RPID: Rust Programmable Interface for Domain-Independent Dynamic Programming

Ryo Kuroiwa, National Institute of Informatics

J. Christopher Beck, University of Toronto

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

UNIVERSITY OF TORONTO

Overview

1. Dynamic Programming (Background)
2. didp-rs: Domain-Independent Dynamic Programming Software (Background)
3. RPID
4. RPID vs. didp-rs
5. Comparison of Different RPID Models
6. RPID vs. Decision Diagram-Based Solvers
7. Summary

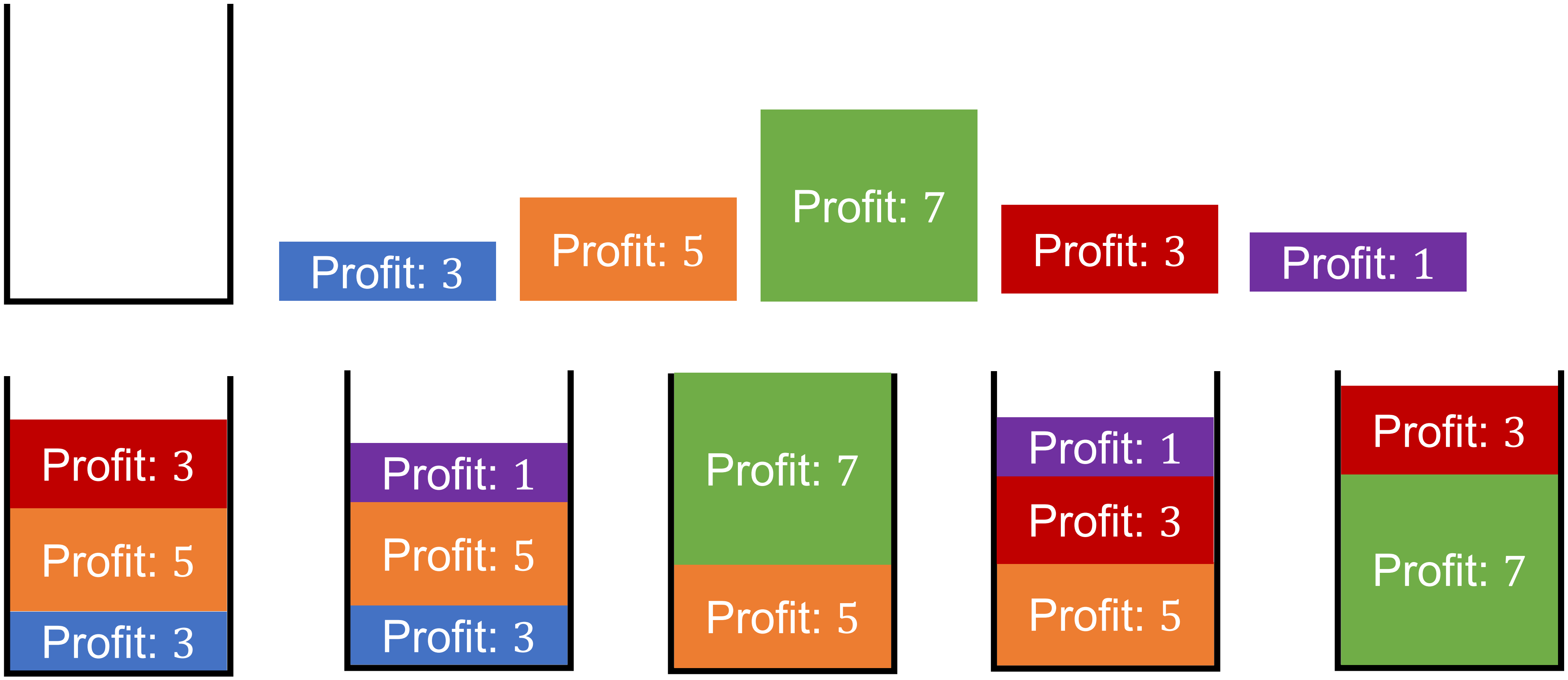
Dynamic Programming (Background)

Example: 0-1 Knapsack

Maximize the total profit of items packed in the knapsack

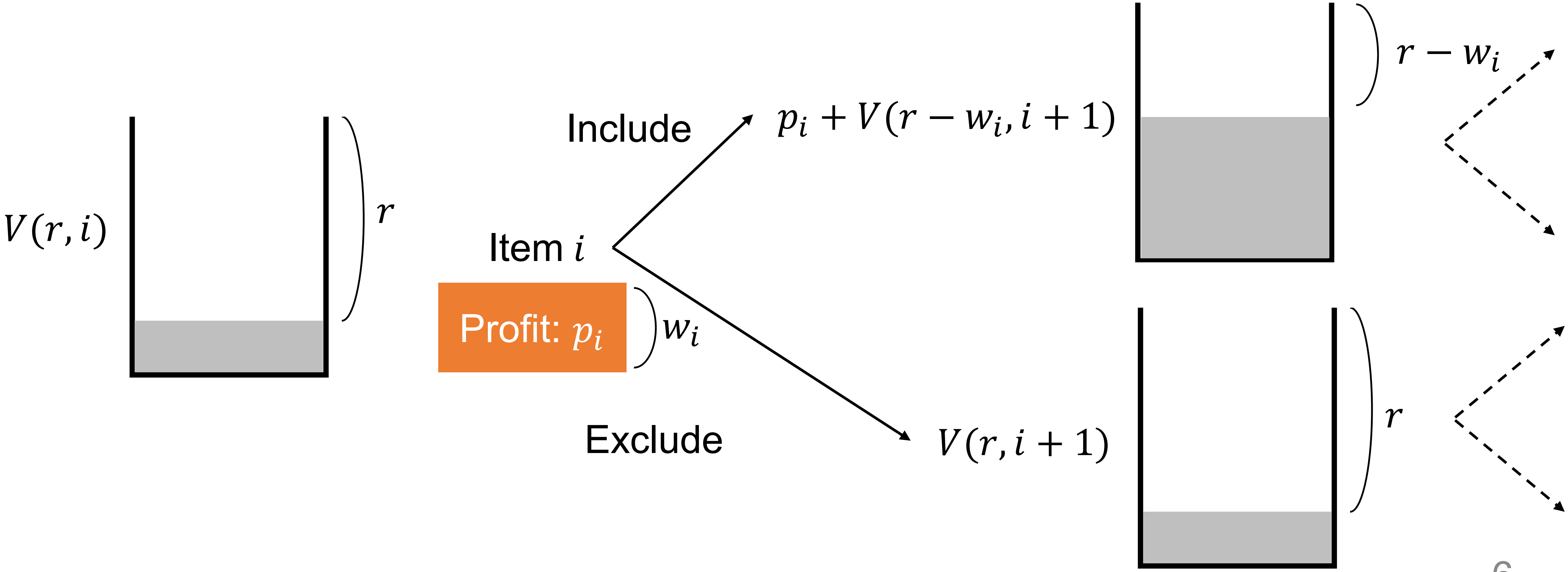
Example: 0-1 Knapsack

Maximize the total profit of items packed in the knapsack



Dynamic Programming (DP) for Knapsack

- State: the current capacity r and the current item index i
- $V(r, i)$: the maximum profit achieved from the state



Dynamic Programming (DP) for Knapsack

- State: the current capacity r and the current item index i
- $V(r, i)$: the maximum profit achieved from the state
- n: # of items
- Objective: compute $V(c, 0)$, where c is the knapsack capacity

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

Dynamic Programming (DP) for Knapsack

- State: the current capacity r and the current item index i
- $V(r,i)$: the maximum profit achieved from the state
- n: # of items
- Objective: compute $V(c,0)$, where c is the knapsack capacity

State transitions

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

Dynamic Programming (DP) for Knapsack

- State: the current capacity r and the current item index i
- $V(r,i)$: the maximum profit achieved from the state
- n: # of items
- Objective: compute $V(c,0)$, where c is the knapsack capacity

Successor states

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

Dynamic Programming (DP) for Knapsack

- State: the current capacity r and the current item index i
- $V(r, i)$: the maximum profit achieved from the state
- n: # of items
- Objective: compute $V(c, 0)$, where c is the knapsack capacity

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

Base case

Dynamic Programming (DP) for Knapsack

- State: the current capacity r and the current item index i
- $V(r, i)$: the maximum profit achieved from the state
- n: # of items
- Objective: compute $V(c, 0)$, where c is the knapsack capacity

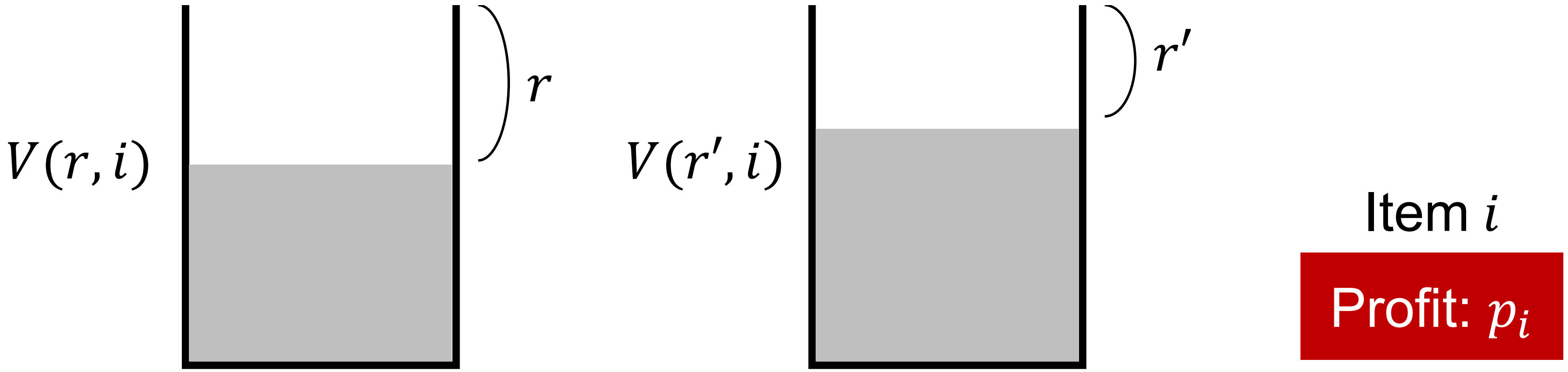
$$V(r,i) = \begin{cases} \max\{p_i + V(r - w_i, i+1), V(r, i+1)\} & \text{if } i < n \text{ and } r \geq w_i \\ V(r, i+1) & \text{if } i < n \text{ and } r < w_i \\ 0 & \text{if } i = n \end{cases}$$

