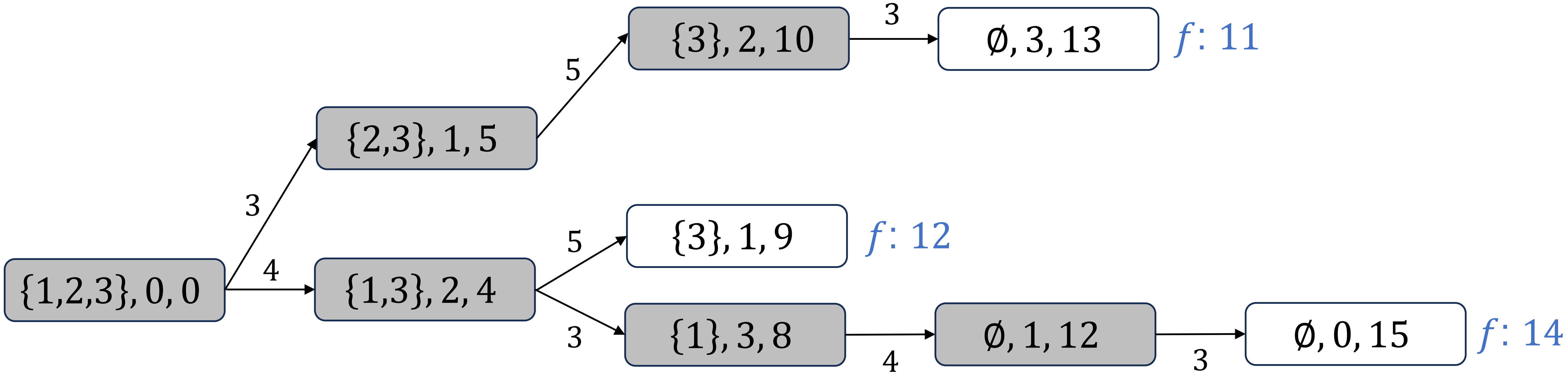
$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \le b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ \boxed{c_{i0} + V(U, 0, t + c_{i0}) \quad \text{if } U = \emptyset, i \neq 0} & \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

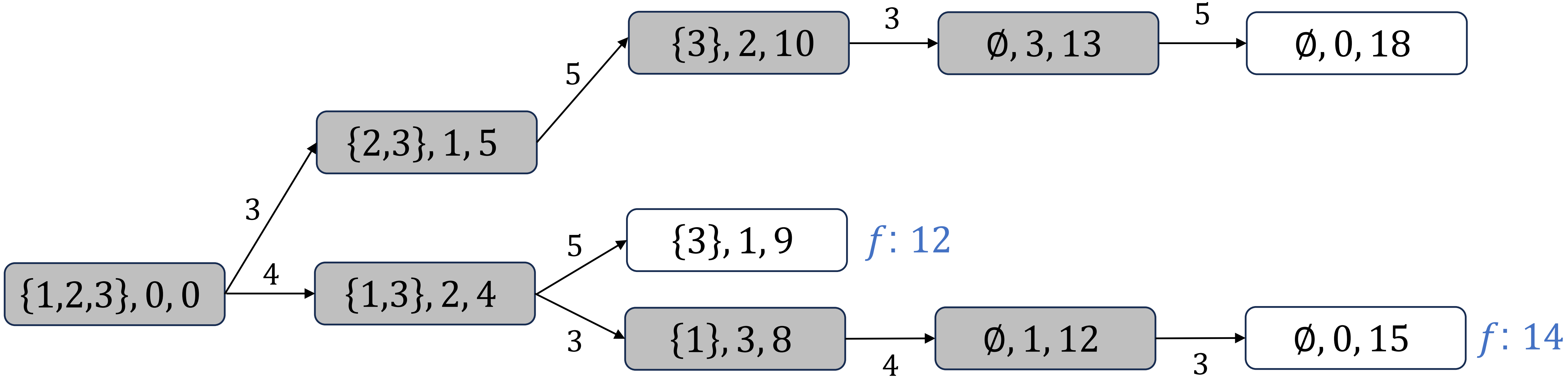
$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \leq b_j} \boxed{c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right)} & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

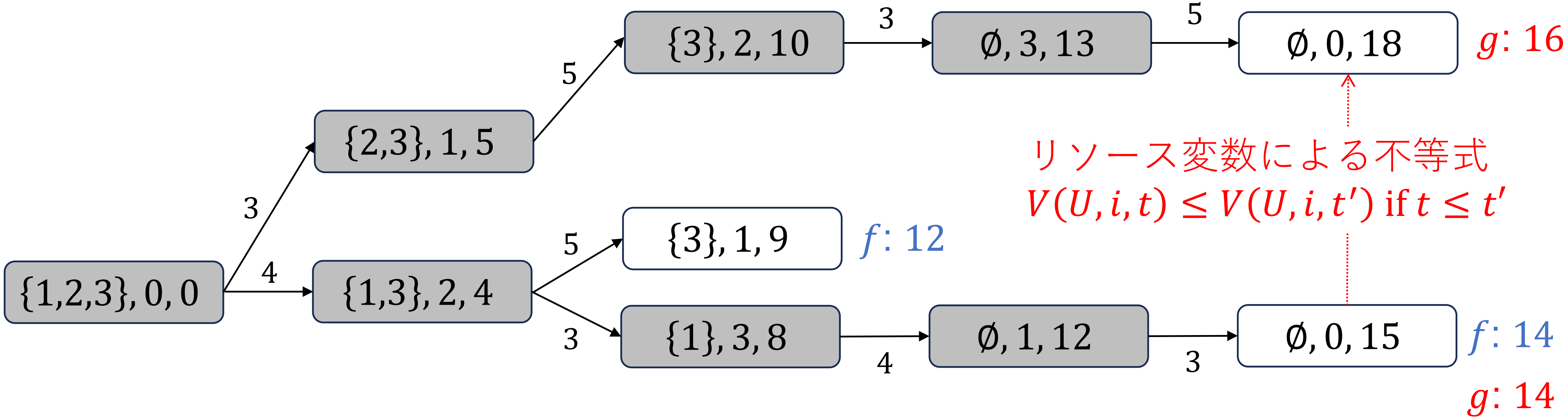
$$V(U,i,t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \le b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ \boxed{c_{i0} + V(U, 0, t + c_{i0}) \quad \text{if } U = \emptyset, i \neq 0} \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

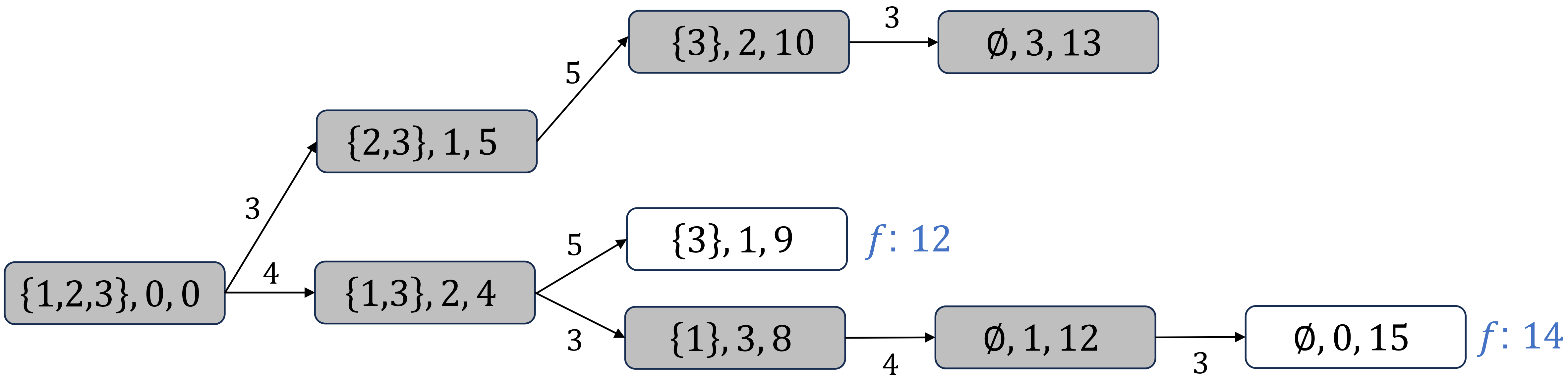
$$V(U,i,t)=\begin{cases}\min\limits_{j\in U: t+c_{ij}\le b_j} c_{ij}+V\left(U\setminus\{j\},j,\max\{t+c_{ij},a_j\}\right) & \text{if } U\neq\emptyset\\ c_{i0}+V(U,0,t+c_{i0}) & \text{if } U=\emptyset, i\neq 0\\ 0 & \text{if } U=\emptyset, i=0\end{cases}$$



A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

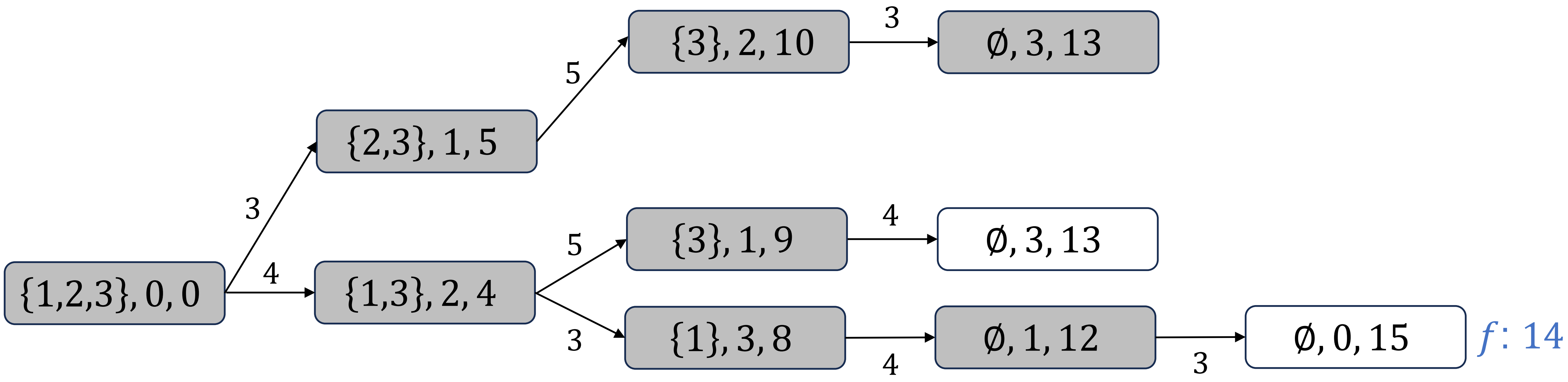
$$V(U,i,t) = \begin{cases} \min\limits_{j \in U : t + c_{ij} \le b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \ne \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \ne 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

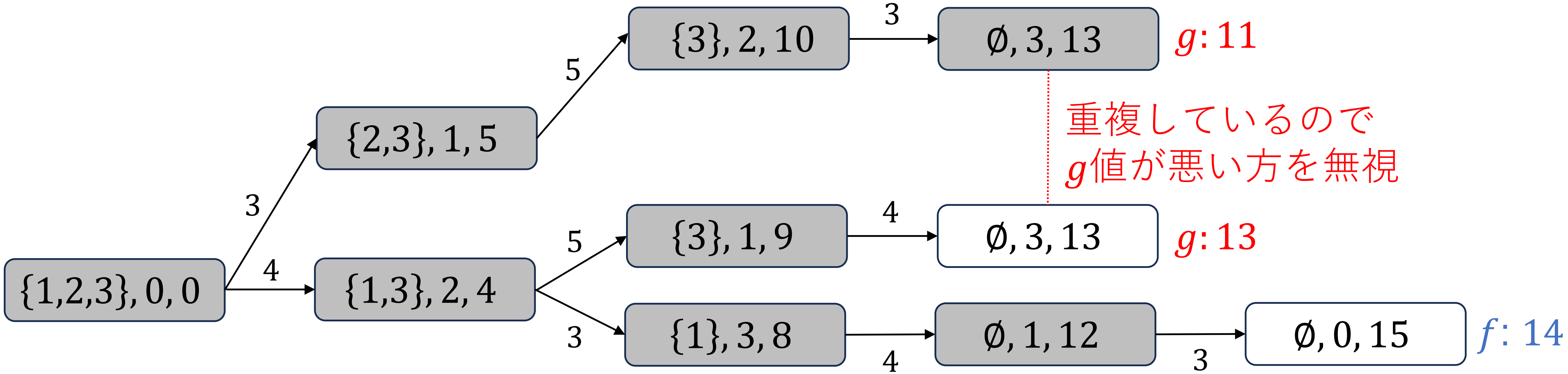
$$V(U,i,t)=\begin{cases}\min\limits_{j\in U:t+c_{ij}\le b_j}\boxed{c_{ij}+V\left(U\setminus\{j\},j,\max\{t+c_{ij},a_j\}\right)} & \text{if } U\neq\emptyset\\ c_{i0}+V(U,0,t+c_{i0}) & \text{if } U=\emptyset, i\neq 0\\ 0 & \text{if } U=\emptyset, i=0\end{cases}$$