We further incorporate redundant information implied by the recursive equation into a model for efficiency

State Dominance in DP

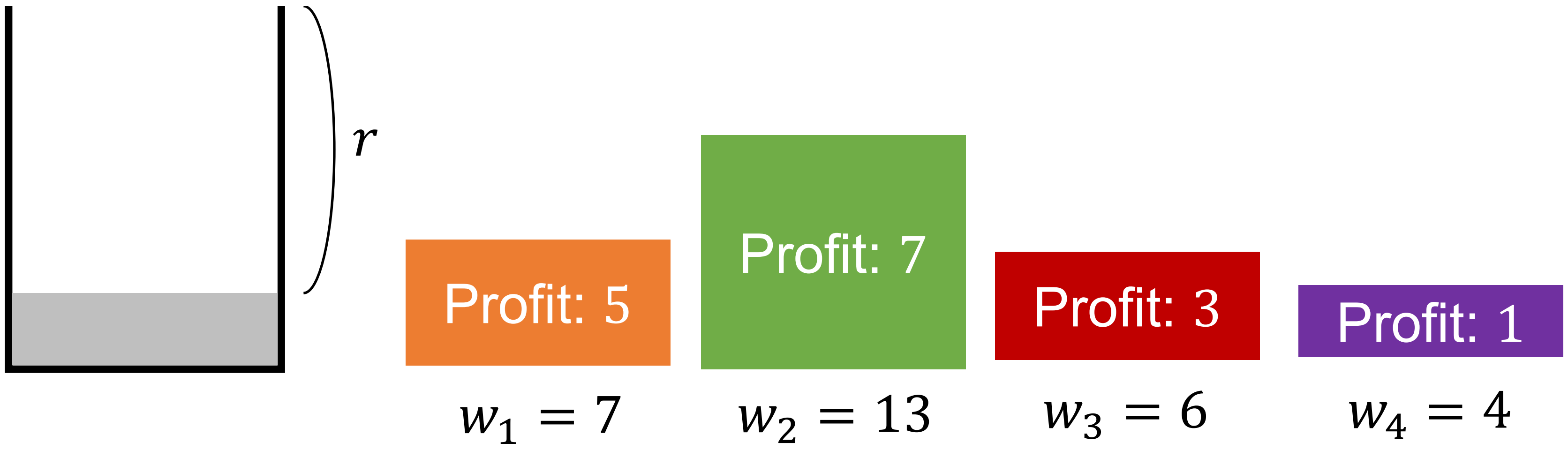
- One state may be known to be better than another state
- If the item index i is the same, having more capacity is better

$$V(r, i) \geq V(r', i) \text{ if } r \geq r'$$



Dual Bound in DP

Upper bound on V in maximization (lower bound in minimization)

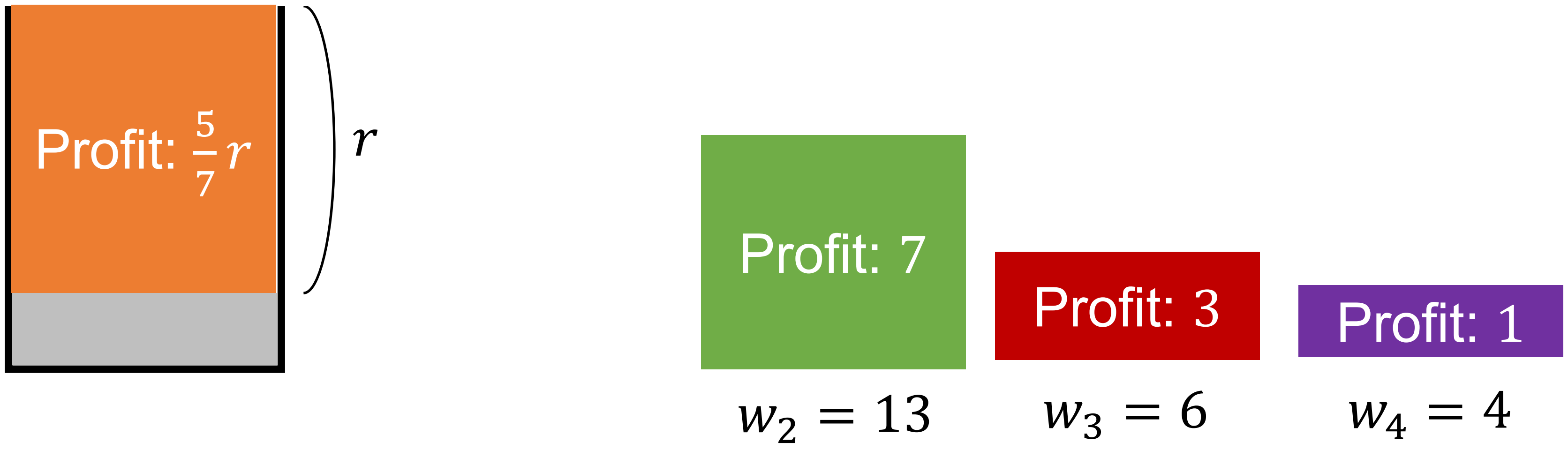


Dual Bound in DP

Upper bound on V in maximization (lower bound in minimization)

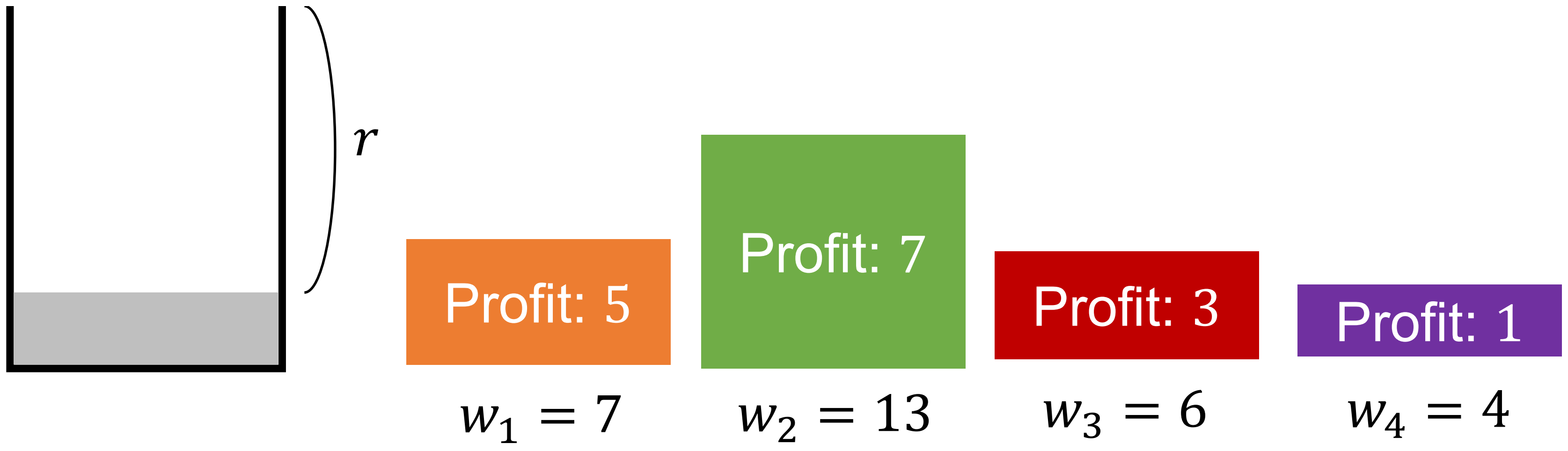
Take the most efficient item and fill the knapsack with the item

$$V(r, i) \leq \left\lfloor r \cdot \max_{j \geq i} \frac{p_j}{w_j} \right\rfloor$$



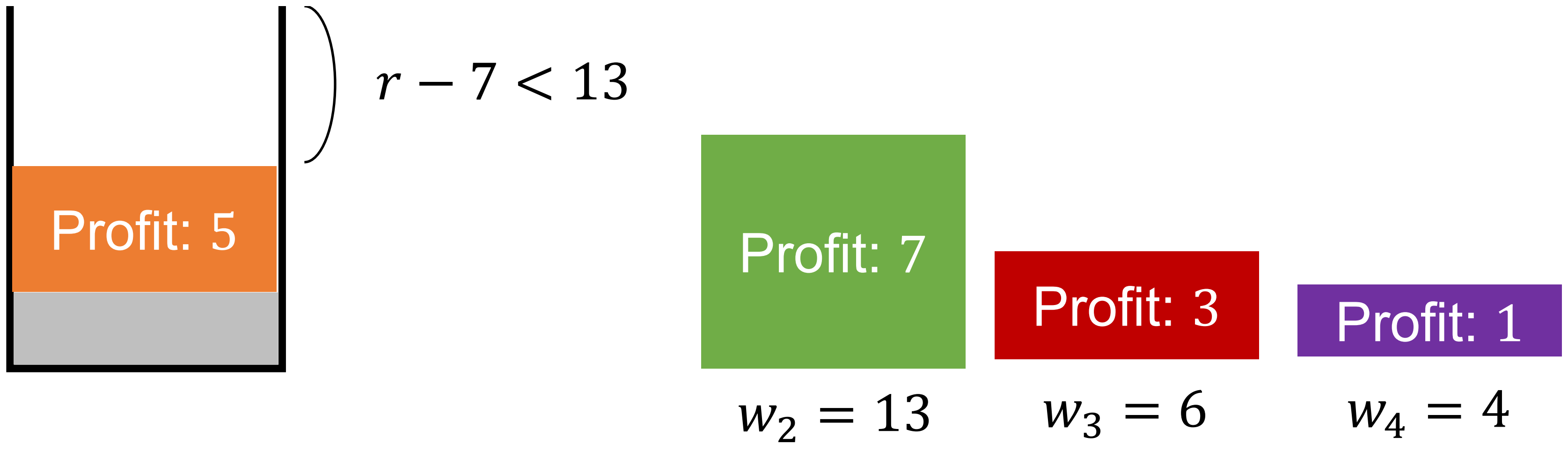
Dantzig Bound for 0-1 Knapsack [Dantzig 1957]

1. Sort items by efficiency $\frac{p_j}{w_j}$



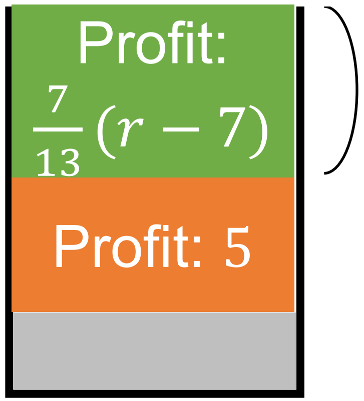
Dantzig Bound for 0-1 Knapsack [Dantzig 1957]

1. Sort items by efficiency $\frac{p_j}{w_j}$
2. Pack items in order until reaching the capacity limit

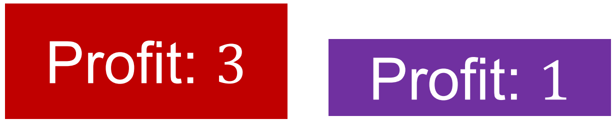


Dantzig Bound for 0-1 Knapsack [Dantzig 1957]

1. Sort items by efficiency $\frac{p_j}{w_j}$
2. Pack items in order until reaching the capacity limit
3. Fractionally include the last item



$r - 7 < 13$



$w_3 = 6$ $w_4 = 4$

didp-rs: Domain-Independent Dynamic Programming Software (Background)

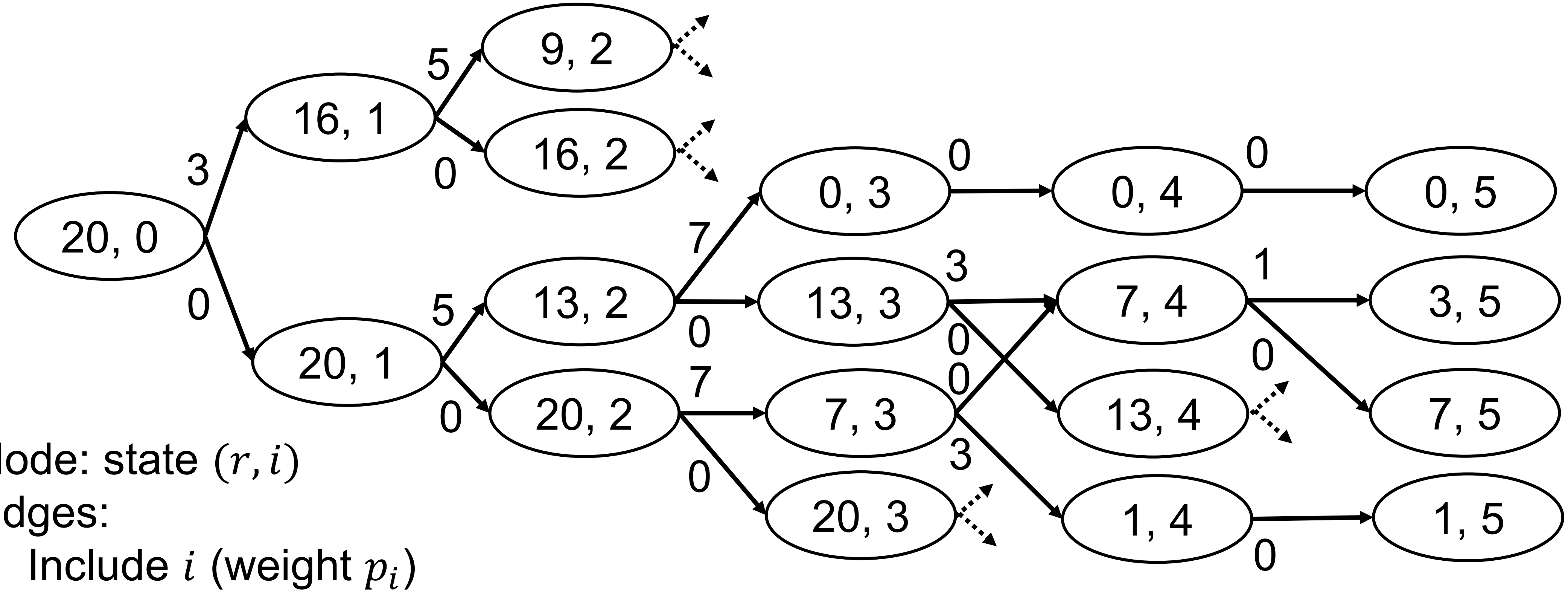
General-Purpose DP Solvers

Similarly to CP, a user formulates a declarative DP model and then solves it with a general-purpose solver

	Modeling interfaces	Solving algorithm
ddo [Gillard, Schaus, and Coppe 2020]	Rust trait Python class	decision diagram-based branch-and-bound
CODD [Michel and van Hoeve 2024]	C++ lambda	decision diagram-based branch-and-bound
didp-rs [Kuroiwa and Beck 2023]	**Rust expressions** **Python expressions** **YAML expressions**	**heuristic state space search**

Heuristic State Space Search in didp-rs

- Find a longest path in a state space graph (shortest for minimization)
- Use state dominance and dual bounds for pruning and guidance



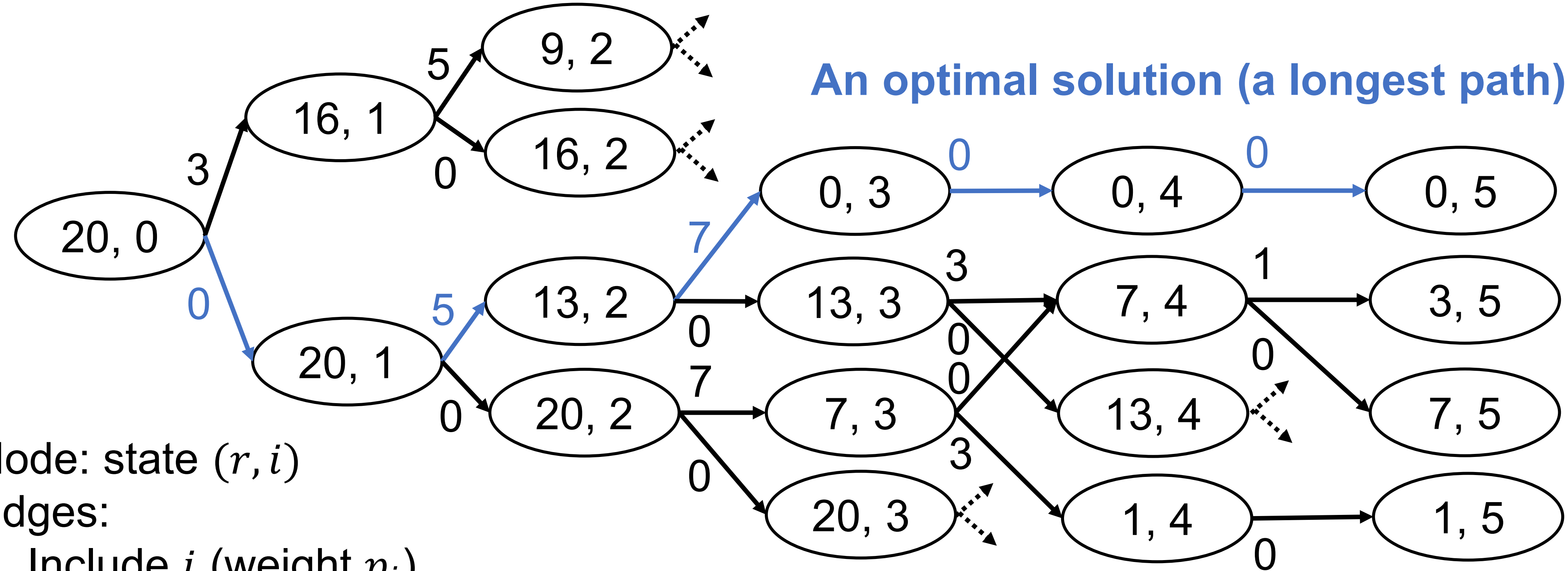
Node: state (r, i)

Edges:

- Include i (weight p_i)
- Exclude i (weight 0)

Heuristic State Space Search in didp-rs

- Find a longest path in a state space graph (shortest for minimization)
- Use state dominance and dual bounds for pruning and guidance



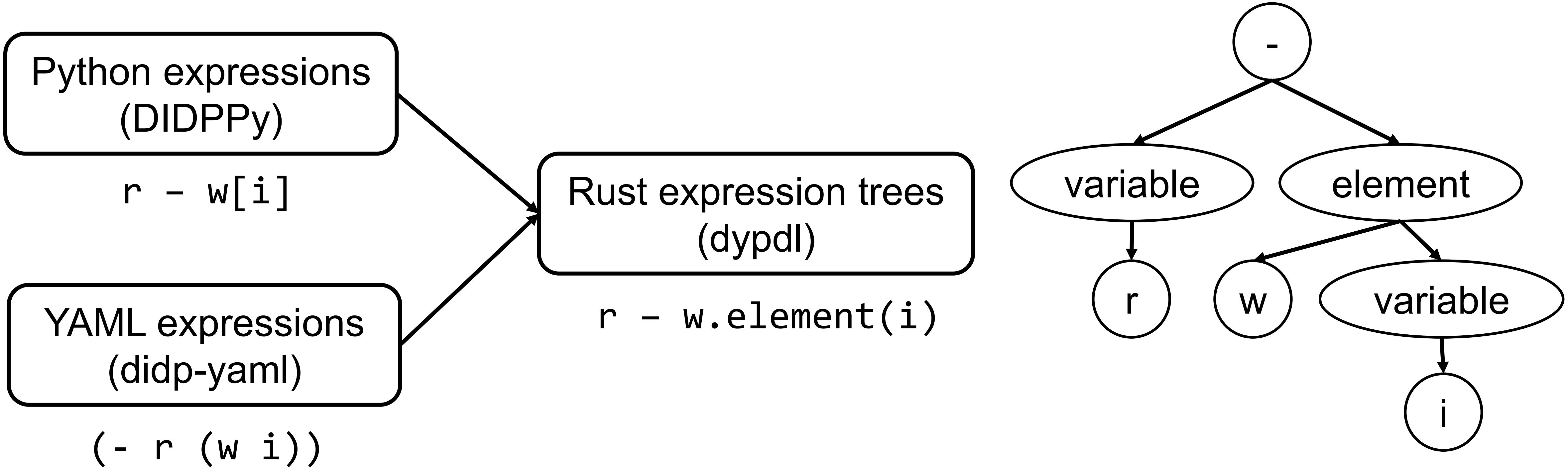
Node: state (r, i)