A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

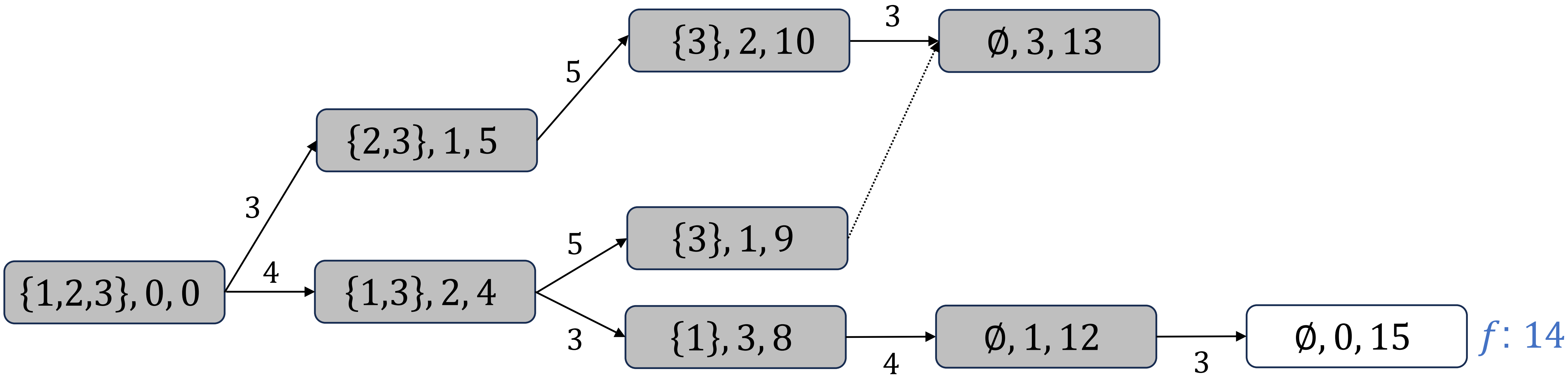
$$V(U,i,t)=\begin{cases}\min\limits_{j\in U:t+c_{ij}\le b_j} c_{ij}+V\left(U\setminus\{j\},j,\max\{t+c_{ij},a_j\}\right) & \text{if } U\neq\emptyset\\ c_{i0}+V(U,0,t+c_{i0}) & \text{if } U=\emptyset, i\neq 0\\ 0 & \text{if } U=\emptyset, i=0\end{cases}$$



A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

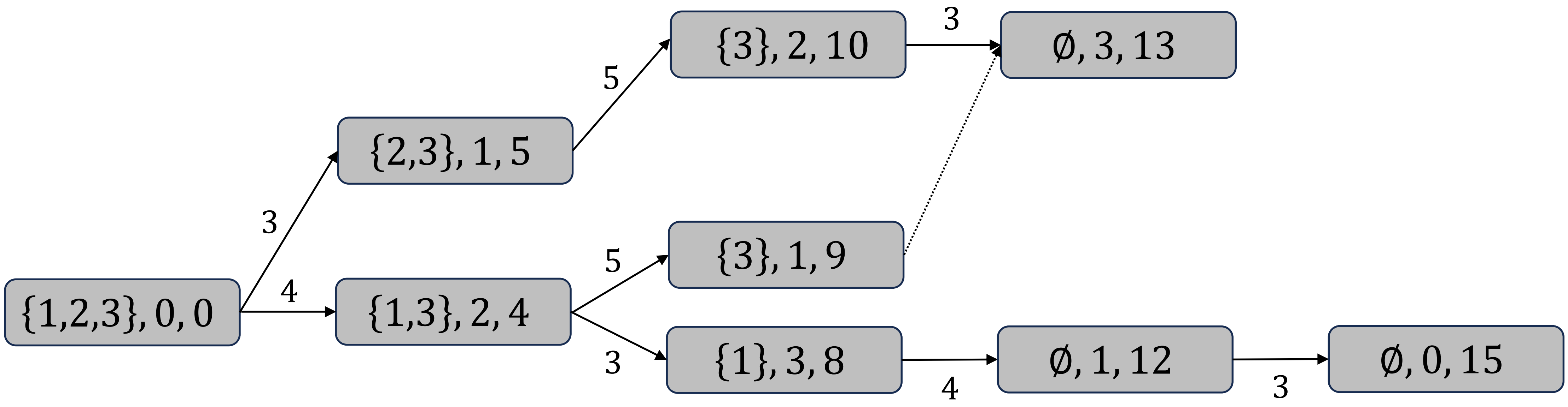
$$V(U,i,t)=\begin{cases}\min\limits_{j\in U:t+c_{ij}\leq b_j} c_{ij}+V\left(U\setminus\{j\},j,\max\{t+c_{ij},a_j\}\right) & \text{if } U\neq\emptyset \\ c_{i0}+V(U,0,t+c_{i0}) & \text{if } U=\emptyset, i\neq 0 \\ 0 & \text{if } U=\emptyset, i=0\end{cases}$$



A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

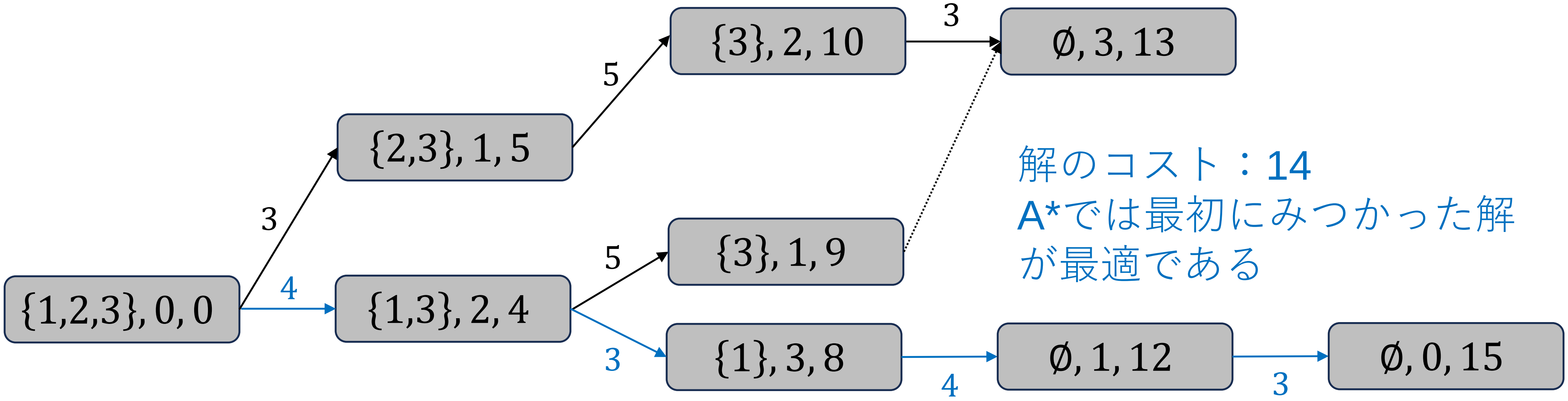
$$V(U,i,t) = \begin{cases} \min_{j \in U : t + c_{ij} \le b_j} c_{ij} + V\left(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}\right) & \text{if } U \neq \emptyset \\ c_{i0} + V(U, 0, t + c_{i0}) & \text{if } U = \emptyset, i \neq 0 \\ 0 & \text{if } U = \emptyset, i = 0 \end{cases}$$