Edges:

- Include i (weight p_i)
- Exclude i (weight 0)

Modeling in didp-rs

- Define a DP model by writing expressions
- Expression tree data structures are implemented in Rust



DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

Constants
(profits, weights, items)

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

State variables
(capacity, item index)

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

Values in the original problem (called the target state)

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

State dominance specified by a resource variable (larger r is better)

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

State transition
to include the current item

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

Expressions in Python syntax

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

$p_i + V(r - w_i, i + 1)$

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

$p_i + V(r - w_i, i + 1)$

if $r \geq w_i$

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

State transition to exclude the current item: $V(r, i + 1)$

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

Base case: $V(r, i) = 0$ if $i = n$

DIDPPy: Python Interface in didp-rs

```python
model = dp.Model(maximize=True)
p = model.add_int_table([3, 5, 7, 3, 1])
w = model.add_int_table([4, 7, 13, 6, 4])
item = model.add_object_type(number=n)
r = model.add_int_resource_var(target=c, less_is_better=False)
i = model.add_element_var(object_type=item, target=0)
include = dp.Transition(
    name="include",
    cost=p[i] + dp.IntExpr.state_cost(),
    effects=[(r, r - w[i]), (i, i + 1)],
    preconditions=[r >= w[i]],
)
model.add_transition(include)
exclude = dp.Transition(
    name="exclude",
    cost=dp.IntExpr.state_cost(),
    effects=[(i, i + 1)],
)
model.add_transition(exclude)
model.add_base_case([i == n])
max_efficiency = model.add_float_table([3 / 4, 5 / 7, 7 / 13, 3 / 6, 1 / 4, 0])
model.add_dual_bound(math.floor(r * max_efficiency[i]))
```

A dual bound function $\left\lfloor r \cdot \max_{j \geq i} \frac{p_j}{w_j} \right\rfloor$

The Dantzig bound is hard to model with the current expressions

dypdl: Rust Modeling Library and Interface in didp-rs

```rust
    let mut model: Model = Model::default();
    model.set_maximize();
    let p: Table1DHandle<Integer> = model.add_table_1d(name: "p", v: vec![3, 5, 7, 3, 1]).unwrap();
    let w: Table1DHandle<Integer> = model.add_table_1d(name: "w", v: vec![4, 7, 13, 6, 4]).unwrap();
    let item: ObjectType = model.add_object_type(name: "item", number: n).unwrap();
    let r: IntegerResourceVariable = model.add_integer_resource_variable(name: "r", less_is_bette… false, c).unwrap();
    let i: ElementVariable = model.add_element_variable(name: "i", ob: item, target: 0).unwrap();
    let mut include: Transition = Transition::new(name: "include");
    include.set_cost(p.element(i) + IntegerExpression::Cost);
    include.add_effect(v: r, expression: r - w.element(i)).unwrap();
    include.add_effect(v: i, expression: i + 1).unwrap();
    include.add_precondition(Condition::comparison_i(op: ComparisonOperator::Ge, lhs: r, rhs: w.element(i)));
    model.add_forward_transition(include).unwrap();
    let mut exclude: Transition = Transition::new(name: "exclude");
    exclude.add_effect(v: i, expression: i + 1).unwrap();
    model.add_forward_transition(exclude).unwrap();
    model.add_base_case(conditions: vec![Condition::comparison_e(ComparisonOperator::Eq, i, n)]).unwrap();
    let max_efficiency: Table1DHandle<f64> = model.add_table_1d(
            name: "max_efficiency",
            v: vec![3.0 / 4.0, 5.0 / 7.0, 7.0 / 13.0, 3.0 / 6.0, 1.0 / 4.0],
        ).unwrap();
    model.add_dual_bound(IntegerExpression::floor(r * max_efficiency.element(i))).unwrap();
```

dypdl: Rust Modeling Library and Interface in didp-rs

```rust
    let mut model: Model = Model::default();
    model.set_maximize();
    let p: Table1DHandle<Integer> = model.add_table_1d(name: "p", v: vec![3, 5, 7, 3, 1]).unwrap();
    let w: Table1DHandle<Integer> = model.add_table_1d(name: "w", v: vec![4, 7, 13, 6, 4]).unwrap();
    let item: ObjectType = model.add_object_type(name: "item", number: n).unwrap();
    let r: IntegerResourceVariable = model.add_integer_resource_variable(name: "r", less_is_bette… false, c).unwrap();
    let i: ElementVariable = model.add_element_variable(name: "i", ob: item, target: 0).unwrap();
    let mut include: Transition = Transition::new(name: "include");
    include.set_cost(p.element(i) + IntegerExpression::Cost);
    include.add_effect(v: r, expression: r - w.element(i)).unwrap();
    include.add_effect(v: i, expression: i + 1).unwrap();
    include.add_precondition(Condition::comparison_i(op: ComparisonOperator::Ge, lhs: r, rhs: w.element(i)));
    model.add_forward_transition(include).unwrap();
    let mut exclude: Transition = Transition::new(name: "exclude");
    exclude.add_effect(v: i, expression: i + 1).unwrap();
    model.add_forward_transition(exclude).unwrap();
    model.add_base_case(conditions: vec![Condition::comparison_e(ComparisonOperator::Eq, i, n)]).unwrap();
    let max_efficiency: Table1DHandle<f64> = model.add_table_1d(
        name: "max_efficiency",
        v: vec![3.0 / 4.0, 5.0 / 7.0, 7.0 / 13.0, 3.0 / 6.0, 1.0 / 4.0],
    ).unwrap();
    model.add_dual_bound(IntegerExpression::floor(r * max_efficiency.element(i))).unwrap();
```

Expressions in Rust syntax

didp-yaml: YAML Interface in didp-rs

```yaml
objects:
  - item
state_variables:
  - name: r
    type: integer
    preference: greater
  - name: i
    type: element
    object: item
tables:
  - name: "n"
    type: element
  - name: p
    type: integer
    args:
      - item
  - name: w
    type: integer
    args:
      - item
  - name: max_efficiency
    type: continuous
    args:
      - item
```

```yaml
transitions:
  - name: include
    cost: (+ (p i) cost)
    effect:
      r: (- r (w i))
      i: (+ i 1)
    preconditions:
      - (>= r (w i))
  - name: exclude
    effect:
      i: (+ i 1)
base_cases:
  - - (= i n)
dual_bounds:
  - (floor (* r (max_efficiency i)))
reduce: max
```

didp-yaml: YAML Interface in didp-rs

```yaml
objects:
  - item
state_variables:
  - name: r
    type: integer
    preference: greater
  - name: i
    type: element
    object: item
tables:
  - name: "n"
    type: element
  - name: p
    type: integer
    args:
      - item
  - name: w
    type: integer
    args:
      - item
  - name: max_efficiency
    type: continuous
    args:
      - item
```

```yaml
transitions:
  - name: include
    cost: (+ (p i) cost)
    effect:
      r: (- r (w i))
      i: (+ i 1)
    preconditions:
      - (>= r (w i))
  - name: exclude
    effect:
      i: (+ i 1)
base_cases:
  - - (= i n)
dual_bounds:
  - (floor (* r (max_efficiency i)))
reduce: max
```

Expressions in a LISP-like syntax

Pros and Cons of Expressions in didp-rs

Pros

- Declarative representation
- Different modeling interfaces with the same solving performance
- Python and Rust models can be exported to YAML files
- Algorithms may exploit structures (e.g., Kuroiwa and Beck CP2023)

Pros and Cons of Expressions in didp-rs

Pros

- Declarative representation
- Different modeling interfaces with the same solving performance
- Python and Rust models can be exported to YAML files
- Algorithms may exploit structures (e.g., Kuroiwa and Beck CP2023)

Cons

- Hard to express algorithmic procedures (e.g., the Dantzig bound)
- Evaluating expression trees is slower than calling native Rust code

RPID

General-Purpose DP Solvers

Similarly to CP, a user formulates a declarative DP model and then solves it with a general-purpose solver

	Modeling interfaces	Solving algorithm
ddo [Gillard, Schaus, and Coppe 2020]	**Rust trait** Python class	decision diagram-based branch-and-bound
CODD [Michel and van Hoeve 2024]	**C++ lambda**	decision diagram-based branch-and-bound
didp-rs [Kuroiwa and Beck 2023]	Rust expressions Python expressions YAML expressions	heuristic state space search

General-Purpose DP Solvers

Similarly to CP, a user formulates a declarative DP model and then solves it with a general-purpose solver

	Modeling interfaces	Solving algorithm
ddo [Gillard, Schaus, and Coppe 2020]	Rust trait Python class	decision diagram-based branch-and-bound
CODD [Michel and van Hoeve 2024]	C++ lambda	decision diagram-based branch-and-bound
didp-rs [Kuroiwa and Beck 2023]	Rust expressions Python expressions YAML expressions	heuristic state space search
RPID (this work)	**Rust trait**	heuristic state space search

Rust Trait

A set of methods defining behavior of a type (similar to an interface or an abstract class, but without data members and inheritance)

```rust
trait Animal { fn sound(&self) -> String; }
1 implementation
struct Dog;
impl Animal for Dog { fn sound(&self) -> String { "Bow".into() } }
1 implementation
struct Cat;
impl Animal for Cat { fn sound(&self) -> String { "Meow".into() } }

fn print_sound<T: Animal>(animal: &T) { println!("{}", animal.sound()); }

▶ Run | ⚙ Debug
fn main() {
    let dog: Dog = Dog;
    print_sound(animal: &dog);
    let cat: Cat = Cat;
    print_sound(animal: &cat)
}
```

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

Struct defining an instance

RPID Example for 0-1 Knapsack

```
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

State represented by 2 variables
(the capacity and the current item index)

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

Maximization of an integral objective

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

State representing the original problem
(called the target state)

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

The successor states, the transition weights, and the transition labels given a state

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

The successor states, the transition weights, and the transition labels given a state

Successor states

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

The successor states, the transition weights, and the transition labels given a state

Transition weights

Objective: the sum of the transition weights (sum by default, can be overridden)

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

The successor states, the transition weights, and the transition labels given a state

Transition labels (include: 1, exclude: 0) to reconstruct a solution

RPID Example for 0-1 Knapsack

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

If a state S is a base case, return $V(S)$
Otherwise, return None

State Dominance in RPID

Optional trait

Compare two states having the same key (similarly to ddo)

```
impl Dominance for Knapsack {
    type State = (i32, usize);
    type Key = usize;

    fn get_key(&self, &(_, i: usize): &Self::State) -> Self::Key {
        i
    }
    fn compare(&self, (r: &i32, _): &Self::State, (q: &i32, _): &Self::State) -> Option<Ordering> {
        Some(r.cmp(q))
    }
}
```

Dual Bound in RPID

Optional trait

```
impl Bound for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_dual_bound(&self, &(r: i32, i: usize): &Self::State) -> Option<Self::CostType> {
        let mut bound: i32 = 0;
        let mut r: i32 = r;
        for j: usize in i..self.n {
            if r >= self.w[j] {
                bound += self.p[j];
                r -= self.w[j];
            } else {
                bound += (((r * self.p[j]) as f64) / (self.w[j] as f64)).floor() as i32;
            }
        }
        Some(bound)
    }
}
```

Pros and Cons of RPID

Pros

- Flexible in modeling (e.g., the Dantzig bound)
- Calling native Rust code is fast

Pros and Cons of RPID

Pros

- Flexible in modeling (e.g., the Dantzig bound)
- Calling native Rust code is fast

Cons

- Less declarative
- A model is a black-box to a solver
- Hard to implement a modeling interface in a different language while maintaining the solving performance

Q. How fast is native Rust code compared with expressions?

Experimental Settings

DP models for 14 combinatorial problem classes including

- 0-1 knapsack
- Single machine total weighted tardiness ($1||\sum w_i T_i$)
- Traveling salesperson problem with time windows (TSPTW)

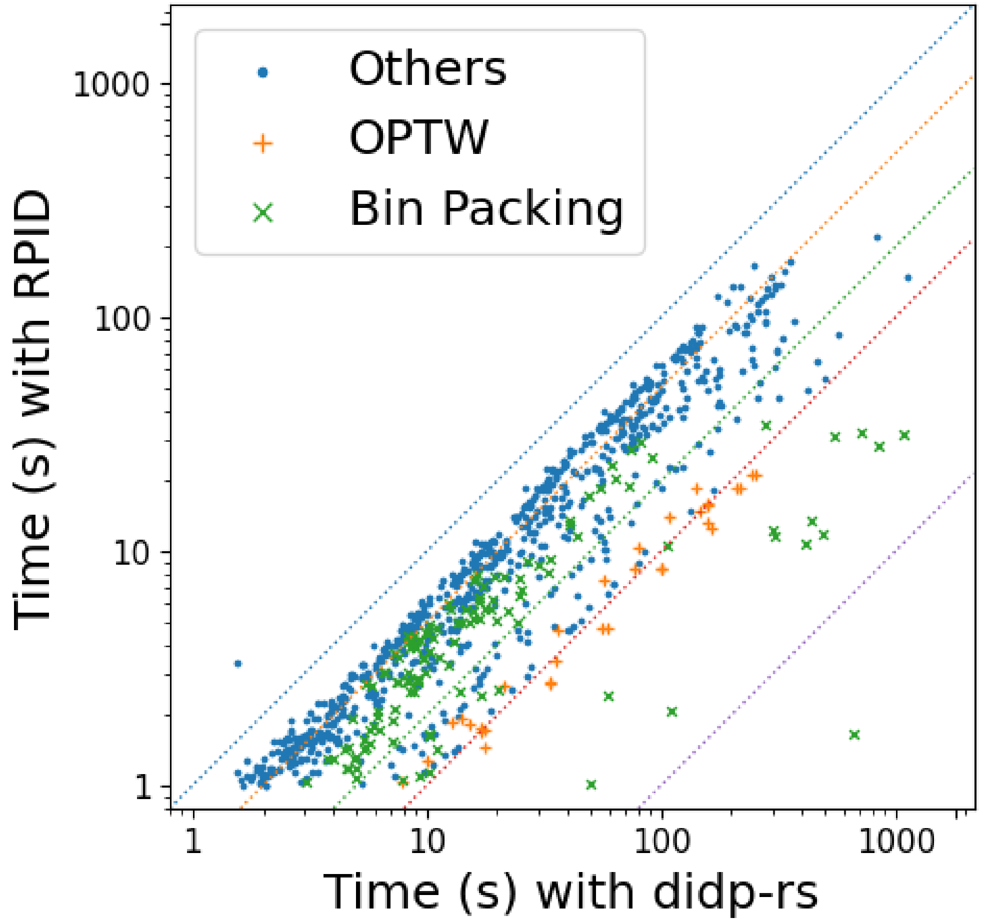
Two solving algorithms with 30-min time and 8 GB memory limit

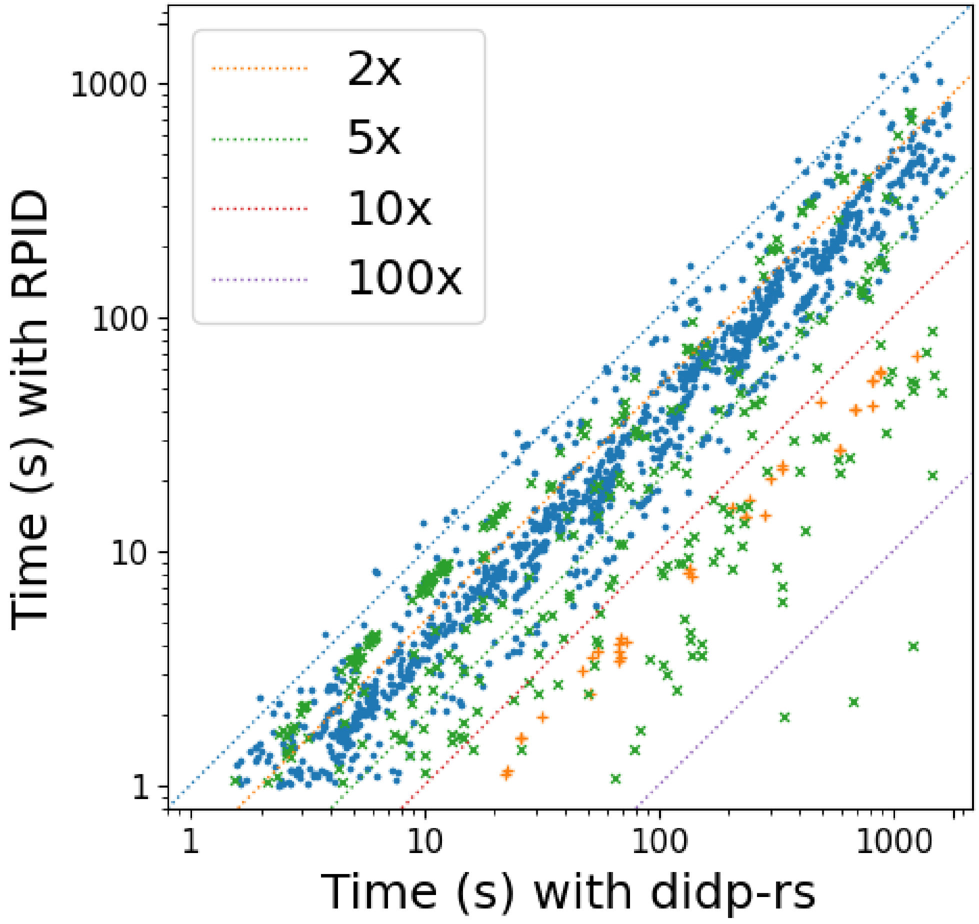
- A*: faster to prove optimality, less memory efficient
  [Hart and Nillson 1968; Kuroiwa and Beck ICAPS2023]
- Complete anytime beam search (CABS): more memory efficient
  [Zhang et al. 1998; Kuroiwa and Beck ICAPS2023]

RPID vs. didp-rs with the Same DP Models

Time (s) to optimally solve each instance

A*

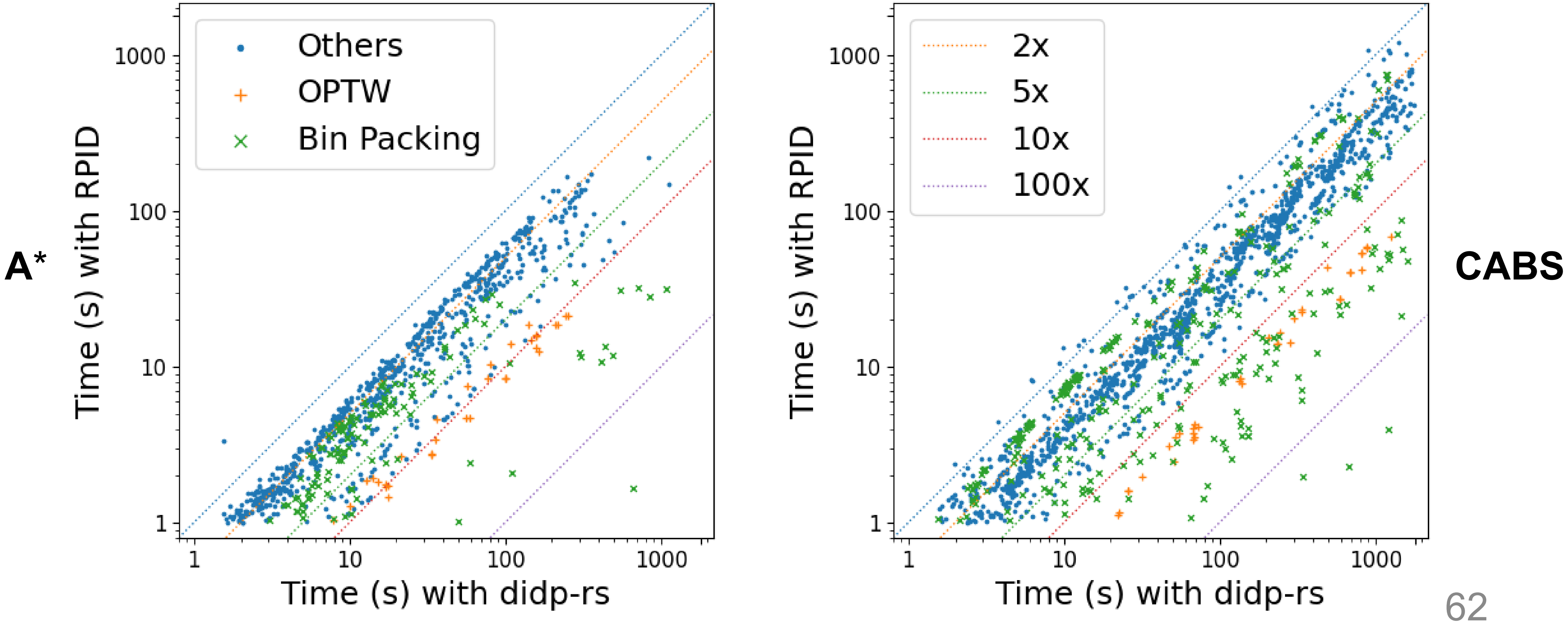




CABS

RPID vs. didp-rs with the Same DP Models

Large speedup in the orienteering problem with time windows (OPTW) and bin packing, where didp-rs uses large expression trees