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

CAASDy：A*でDPを解く

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

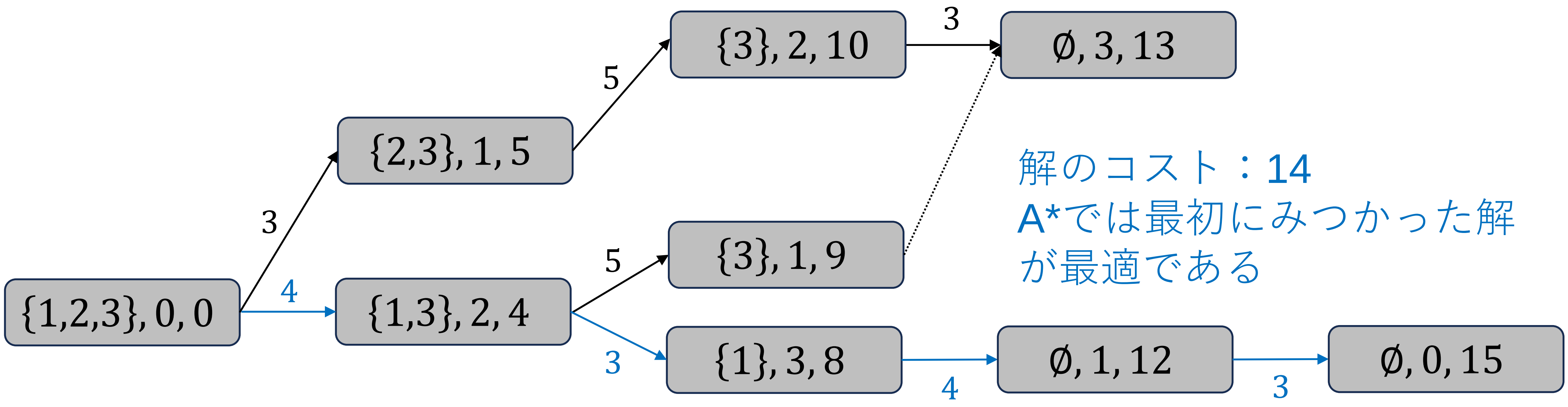


解のコスト：14
A*では最初にみつかった解が最適である

A*：$f(S)=g(S)+h(S)$ を最小化する状態 S を展開する

CAASDyの問題点

- 最適解をみつけるまで何の解もみつからない
- 状態を保存するのにメモリを多く消費する



A*：$f(S) = g(S) + h(S)$ を最小化する状態 S を展開する

ビームサーチで高速に（非最適）解を見つける

層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する

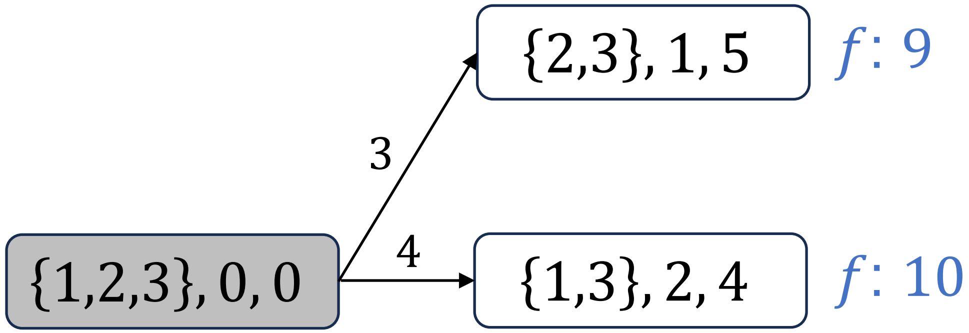
$b = 2$

$\boxed{\{1,2,3\}, 0, 0}$ $f: 9$

ビームサーチで高速に（非最適）解を見つける

層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する

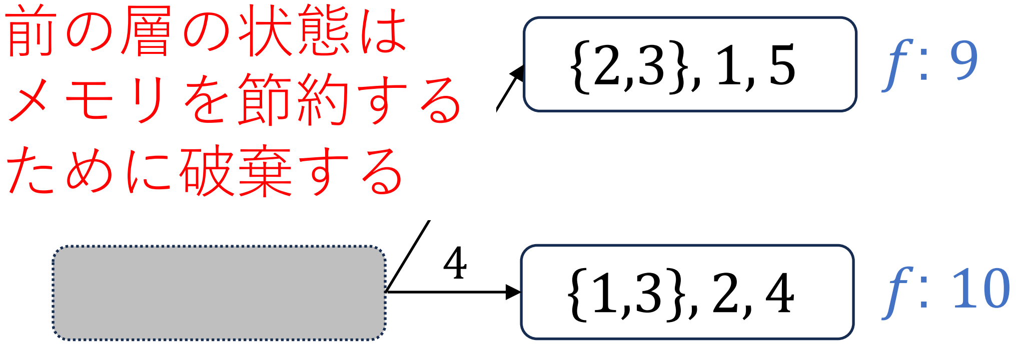
$b = 2$



ビームサーチで高速に（非最適）解を見つける

層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する

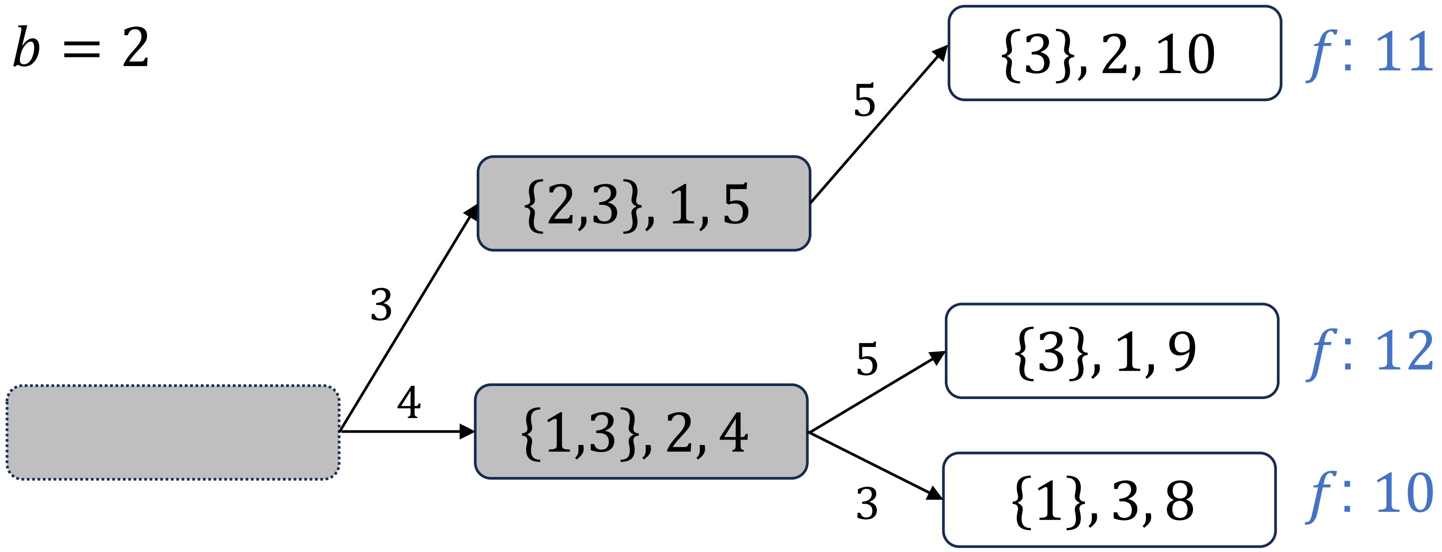
$b = 2$



ビームサーチで高速に（非最適）解を見つける

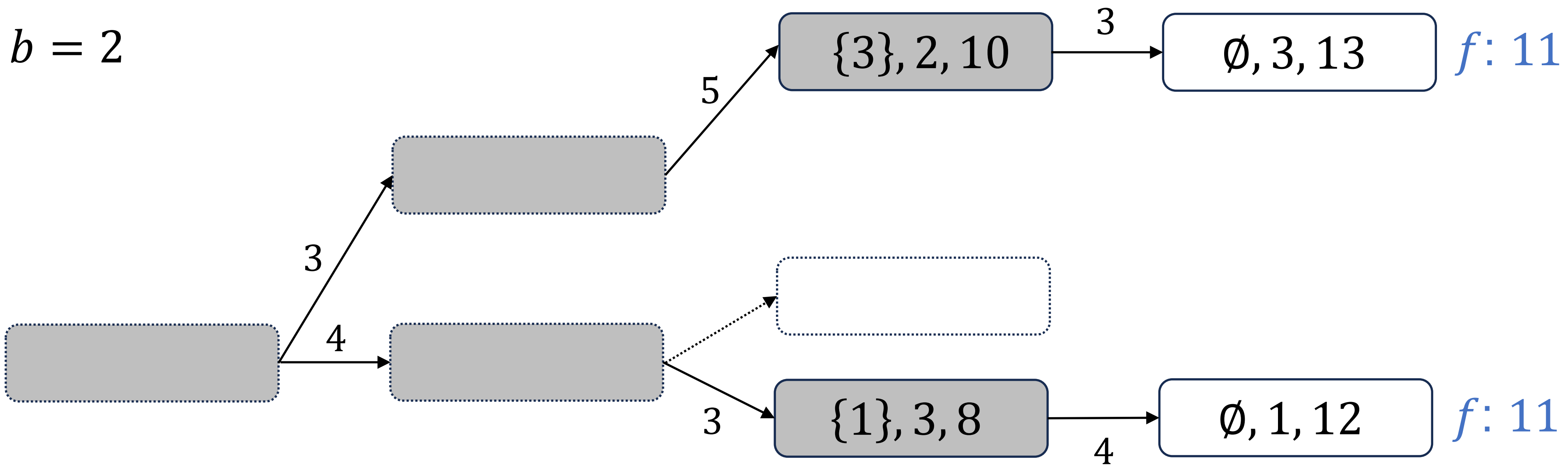
層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する

$b = 2$



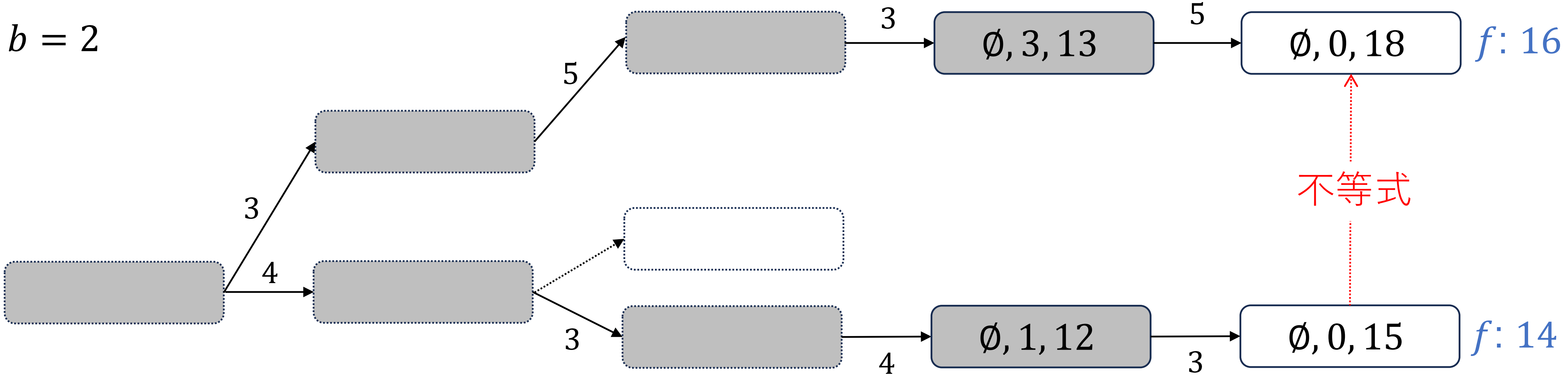
ビームサーチで高速に（非最適）解を見つける

層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する



ビームサーチで高速に（非最適）解を見つける

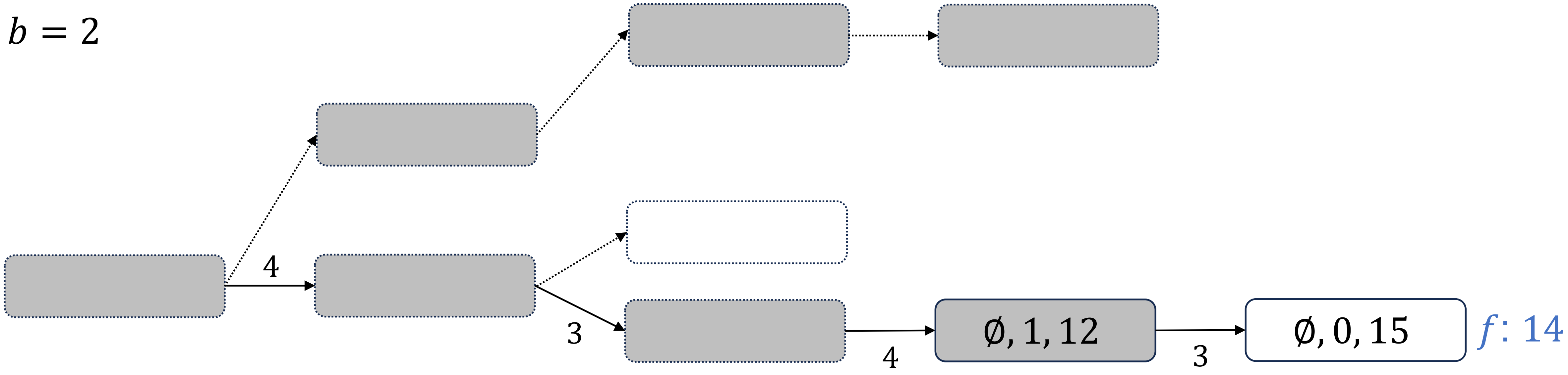
層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する



ビームサーチで高速に（非最適）解を見つける

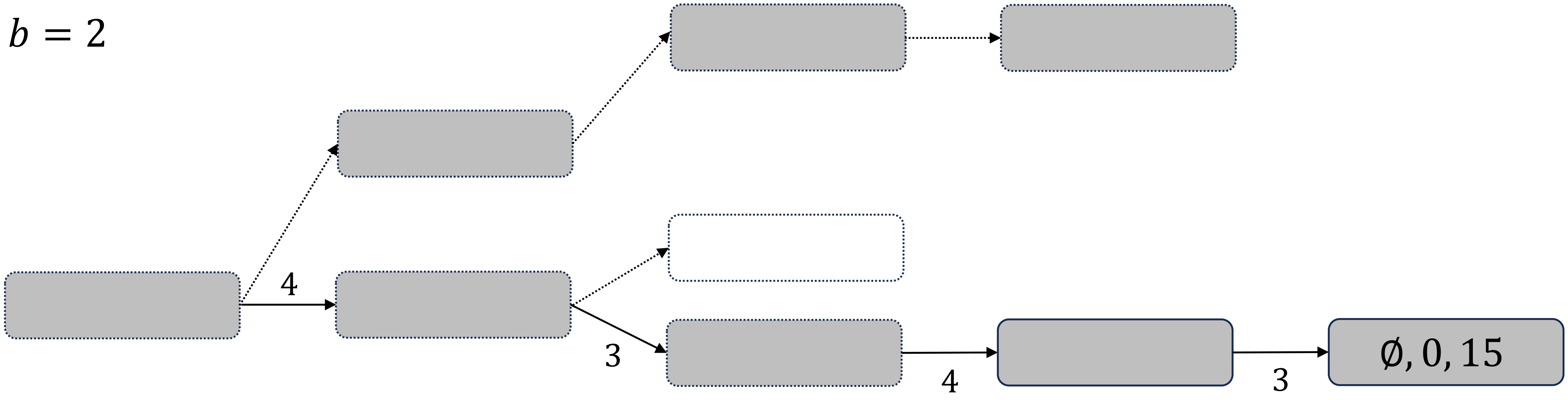
層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する

$b = 2$



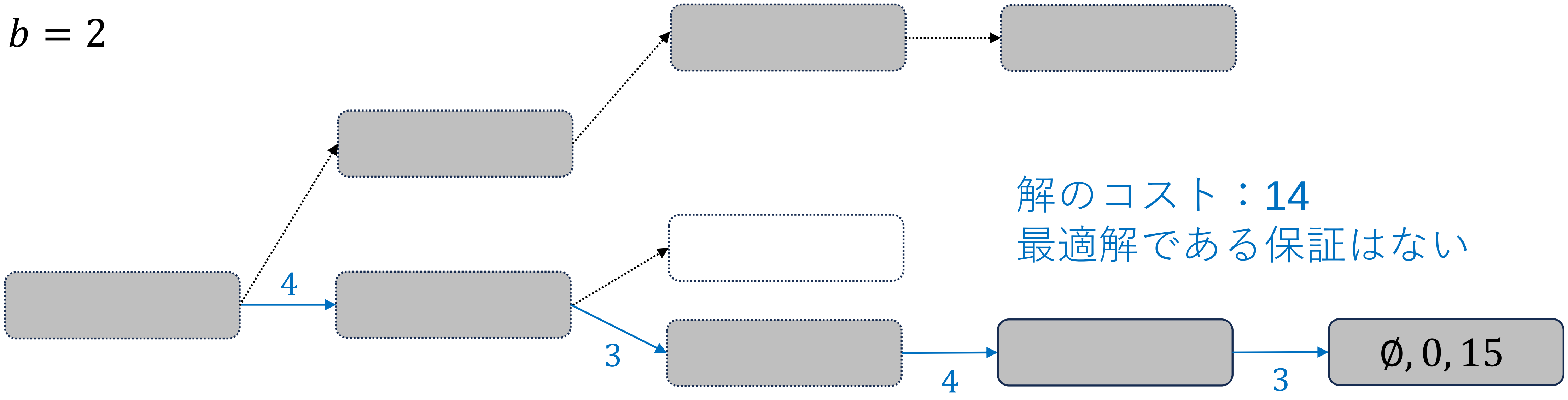
ビームサーチで高速に（非最適）解を見つける

層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する



ビームサーチで高速に（非最適）解を見つける

層ごとに $f(S) = g(S) + h(S)$ を最小化する b 状態を展開する



Complete Anytime Beam Search (CABS)

bを倍々に増やしながらビームサーチを繰り返す

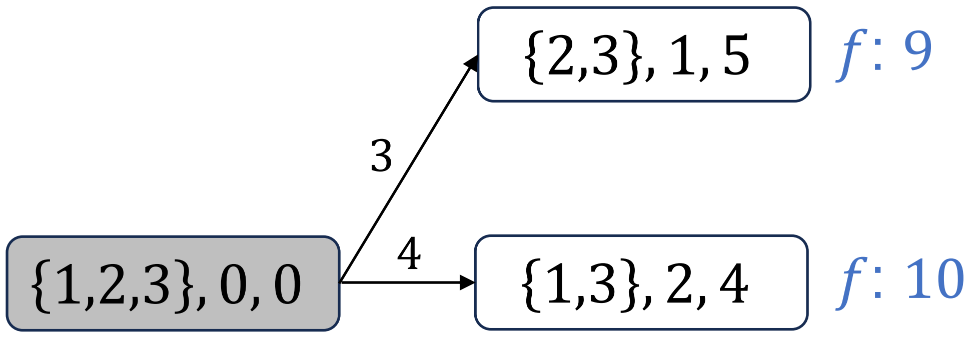
$b = 4$、現在の解のコスト：14

$\boxed{\{1,2,3\}, 0, 0}$ f: 9

Complete Anytime Beam Search (CABS)

bを倍々に増やしながらビームサーチを繰り返す

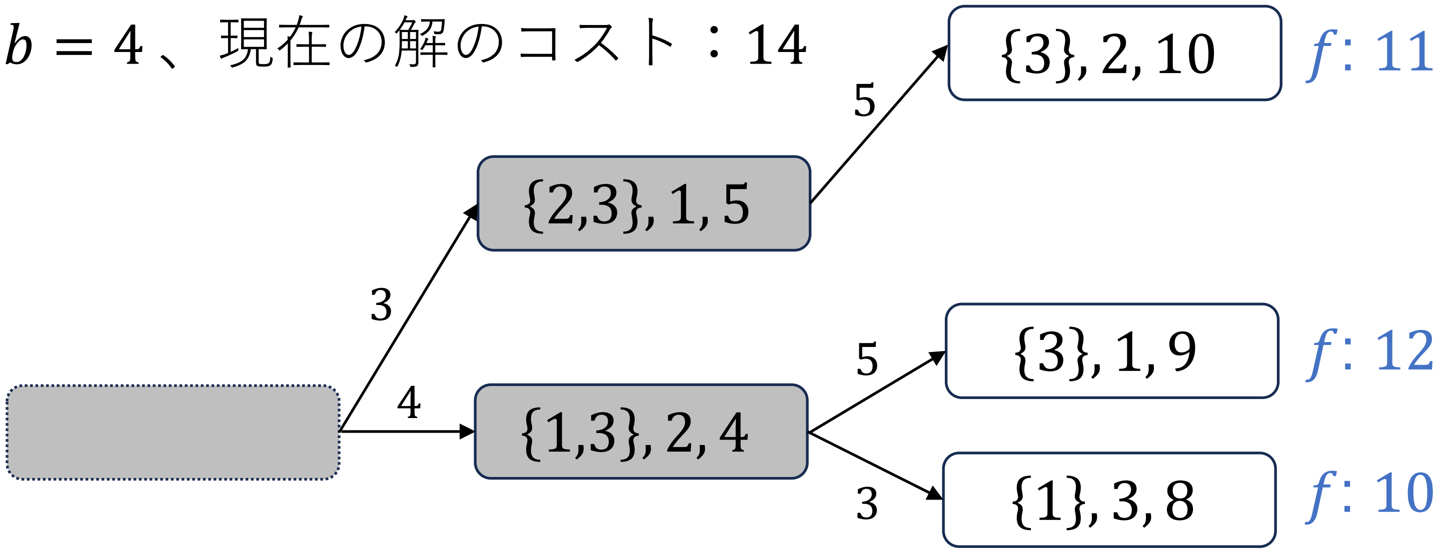
$b = 4$、現在の解のコスト：14



Complete Anytime Beam Search (CABS)