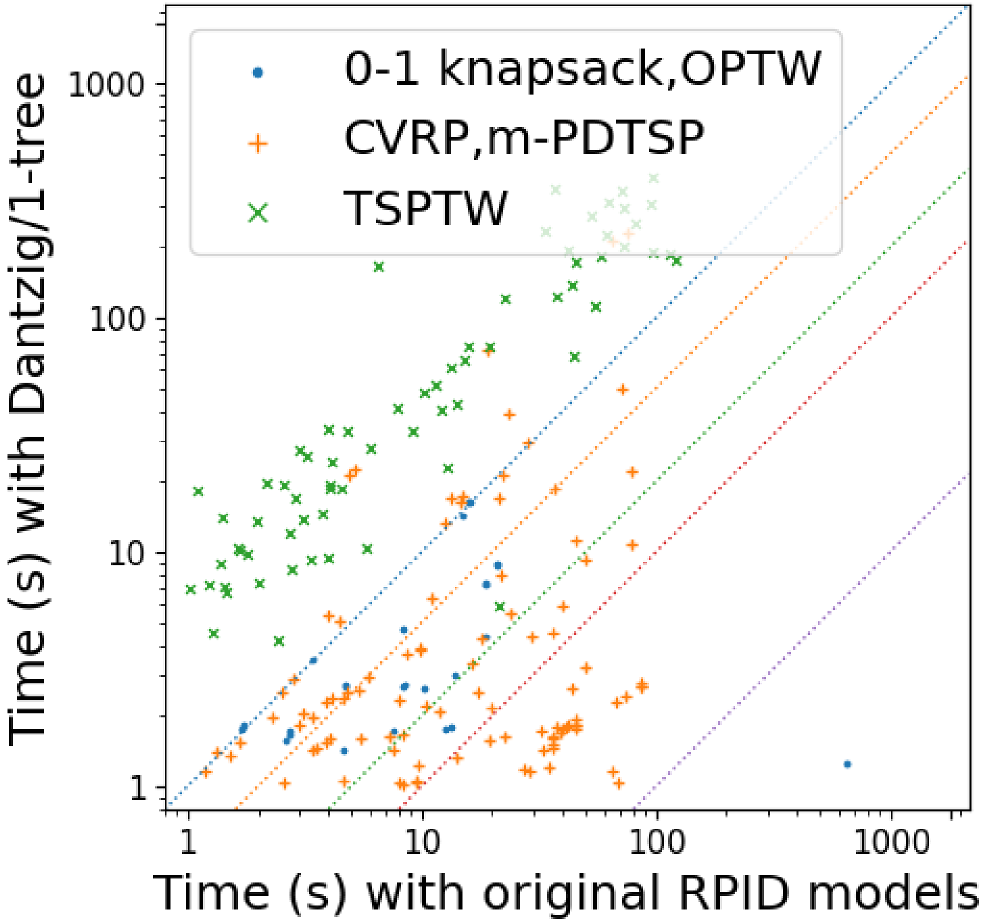


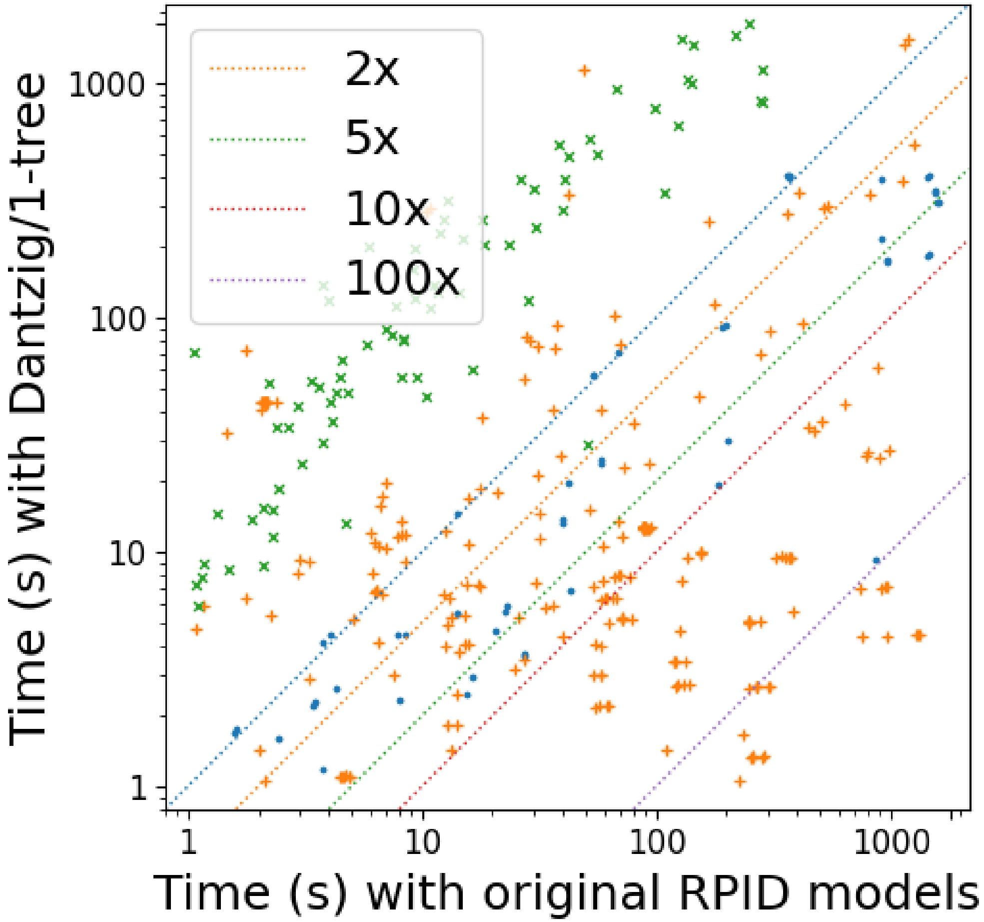
Q. How much gain do we get from algorithmic dual bound functions facilitated by RPID?

RPID with Algorithmic Dual Bound Functions

- The Dantzig bound for 0-1 knapsack and OPTW
- Bound using the minimum spanning tree (1-tree) for 3 TSP variants

A*



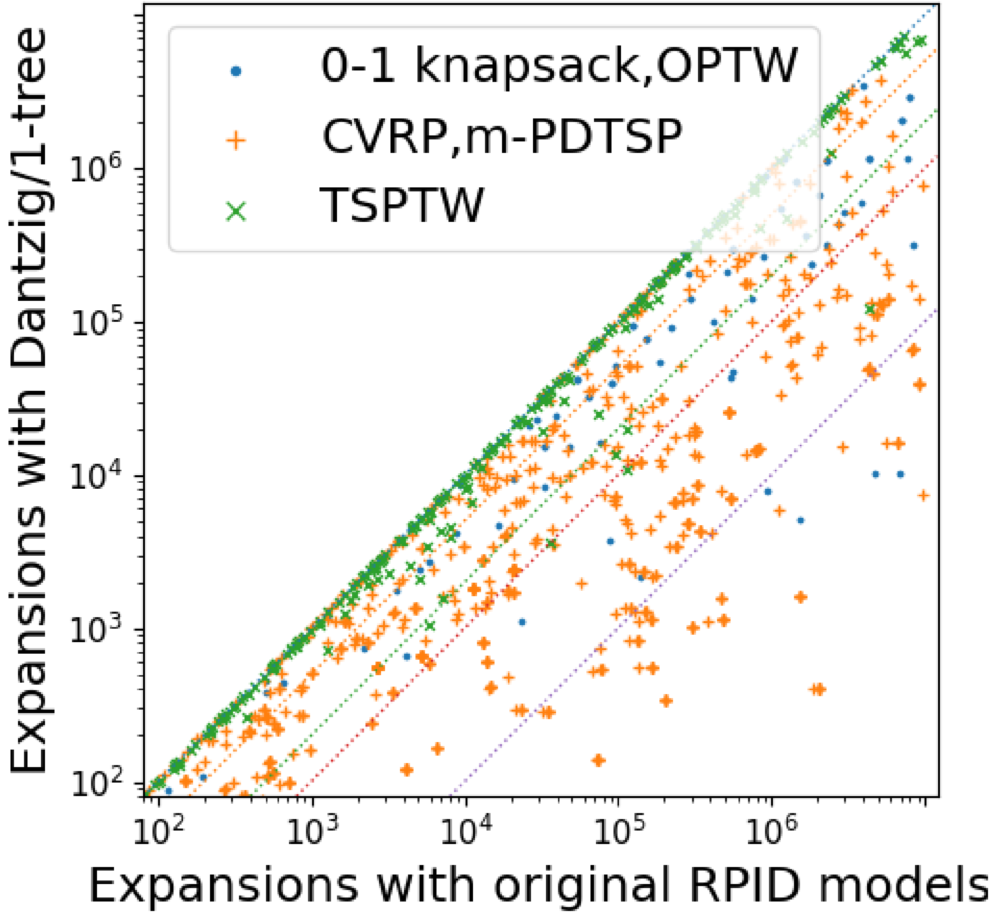


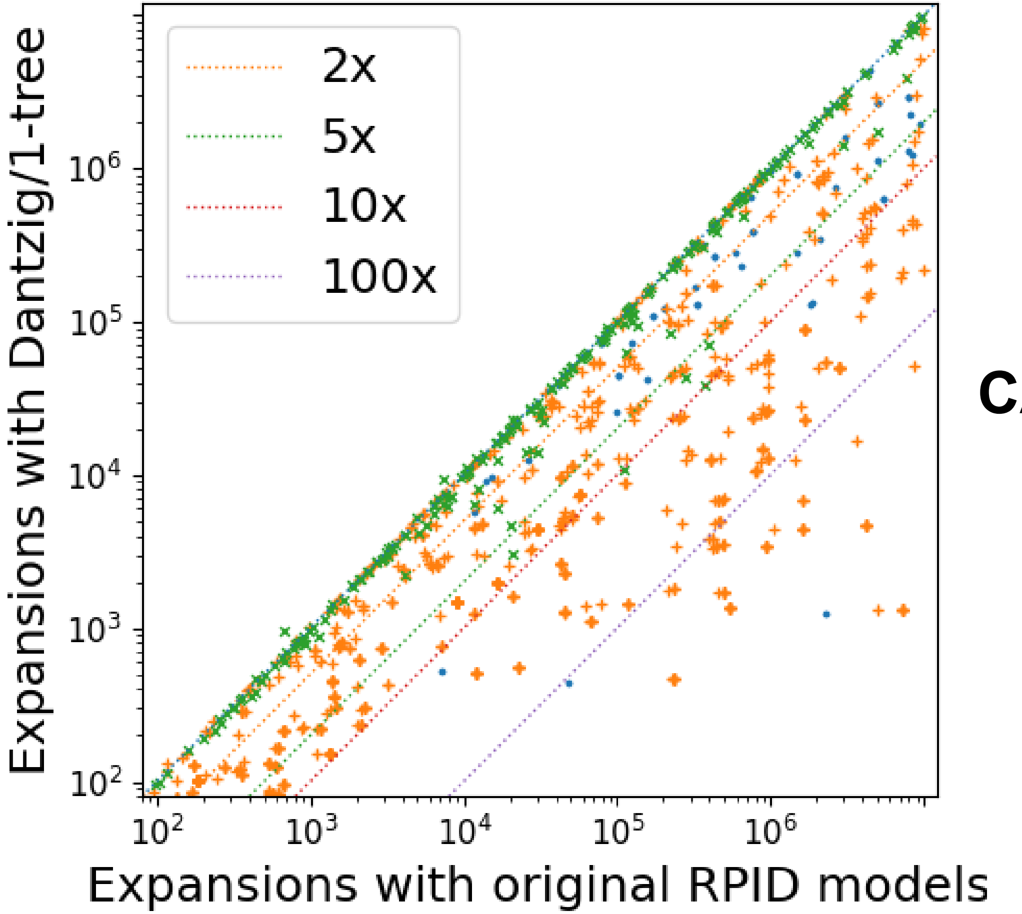
CABS

RPID with Algorithmic Dual Bound Functions

No reduction in search effort in many TSPTW instances possibly because time window constraints already prune many states

A*





CABS

Q. Is RPID competitive with DD-based solvers in the same problem classes?

Experimental Settings

3 problems used in the CODD paper [Michel and van Hoeve 2024], with which they compared ddo, didp-rs, and CODD:

- 0-1 Knapsack
- Golomb ruler
- Maximum independent set problem (MISP)

Model code in the original authors' repositories

The best parameters reported in previous work or the default ones

Time in seconds to solve each instance optimally, omitting instances solved within 1 second by all methods

Time (s) to Optimally Solve 0-1 Knapsack Instances

All solvers use the Dantzig bound and state dominance

	Ddo (width: 4)	CODD		RPID (A*)	RPID (CABS)
PI:1 5000	0.47	width: 64	2.73	**0.13**	0.28
PI:1 10000	0.71	width: 64	memory out	**0.16**	0.28
PI:2 2000	0.16	width: 64	3.70	**0.02**	0.29
PI:2 5000	0.44	width: 64	memory out	**0.15**	0.27
PI:2 10000	0.76	width: 64	memory out	**0.16**	0.27
PI:3 2000	3.23	width: 2048	3.47	**0.34**	1.38
PI:3 5000	4.87	width: 4096	6.29	**0.74**	4.83
PI:3 10000	4.32	width: 8192	memory out	**1.26**	8.73

Time (s) to Optimally Solve Golomb Ruler Instances

	Ddo (width: 10)	CODD (width: 128)	RPID (A*)	RPID (CABS)
n=8	0.71	**0.04**	1.88	1.19
n=9	6.89	**0.20**	14.02	13.85
n=10	50.55	**0.68**	memory out	143.84
n=11	memory out	**10.51**	memory out	memory out
n=12	memory out	**55.56**	memory out	time out
n=13	memory out	**1318.98**	memory out	memory out
n=14	memory out	time out	memory out	time out

Time (s) to Optimally Solve MISP Instances