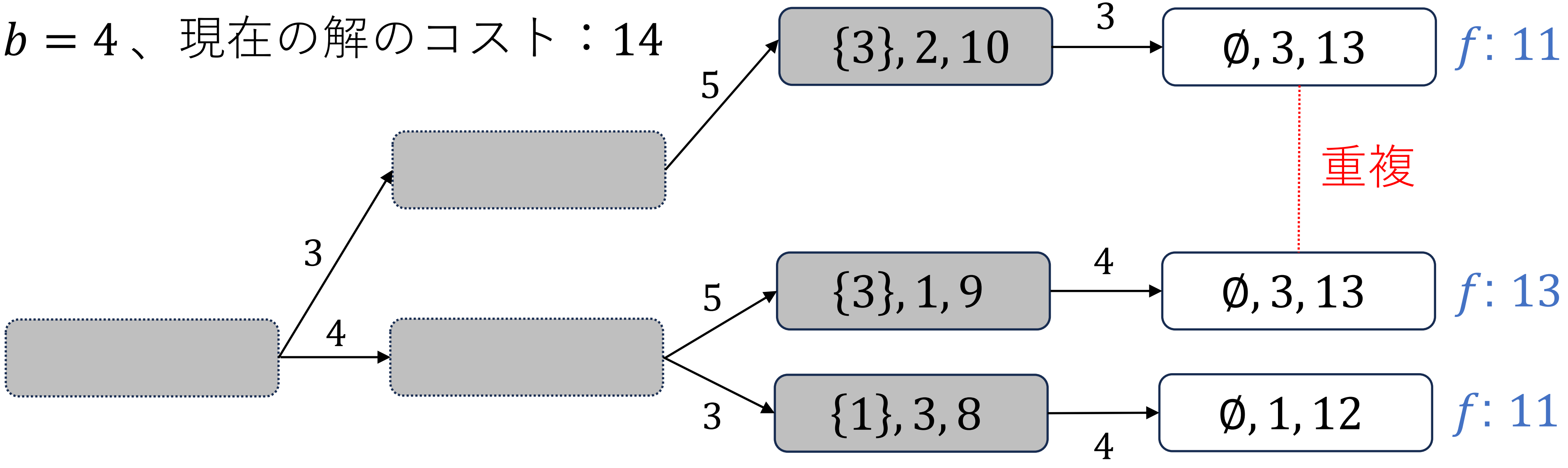
bを倍々に増やしながらビームサーチを繰り返す

$b = 4$、現在の解のコスト：14



Complete Anytime Beam Search (CABS)

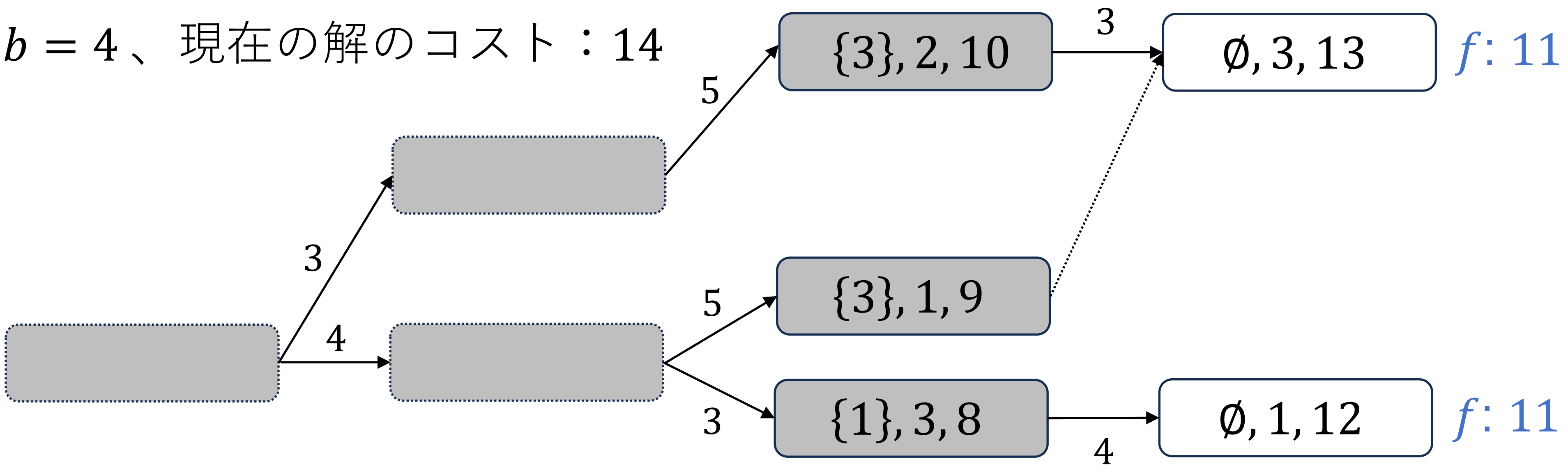
bを倍々に増やしながらビームサーチを繰り返す



Complete Anytime Beam Search (CABS)

bを倍々に増やしながらビームサーチを繰り返す

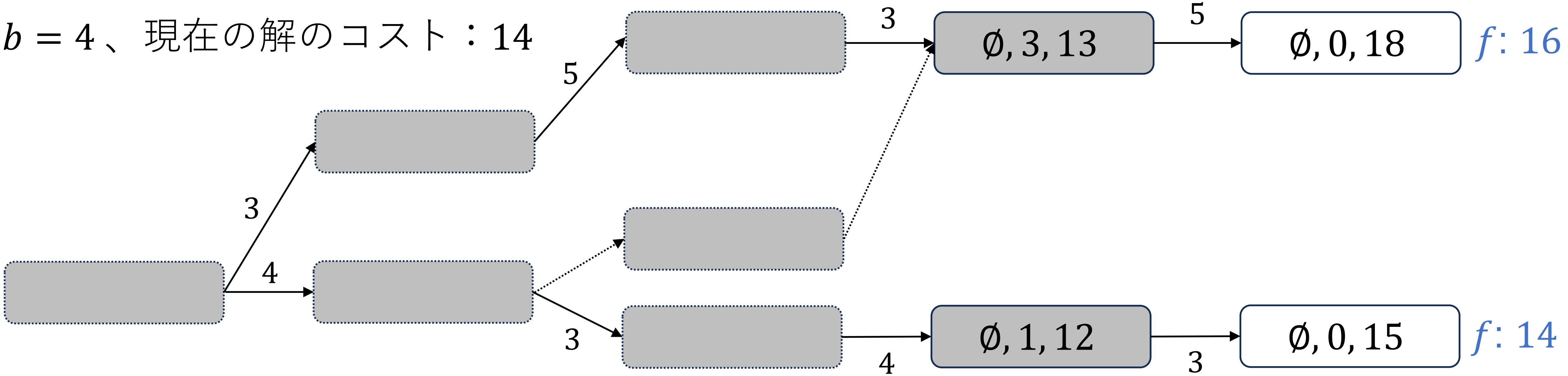
$b = 4$、現在の解のコスト：14



Complete Anytime Beam Search (CABS)

bを倍々に増やしながらビームサーチを繰り返す

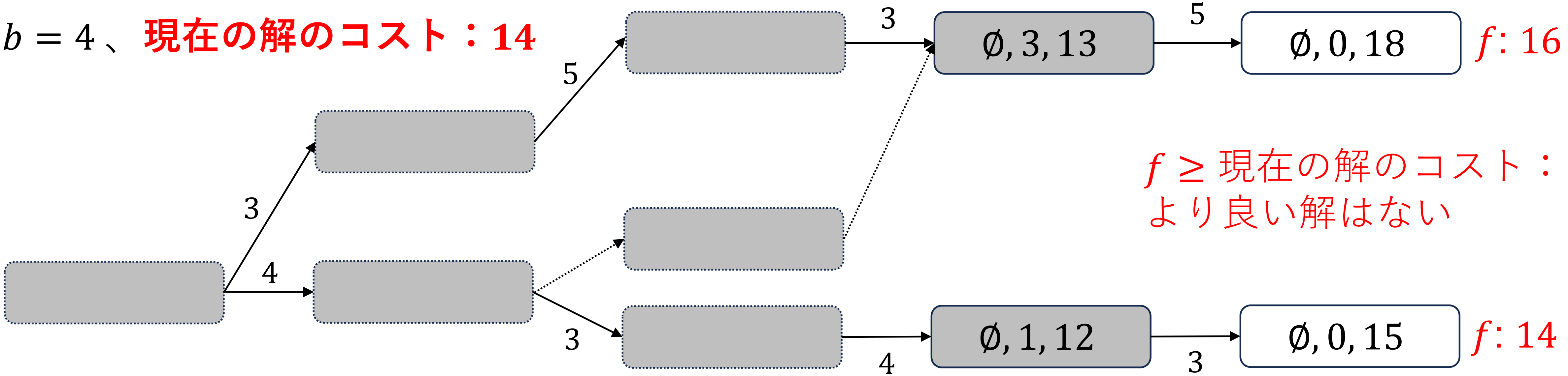
$b = 4$、現在の解のコスト：14



Complete Anytime Beam Search (CABS)

bを倍々に増やしながらビームサーチを繰り返す

$b = 4$、**現在の解のコスト：14**

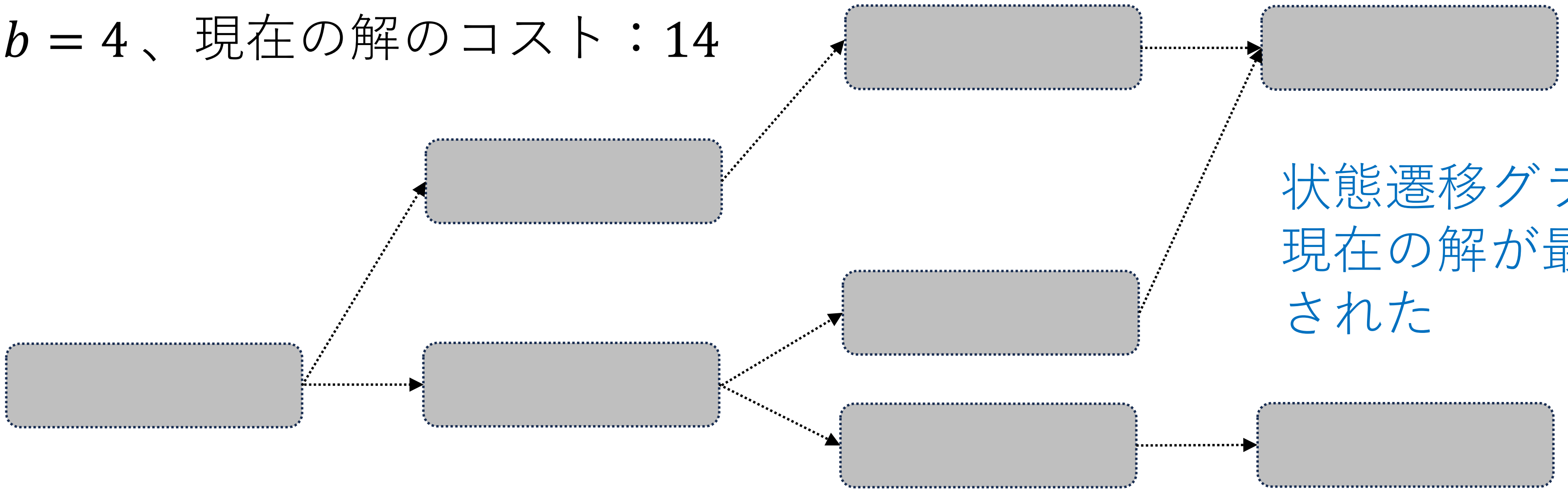


$f \geq$ 現在の解のコスト：
より良い解はない

Complete Anytime Beam Search (CABS)

bを倍々に増やしながらビームサーチを繰り返す

$b=4$、現在の解のコスト：14



状態遷移グラフを探索し尽し、
現在の解が最適であると証明
された

余談：DP、分枝限定法、ヒューリスティック探索

状態（ノード）の保存による重複排除+上界と下界を使った枝狩りを行う手法は様々な名前で呼ばれてきた

- DPアルゴリズムに下界と上界を使った枝狩りを入れる
  => bounded DP
- 分枝限定法にノードをメモリに保存し重複を避ける仕組みを入れる
  => test for dominance; subproblem dominance;
  branch, bound, and remember; branch-and-memorize; caching;
  nogood recording

茨木俊秀先生が1978年にDP、分枝限定法、ヒューリスティック探索を統一的に扱う理論的枠組みを提案 [Ibaraki 1978]

実験結果

最適に解けた問題の数（30分、8GB）

問題	説明	MIP	CP	CAASDy	CABS
TSPTW (340)	時間枠付きTSP	224	47	257	**259**
CVRP (207)	配車計画問題	**28**	0	6	6
m-PDTSP (1180)	配送計画TSP	940	**1049**	952	1035
OPTW (144)	オリエンテーリング	16	49	**64**	**64**
MDKP (277)	多次元ナップザック	168	6	4	5
Bin Packing (1615)	ビンパッキング	1160	**1234**	922	1167
SALBP-1 (2100)	組立ライン最適化	1431	1584	1657	**1802**
$1\|\|\Sigma w_i T_i$ (375)	ジョブスケジュール	107	150	270	**288**
Talent Scheduling (1000)	撮影スケジュール	0	0	207	**239**
MOSP (570)	生産順序最適化	241	437	483	**527**
Graph-Clear (135)	ロボット警備最適化	26	4	78	**103**

MIP: Gurobi 11.0.2, CP: IBM ILOG CP Optimizer 22.1.0

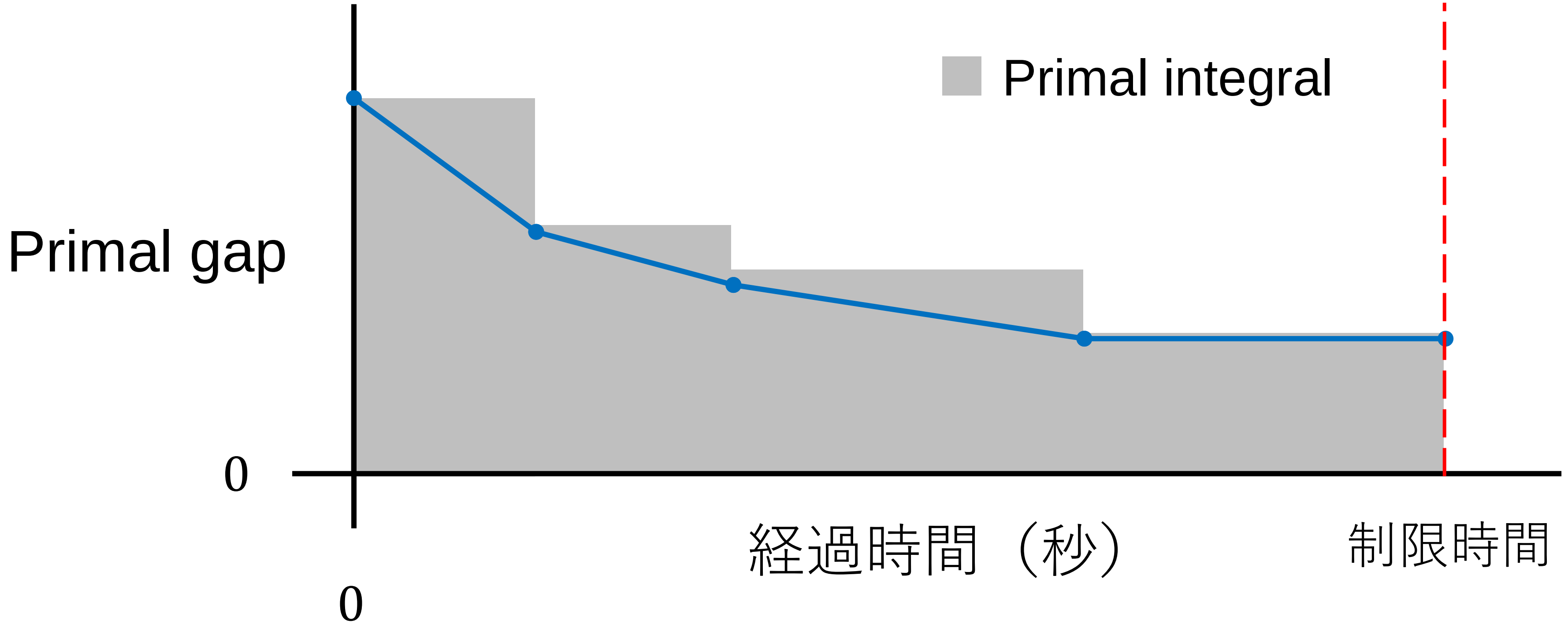
平均の双対ギャップ（30分、8GB）

問題	説明	MIP	CP	CAASDy	CABS
TSPTW (340)	時間枠付きTSP	0.220	0.716	0.244	**0.115**
CVRP (207)	配車計画問題	0.865	0.987	0.971	**0.691**
m-PDTSP (1180)	配送計画TSP	0.184	**0.110**	0.275	0.160
OPTW (144)	オリエンテーリング	0.665	0.289	0.556	**0.270**
MDKP (277)	多次元ナップザック	**0.001**	0.422	0.986	0.468
Bin Packing (1615)	ビンパッキング	0.044	**0.004**	0.429	0.005
SALBP-1 (2100)	組立ライン最適化	0.271	0.017	0.211	**0.006**
$1\|\|\Sigma w_i T_i$ (375)	ジョブスケジュール	0.498	0.371	0.280	**0.225**
Talent Scheduling (1000)	撮影スケジュール	0.887	0.951	0.793	**0.170**
MOSP (570)	生産順序最適化	0.317	0.193	0.153	**0.020**
Graph-Clear (135)	ロボット警備最適化	0.447	0.456	0.422	**0.061**

$$|\text{解のコスト} - \text{双対限界}| / \max\{|\text{解のコスト}|, |\text{双対限界}|\}$$

Primal Integral：解のコストと

Primal gap：$\dfrac{|\text{解のコスト} - \text{知られている最良コスト}|}{\max\{|\text{解のコスト}|, |\text{知られている最良コスト}|\}}$ (解なしは 1)



平均のPrimal Integral（30分、8GB）

問題	説明	MIP	CP	CAASDy	CABS
TSPTW (340)	時間枠付きTSP	479.0	49.0	458.3	**9.3**
CVRP (207)	配車計画問題	1127.6	482.9	1748.2	**333.7**
m-PDTSP (1180)	配送計画TSP	177.6	26.0	333.5	**5.2**
OPTW (144)	オリエンテーリング	438.1	**15.6**	1018.2	58.0
MDKP (277)	多次元ナップザック	**0.7**	15.9	1773.9	201.7
Bin Packing (1615)	ビンパッキング	88.1	8.1	778.6	**5.0**
SALBP-1 (2100)	組立ライン最適化	538.8	28.5	383.9	**1.9**
$1\|\|\Sigma w_i T_i$ (375)	ジョブスケジュール	64.9	3.5	513.2	71.2
Talent Scheduling (1000)	撮影スケジュール	106.1	**18.9**	1435.1	25.4
MOSP (570)	生産順序最適化	95.2	**13.0**	275.5	**0.3**
Graph-Clear (135)	ロボット警備最適化	334.9	83.5	764.0	**0.4**

MIP: Gurobi 11.0.2, CP: IBM ILO CP Optimizer 22.1.0

余談：制約プログラミング（CP）とは？

制約充足や組合せ最適化に有用なプログラミング言語・汎用ソルバ
特定の部分構造に特化したglobal constraintsを提供

- 商用ソルバの例：IBM ILOG CP Optimizer
- オープンソースソルバの例：Google OR-Tools CP-SAT
  https://developers.google.com/optimization/cp/cp_solver

$$
\begin{aligned}
\min \sum_{i=0}^{n-1} & c_{x_i x_{i+1}} \\
\text{s.t. } & x_i \neq x_j \qquad \forall i \in N, \forall j \in N \setminus \{i\} \\
& x_0 = x_n = 0 \\
& x_i \in N \setminus \{0\} \quad \forall i \in N \setminus \{0\}
\end{aligned}
$$

単純なCPモデル

$$
\begin{aligned}
\min \sum_{i=0}^{n-1} & c_{x_i x_{i+1}} \\
\text{s.t. } & \text{all\_different}(\{x_i \mid \forall i \in N\}) \\
& x_0 = x_n = 0 \\
& x_i \in N \setminus \{0\} \quad \forall i \in N \setminus \{0\}
\end{aligned}
$$

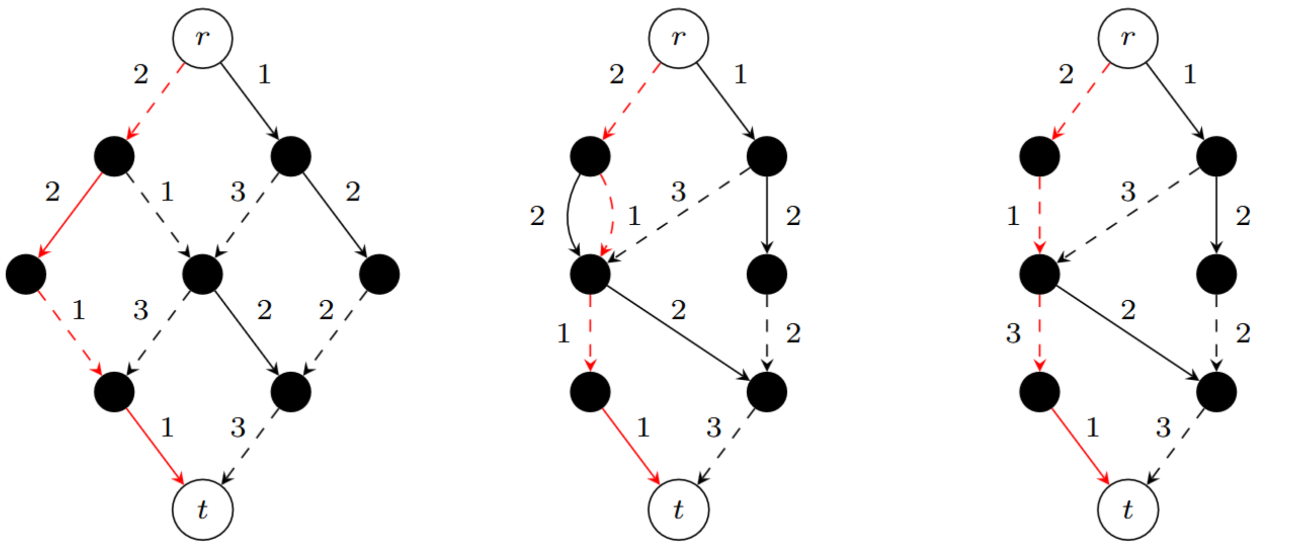
all_differentを使ったCPモデル

余談：Decision Diagram-Based Solvers

Decision Diagrams（DD）と呼ばれるデータ構造でDPモデルを表現し relaxed DDとrestricted DDで下界と上界を計算する分枝限定法ソルバ

- DDO: https://github.com/xgillard/ddo
- CODD: https://github.com/ldmbouge/CODD

ただしmerging operatorという操作を定義する必要がある

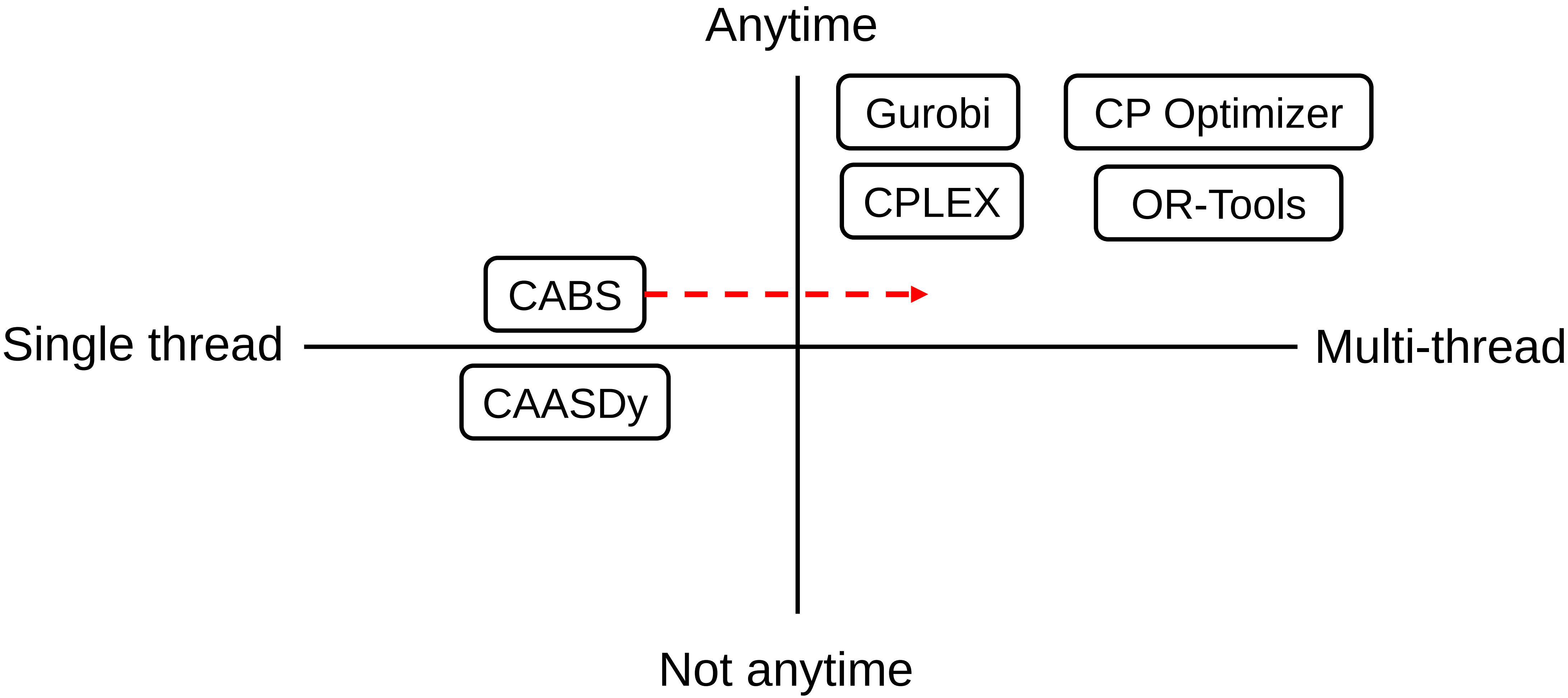


(a) Exact DD. (b) Relaxed DD. (c) Restricted DD.