	Ddo (width: auto)	CODD (width: 128)	RPID (A*)	RPID (CABS)
johnson16-2-4	2.64	1.43	0.92	**0.62**
keller4	**5.47**	29.63	15.62	39.02
hamming6-2	**0.17**	0.28	1.56	9.07
hamming8-2	**0.30**	63.69	memory out	time out
hamming8-4	**29.65**	36.81	memory out	memory out
hamming10-2	**10.01**	1520.08	memory out	time out
brock200-1	**403.20**	time out	memory out	memory out
brock200-2	**1.10**	3.68	3.41	8.72
brock200-3	**7.88**	22.21	memory out	80.07
brock200-4	**22.67**	102.52	memory out	455.68
p_hat300-1	**0.49**	1.25	1.42	1.01
p_hat300-2	**19.82**	317.83	memory out	memory out

Summary

- Faster and more flexible interface for DIDP when writing a model in Rust is acceptable
- Algorithmic dual bound functions usually (but not always) improve solving performance
- No single winner in general-purpose DP solver frameworks

DIDP website: https://didp.ai

RPID code: https://github.com/domain-independent-dp/rpid

Model code: https://github.com/Kurorororo/didp-rust-models

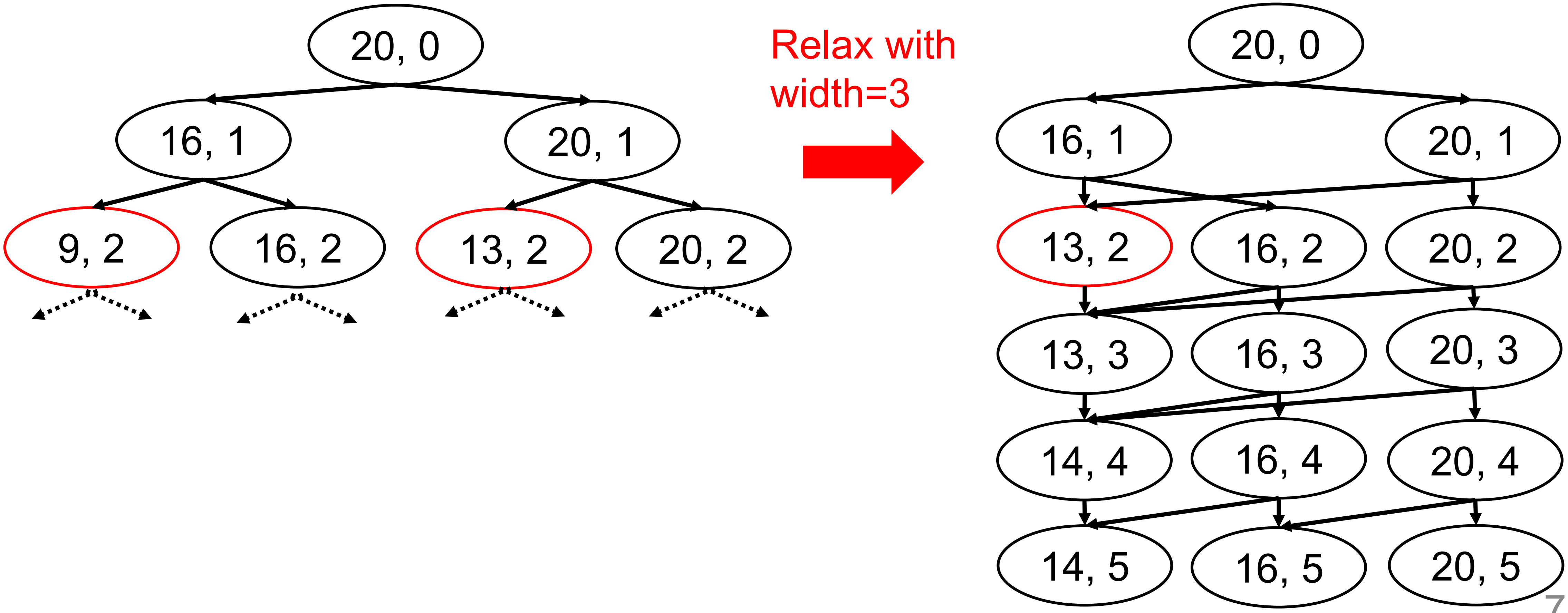
General-Purpose DP Solvers

Similarly to CP, a user formulates a declarative DP model and then solves it with a general-purpose solver

	Modeling interfaces	Solving algorithm
Ddo [Gillard et al. 2021]	**Rust trait** Python class	**decision diagram-based branch-and-bound**
CODD [Michel and van Hoeve 2024]	**C++ lambda**	**decision diagram-based branch-and-bound**
didp-rs [Kuroiwa and Beck 2023]	Rust expressions Python expressions YAML expressions	Heuristic state space search
RPID (this work)	Rust trait	Heuristic state space search

Decision Diagram-Based Branch-and-Bound

- Create multi-valued decision diagrams (MDDs) representing a model
- A dual bound is given by a relaxed DD where states are merged

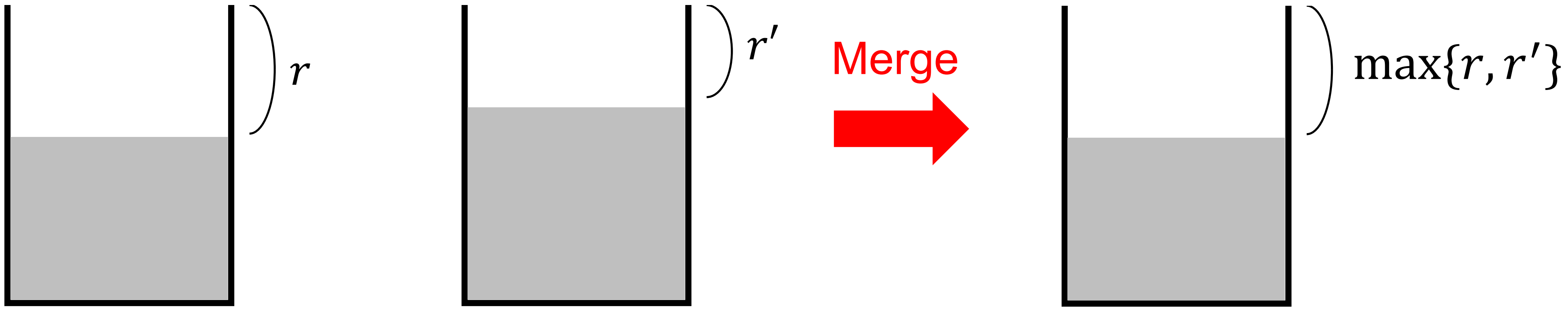


Merge Operator for Relaxed DDs

Merge states into one state better than or equal to the original states

E.g., merge states with the same item index i using a larger capacity

$$\oplus\big((r,i),(r',i)\big) = (\max\{r,r'\}, i)$$



Ddo Example for 0-1 Knapsack

A Rust trait with 7 methods and 2 additional traits for a merge operator

A trait for state dominance and a method for a dual bound are optional