並列ソルバ

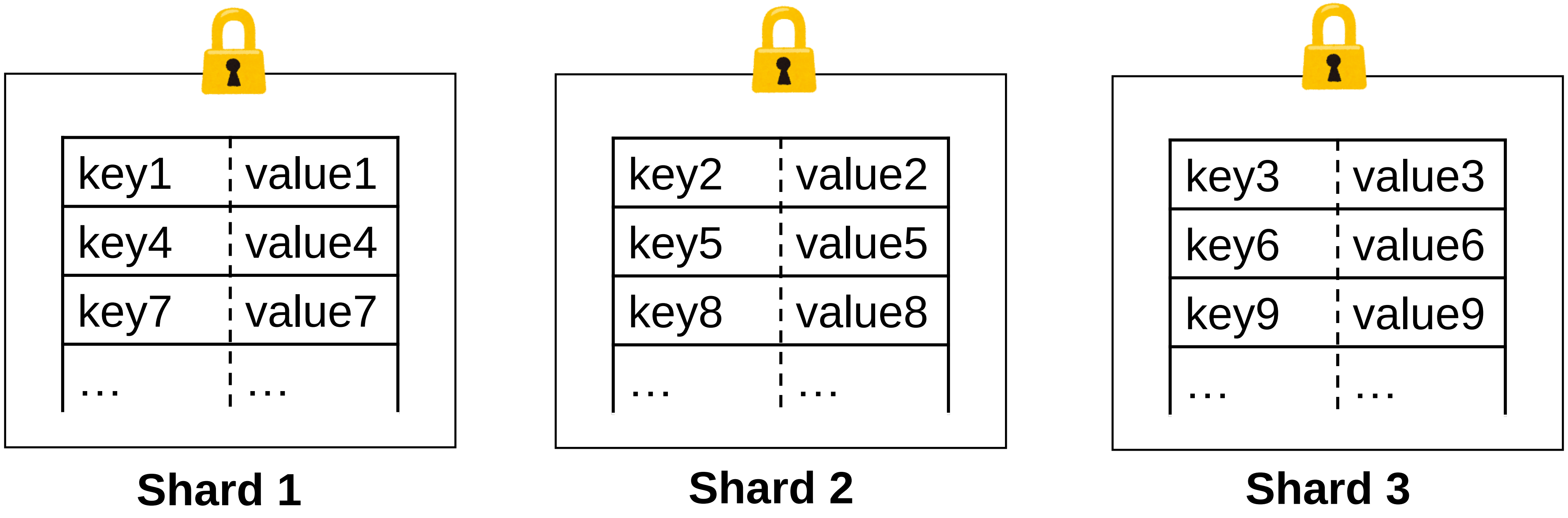
[Kuroiwa and Beck AAAI 2024]

既存の汎用ソルバの多くが複数スレッドを使える



手法1：Shared Beam Search (SBS)

- 最もよい b 状態（**並列ソート**で特定）を並列に展開する
- **並行ハッシュテーブル**で重複する状態を検知する
  複数のshardsに分かれていて、それぞれが排他ロックを持つ



Frohner+ (2023) による問題特化の並列ビームサーチに類似

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

$\{1,2,3\}, 0, 0$ $f: 9$

Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

$\boxed{\{1,2,3\}, 0, 0}$ $f: 9$

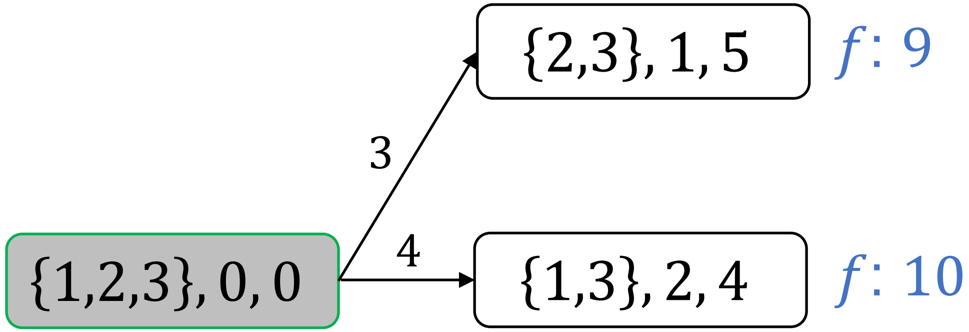
スレッド1に割り当て

Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

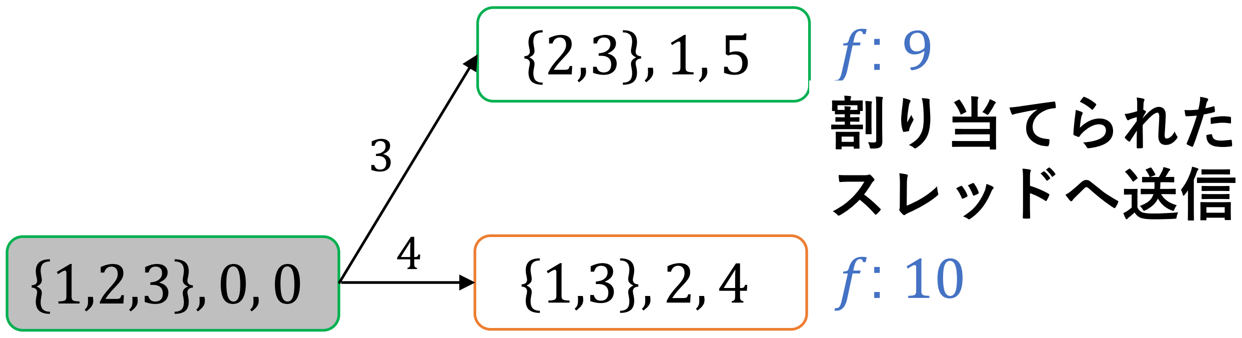


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b=4$、スレッド数 $=2$

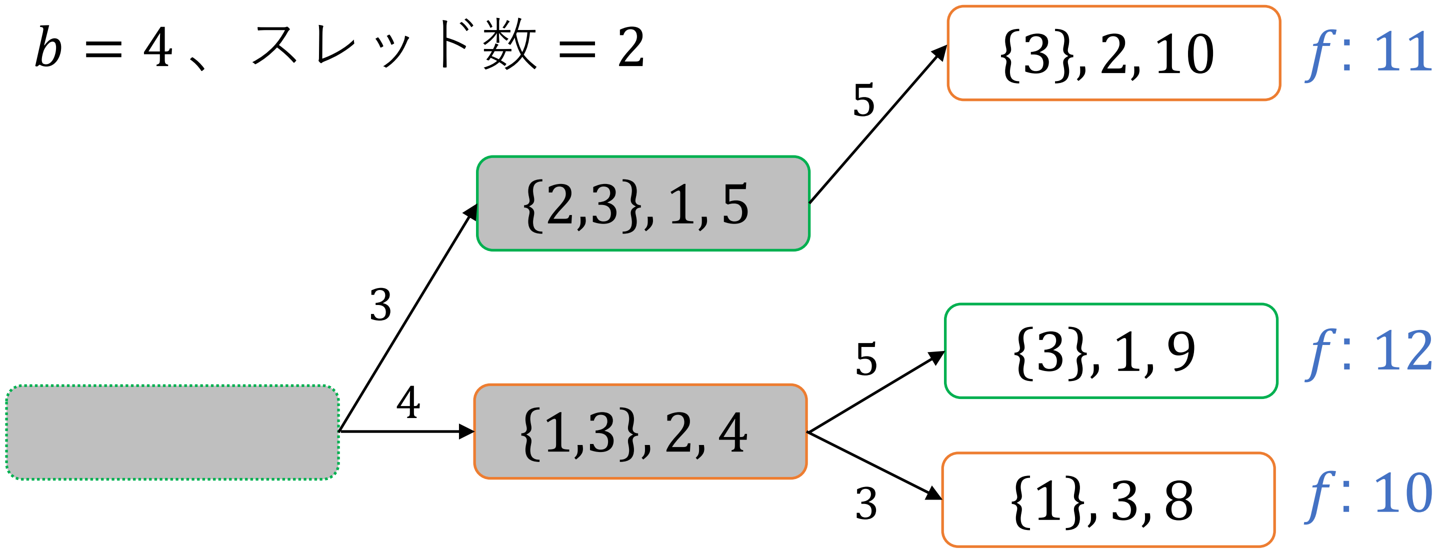


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し b/スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

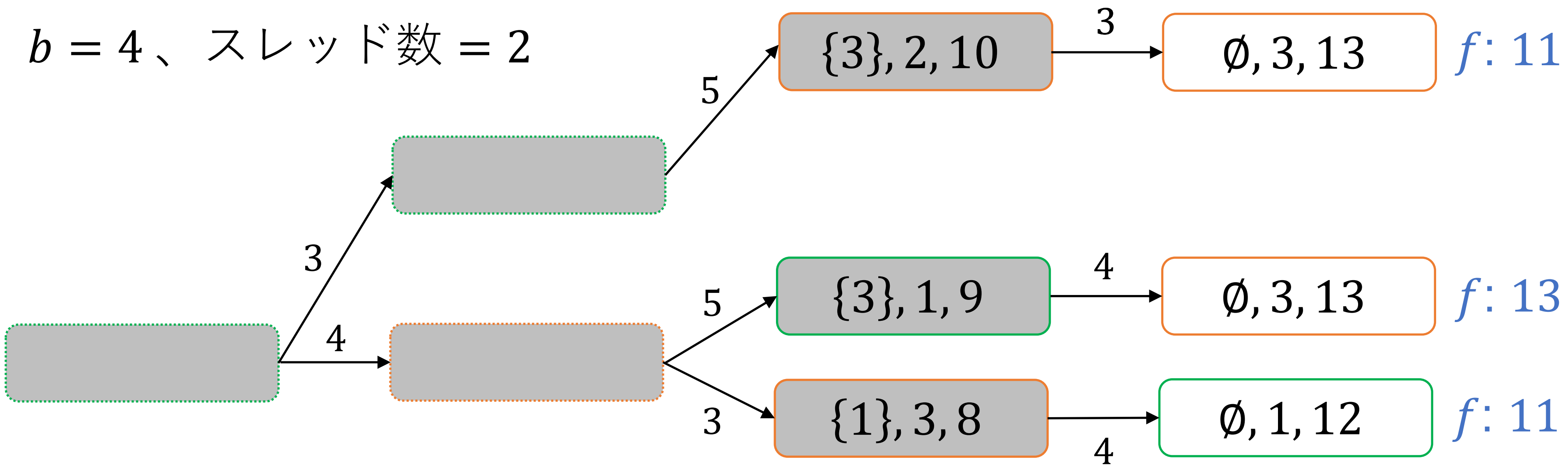


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

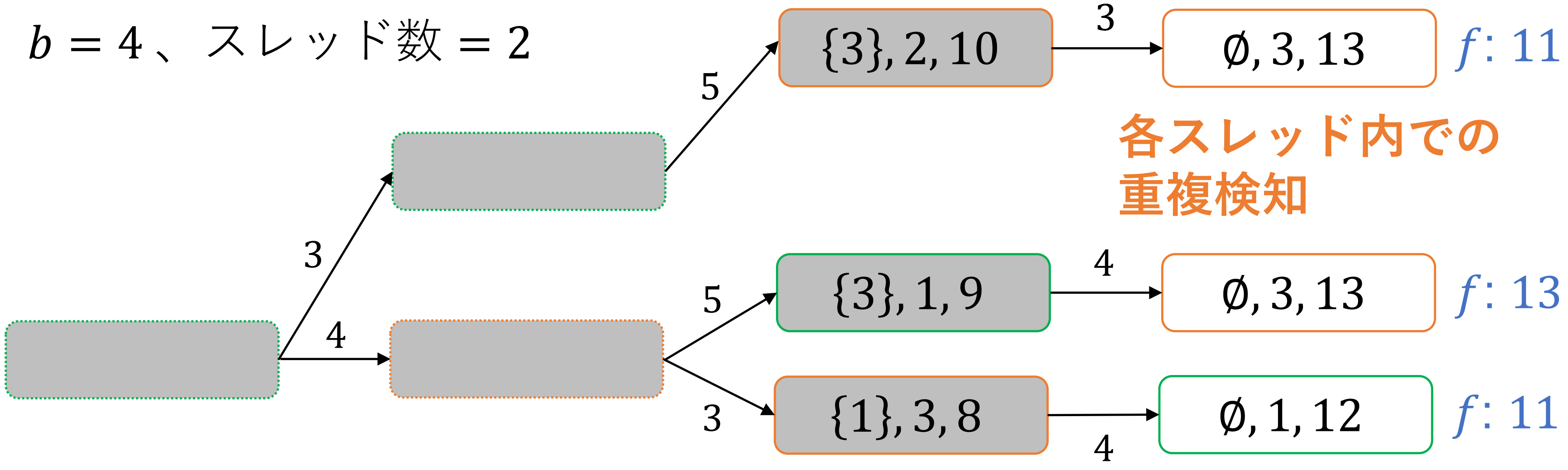


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

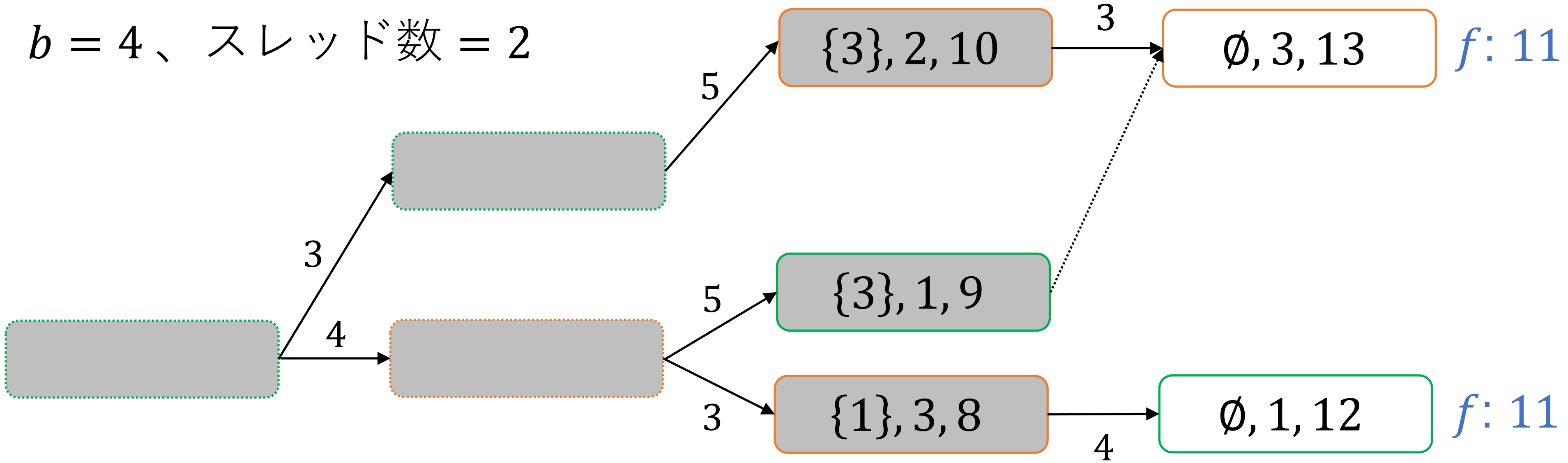


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

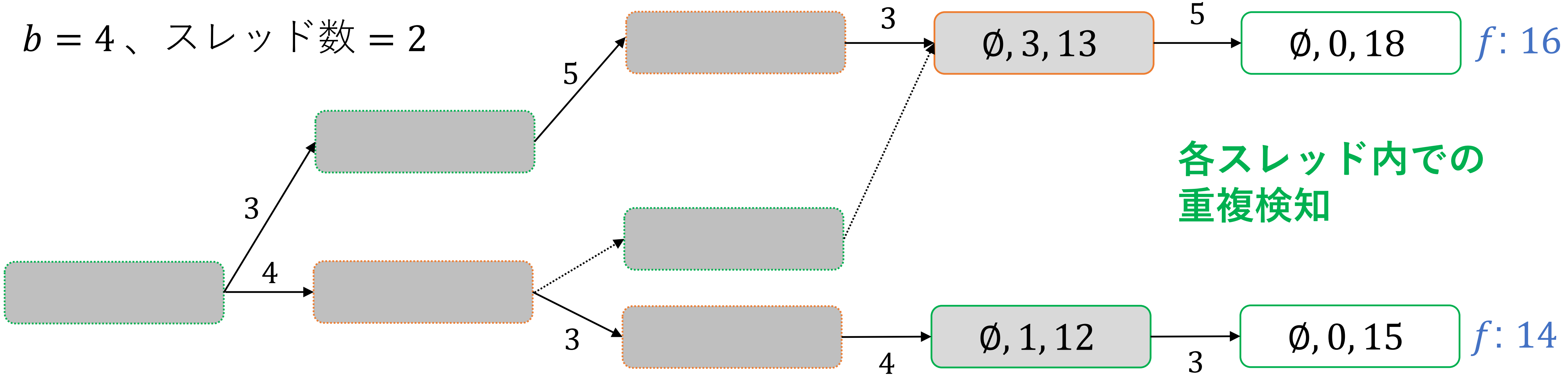
$b=4$、スレッド数 $=2$



Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

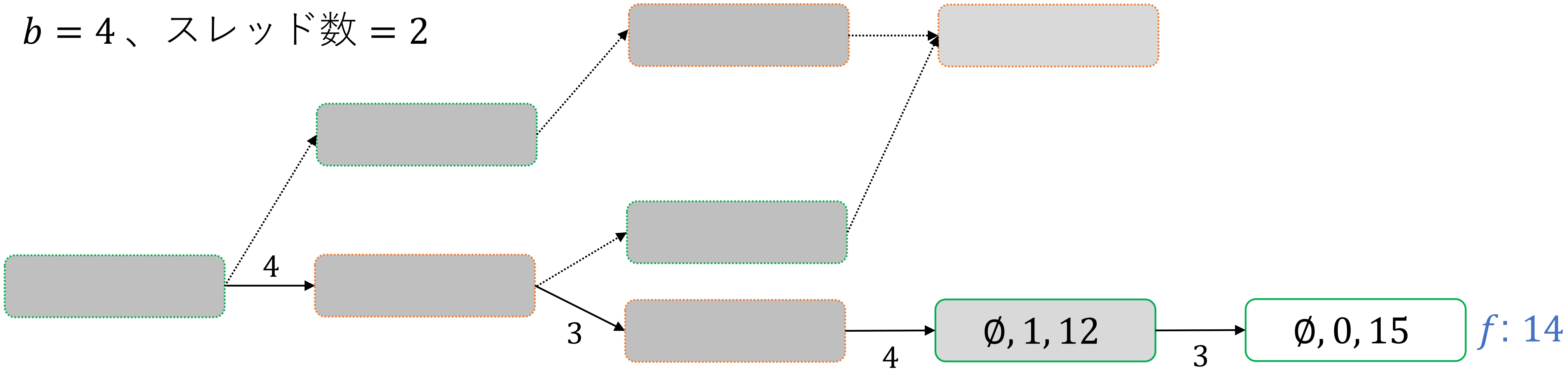


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し b/スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

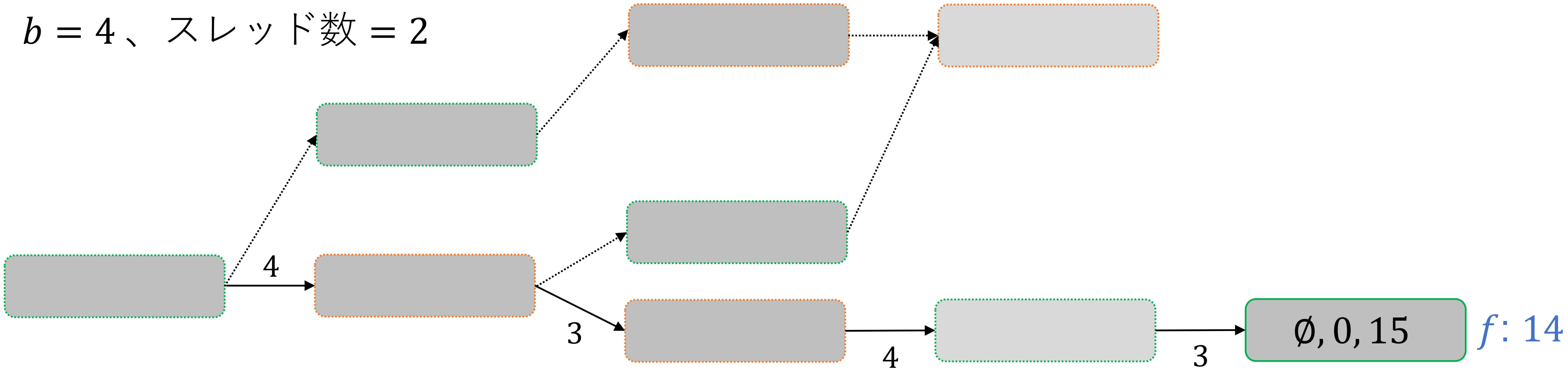


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

手法2：Hash Distributed Beam Search (HDBS)

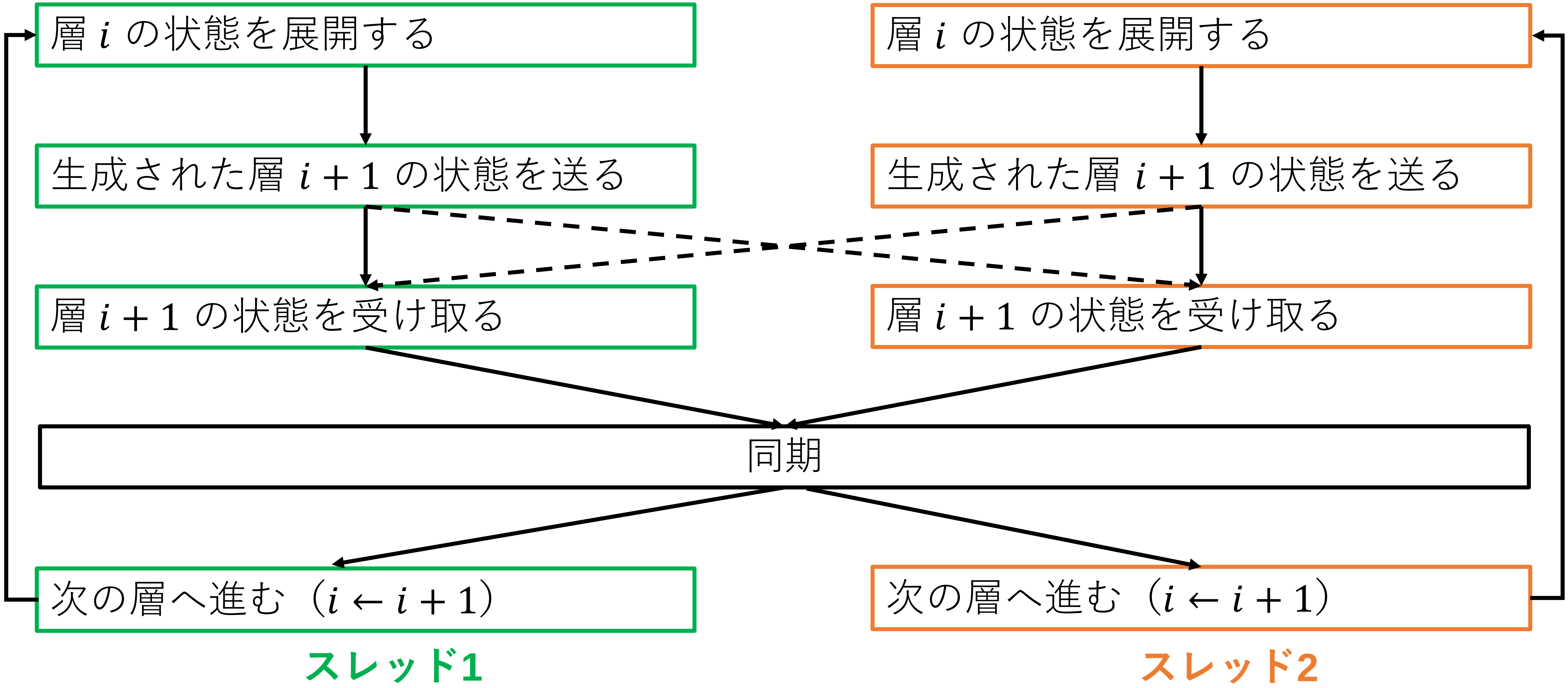
- 状態の**ハッシュ値**からスレッドを割り当て**メッセージパッシング**で送る（同じ状態は同じスレッドに送られる）
- 各スレッド内で重複する状態を検知し $b/$スレッド数 の状態を展開