```
impl Problem for Knapsack {
    type State = (isize, usize);

    fn initial_state(&self) -> Self::State { (self.c, 0) }
    fn for_each_in_domain(&self, variable: Variable, &(r: isize, _): &Self::State, f: &mut dyn DecisionCallback) {
        if r >= self.w[variable.id()] { f.apply(Decision { variable, value: 1 }); }
        f.apply(Decision { variable, value: 0});
    }
    fn transition(&self, &(r: isize, i: usize): &Self::State, dec: Decision) -> Self::State {
        if dec.value == 0 { (r, i + 1) } else { (r - self.w[dec.variable.id()], i + 1) }
    }
    fn transition_cost(&self, _: &Self::State, _: &Self::State, dec: Decision) -> isize {
        self.p[dec.variable.id()] * dec.value
    }
    fn nb_variables(&self) -> usize { self.n }
    fn initial_value(&self) -> isize { 0 }
    fn next_variable(&self, depth: usize, _: &mut dyn Iterator<Item = &Self::State>) -> Option<Variable> {
        if depth < self.nb_variables() { Some(Variable(depth)) } else { None }
    }
}
```

A Merge Operator and a Dual Bound in Ddo

```
struct KPRelax<'a>{pub pb: &'a Knapsack}
impl Relaxation for KPRelax<'_> {
    type State = (isize, usize);

    fn merge(&self, states: &mut dyn Iterator<Item = &Self::State>) -> Self::State {
        states.max_by_key(|&(r: &isize, _)| r).copied().unwrap()
    }
    fn relax(&self, _: &Self::State, _: &Self::State, _: &Self::State, _: Decision, cost: isize) -> isize { cost }
    fn fast_upper_bound(&self, &(r: isize, i: usize): &Self::State) -> isize {
        let mut bound: isize = 0;
        let mut r: isize = r;
        for j: usize in i..self.pb.n {
            if r >= self.pb.w[j] {
                bound += self.pb.p[j];
                r -= self.pb.w[j];
            } else {
                bound += (((r * self.pb.p[j]) as f64) / (self.pb.w[j] as f64)).floor() as isize;
            }
        }
        bound
    }
}
```

Sate Ranking and State Dominance in Ddo

```rust
struct KPRanking;
impl StateRanking for KPRanking {
    type State = (isize, usize);

    fn compare(&self, (r1: &isize, _): &Self::State, (r2: &isize, _): &Self::State) -> Ordering {
        r1.cmp(r2)
    }
}

1 implementation
struct KPDominance;
impl Dominance for KPDominance {
    type State = (isize, usize);
    type Key = usize;

    fn get_key(&self, state: Arc<Self::State>) -> Option<Self::Key> { Some(state.1) }
    fn nb_dimensions(&self, _: &Self::State) -> usize { 1 }
    fn get_coordinate(&self, &(r: isize, _): &Self::State, _: usize) -> isize { r }
    fn use_value(&self) -> bool { true }
}
```

CODD Example for 0-1 Knapsack

6 closures (C++ lambda) for DP and a merge operator in addition

Closures for state dominance and a dual bound function are optional

```cpp
    const auto init = [c]() { return State {c, 0}; };
    const auto lgf = [w](const State &s, DDContext) {
        return  Range::close(0, s.r >= w[s.i]);
    };
    const auto stf = [n, &w](const State &s, const int label) -> std::optional<State> {
      if (s.i < n-1) {
        return State { s.r - label * w[s.i], s.i + 1 };
      } else return State { 0, n };
    };
    const auto scf = [p](const State &s, int label) { return p[s.i] * label; };
    const auto target = [n]() { return State {0, n}; };
    const auto eqs = [n](const State &s) { return s.i == n; };
```

State Dominance and a Dual Bound in CODD

```
    const auto sdom = [](const State &s1, const State &s2) -> bool {
        return  s1.i == s2.i && s1.r >= s2.r;
    };
    const auto local = [n, &w, &p](const State &s, LocalContext) {
        double bound = 0;
        int r = s.r;
        for (int j = s.i; j < n; j++) {
            if (r >= w[j]) {
                r -= w[j];
                bound += p[j];
            } else {
                bound += std::floor(((double)r / w[j]) * p[j]);
            }
        }
        return bound;
    };
```

Traits vs. Closures

Traits

- Method signatures are explicit in example code and trait definitions
- A struct implementing traits is required in addition to a state struct

Closures

- Closure signatures are implicit in example code
- Succinct, no need to create a single struct with many fields as each closure can capture only necessary information

Successor Generation Approaches

3 steps to generate successor states:

1. Identify applicable transitions
2. Generate the successor states
3. Compute the transition weights

RPID does all in a single method

Ddo and CODD do each in a separate function

Successor Generation in RPID

```rust
struct Knapsack { n: usize, c: i32, p: Vec<i32>, w: Vec<i32> }

impl Dp for Knapsack {
    type State = (i32, usize);
    type CostType = i32;

    fn get_target(&self) -> Self::State { (self.c, 0) }
    fn get_successors(&self, &(r: i32, i: usize): &Self::State)
        -> impl IntoIterator<Item = (Self::State, Self::CostType, usize)> {
        if r >= self.w[i] {
            vec![((r - self.w[i], i + 1), self.p[i], 1), ((r, i + 1), 0, 0)]
        } else {
            vec![((r, i + 1), 0, 0)]
        }
    }
    fn get_base_cost(&self, &(_, i: usize): &Self::State) -> Option<Self::CostType> {
        if i == self.n { Some(0) } else { None }
    }
    fn get_optimization_mode(&self) -> OptimizationMode { OptimizationMode::Maximization }
}
```

The successor states, the transition weights, and the transition labels given a state

Successor Generation in Ddo

```
impl Problem for Knapsack {
    type State = (isize, usize);

    fn initial_state(&self) -> Self::State { (self.c, 0) }
    fn for_each_in_domain(&self, variable: Variable, &(r: isize, _): &Self::State, f: &mut dyn DecisionCallback) {
        if r >= self.w[variable.id()] { f.apply(Decision { variable, value: 1 }); }
        f.apply(Decision { variable, value: 0});
    }
    fn transition(&self, &(r: isize, i: usize): &Self::State, dec: Decision) -> Self::State {
        if dec.value == 0 { (r, i + 1) } else { (r - self.w[dec.variable.id()], i + 1) }
    }
    fn transition_cost(&self, _: &Self::State, _: &Self::State, dec: Decision) -> isize {
        self.p[dec.variable.id()] * dec.value
    }
    fn nb_variables(&self) -> usize { self.n }
    fn initial_value(&self) -> isize { 0 }
    fn next_variable(&self, depth: usize, _: &mut dyn Iterator<Item = &Self::State>) -> Option<Variable> {
        if depth < self.nb_variables() { Some(Variable(depth)) } else { None }
    }
}
```

Successor generation separated in 3 methods

Successor Generation in CODD

```
const auto init = [c]() { return State {c, 0}; };
const auto lgf = [w](const State &s, DDContext) {
    return  Range::close(0, s.r >= w[s.i]);
};
const auto stf = [n, &w](const State &s, const int label) -> std::optional<State> {
  if (s.i < n-1) {
    return State { s.r - label * w[s.i], s.i + 1 };
  } else return State { 0, n };
};
const auto scf = [p](const State &s, int label) { return p[s.i] * label; };
const auto target = [n]() { return State {0, n}; };
const auto eqs = [n](const State &s) { return s.i == n; };
```

Successor generation separated in 3 closures

Example: Single Machine Total Weighted Tardiness

Given a set of scheduled jobs S (a state), if job j is scheduled next (a transition), the tardiness becomes $(\sum_{k \in S} p_k) + p_j - d_j$ (the weight)

$$V(S) = \begin{cases} \min_{j \in N \setminus S} w_j \max\left\{\left(\sum_{k \in S} p_k\right) + p_j - d_j, 0\right\} + V(S \cup \{j\}) & \text{if } S \neq N \\ 0 & \text{if } S = N \end{cases}$$

In RPID, $\sum_{k \in S} p_k$ is computed once in `get_successors` and reused for each transition to schedule job j

Ddo and CODD may also avoid recomputation by caching it in a state (but with additional memory per state)

Q. Does generating all successor states in a single function have advantage in performance?

Performance of Successor Generation Methods

3 RPID models for the single machine total weighted tardiness:

- Original: $\sum_{k \in S} p_k$ is computed once for all transitions
- Separate: $\sum_{k \in S} p_k$ is computed for each transition to schedule job j
- StateCache: $\sum_{k \in S} p_k$ is used as an additional state variable

Performance of Successor Generation Methods

3 RPID models for the single machine total weighted tardiness:

- Original: $\sum_{k \in S} p_k$ is computed once for all transitions
- Separate: $\sum_{k \in S} p_k$ is computed for each transition to schedule job j
- StateCache: $\sum_{k \in S} p_k$ is used as an additional state variable

	Original		Separate		StateCache	
	#solved	average time (s)	#solved	average time (s)	#solved	average time (s)
A*	277	27	277	28	274	27
CABS	299	139	295	154	298	144