$b = 4$、スレッド数 $= 2$

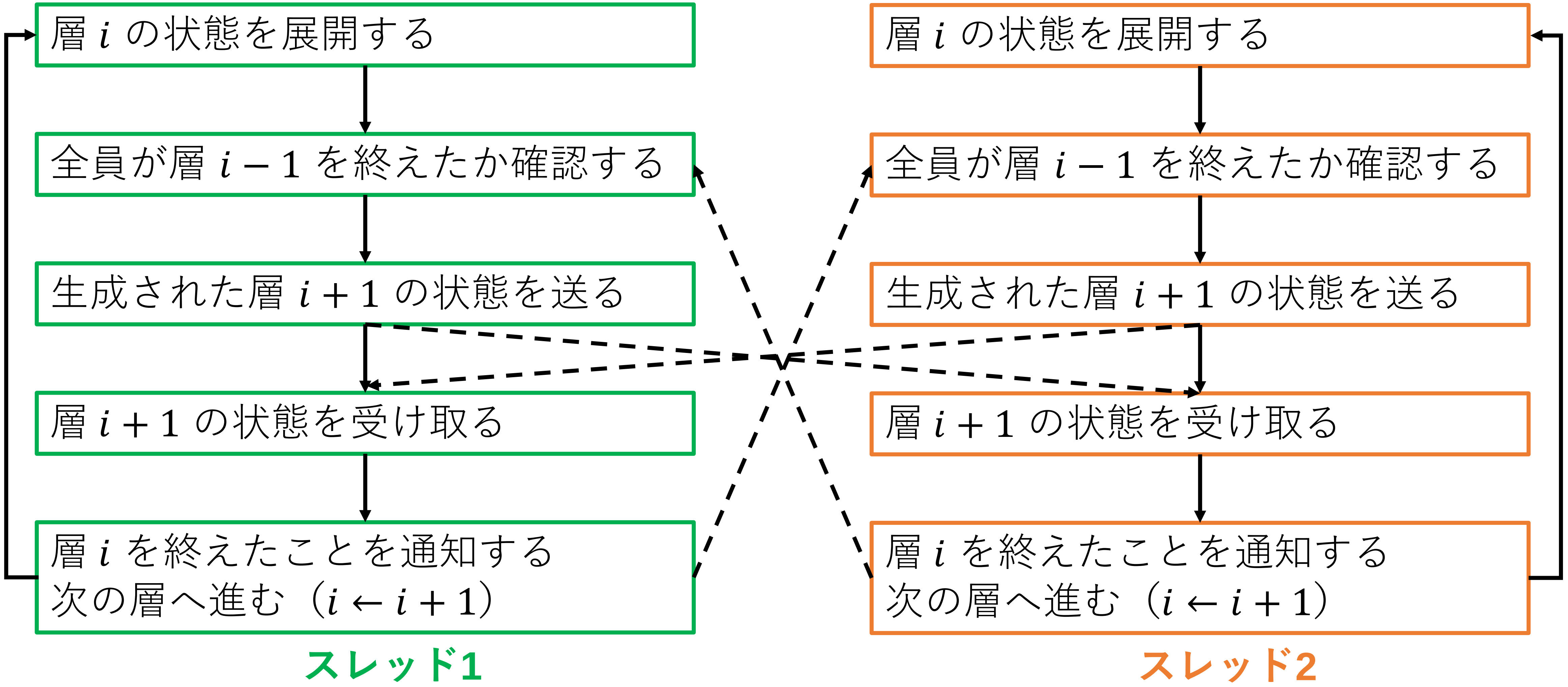


Hash Distributed A* [Kishimoto+ 2013] のビームサーチへの応用

HDBS1：すぐに層を同期する

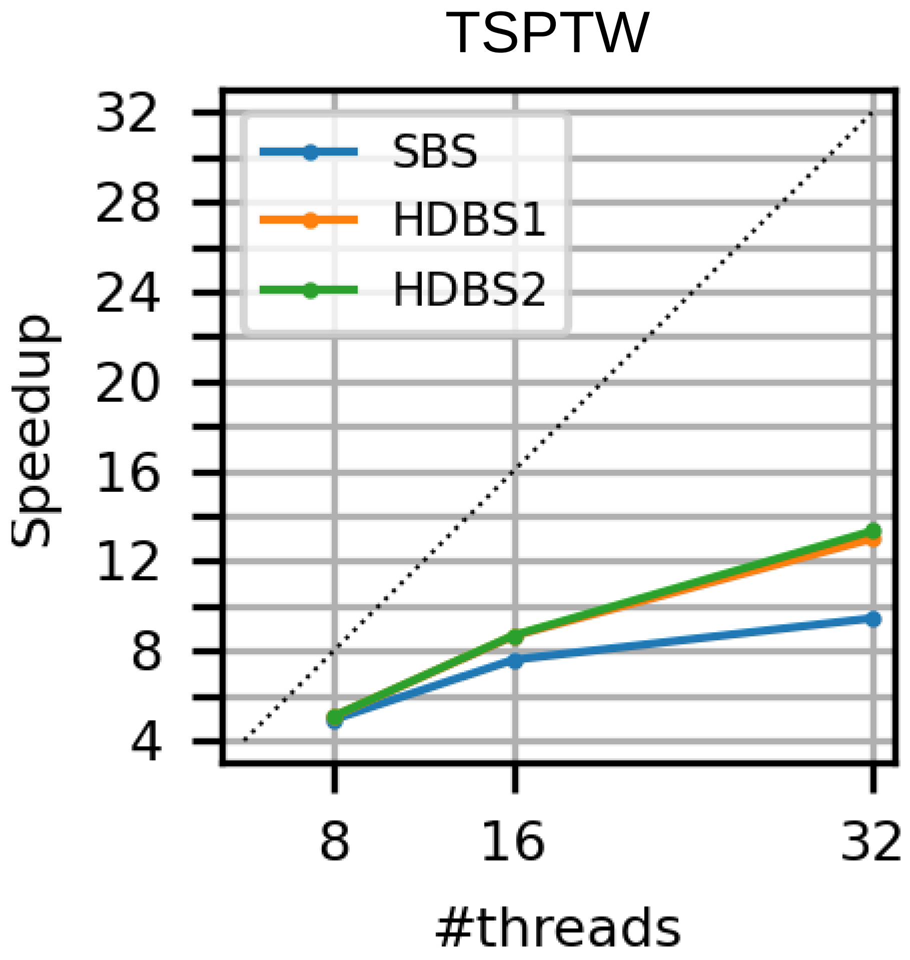


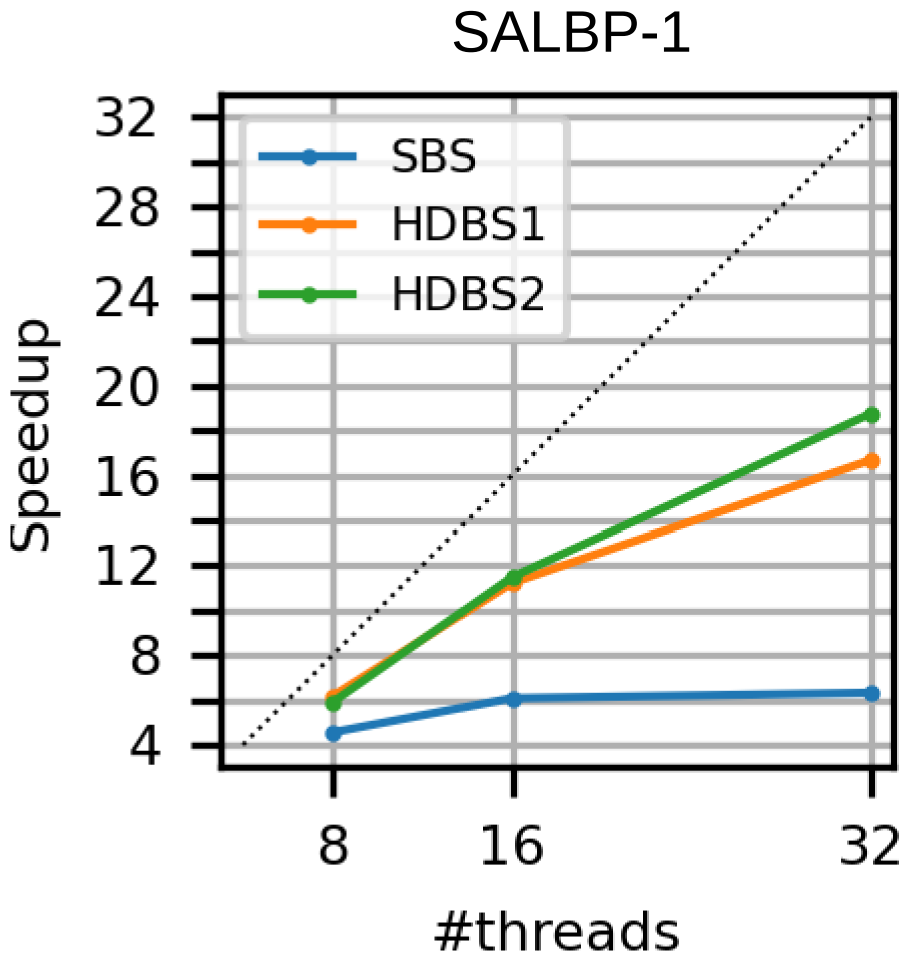
HDBS2：あとで層を同期する



実験結果

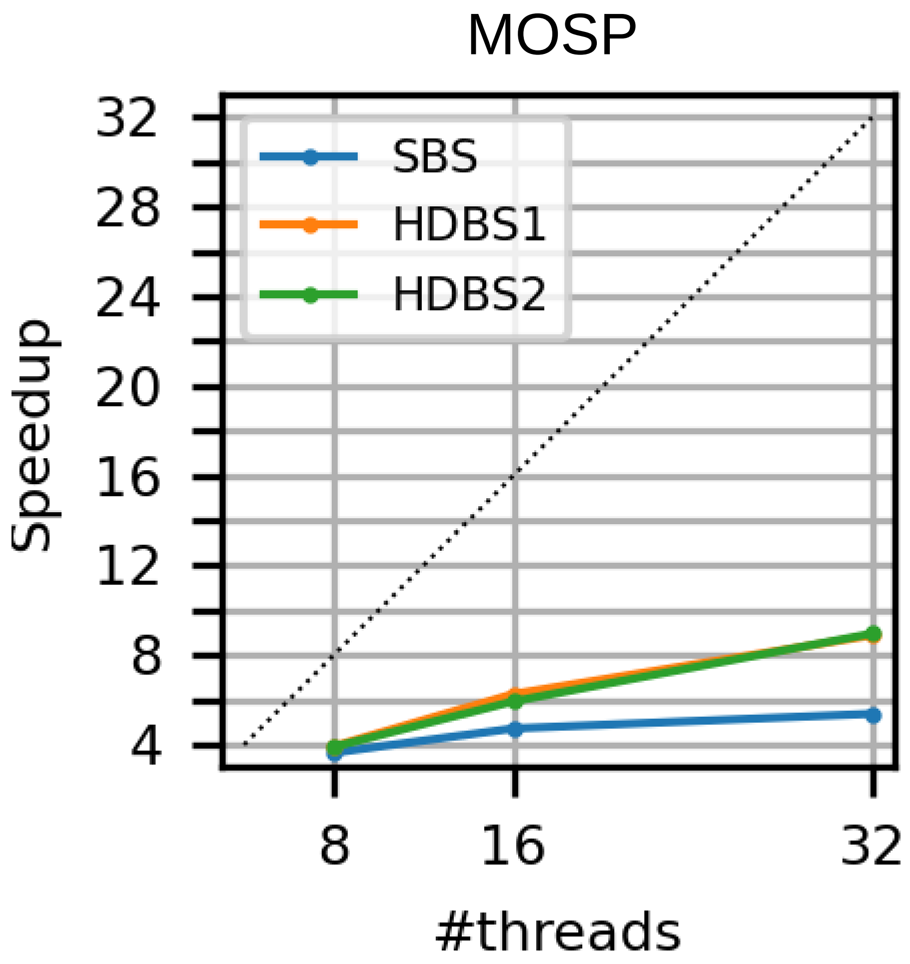
SBS vs. HDBS：1スレッドからの平均スピードアップ

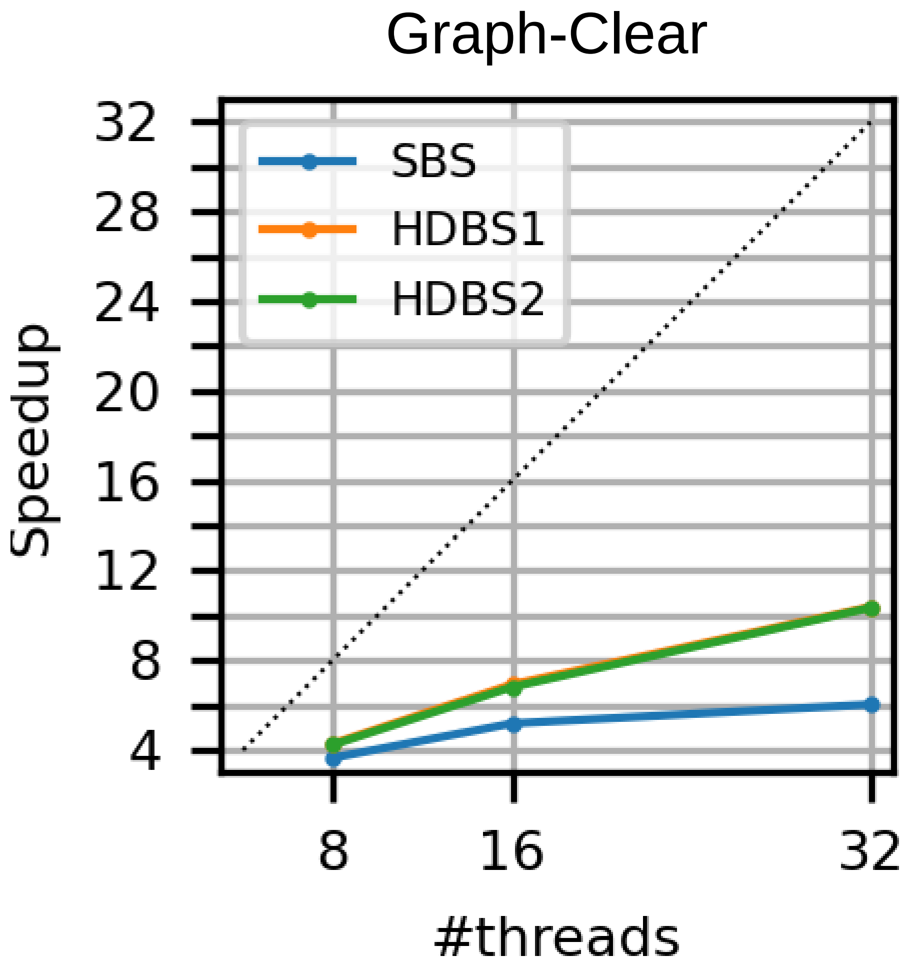




CABSでインスタンスを最適に解くまでの時間を計測（時間制限5分、メモリ188GB）

SBS vs. HDBS：1スレッドからの平均スピードアップ





CABSでインスタンスを最適に解くまでの時間を計測（時間制限5分、メモリ188GB）

マルチスレッドMIP・CPソルバとの比較

問題	説明	MIP	CP	CAHDBS
TSPTW (340)	時間枠付きTSP	239/4.2	27/0.1	**262**/13.3
CVRP (207)	配車計画問題	**29**/5.3	0/ -	8/ 9.3
Bin Packing (1615)	ビンパッキング	1192/6.4	**1251**/9.2	1239/39.6
SALBP-1 (2100)	組立ライン最適化	1351/1.3	1581/1.4	**1826**/18.8
MOSP (570)	生産順序最適化	238/3.1	397/0.3	**531**/ 9.0
Graph-Clear (135)	ロボット警備最適化	16/2.0	4/3.2	**113**/10.3

最適に解けた数 / 平均スピードアップ

32 スレッド、5分、188GB

MIP: Gurobi 10.0.1, CP: CP Optimizer 22.1.0

まとめ

- DIDPは複数の問題で整数計画法や制約プログラミングを上回る有望な汎用ソルバである
- 今後の展望：モデリング機能の拡張、ソルバの改善、他のソルバとの組合せ、などなど

チュートリアル・APIドキュメント

ホームページ

GitHubリポジトリ