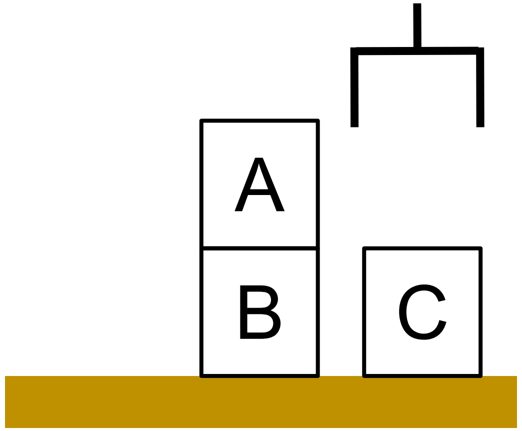
Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization

Ryo Kuroiwa and J. Christopher Beck

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

Domain-Independent Planning

Any planning problem



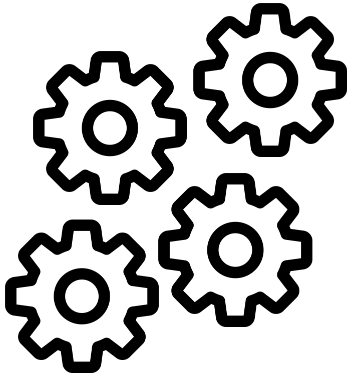
Model

State-based PDDL model

```
(define (domain BLOCKS)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x)
               )

  (:action pick-up
...........................
...........................
```

Solve

AI planner

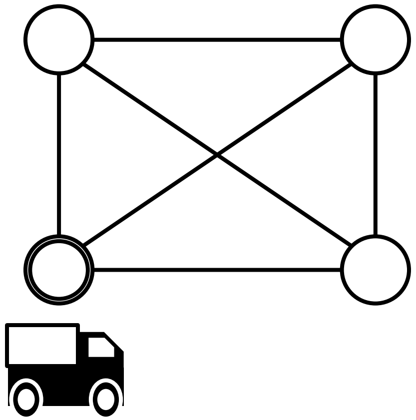


Heuristic search
is popular

What We Propose: DIDP

Domain-Independent Dynamic Programming (DIDP)

Any combinatorial optimization problem



Model

State-based DP model

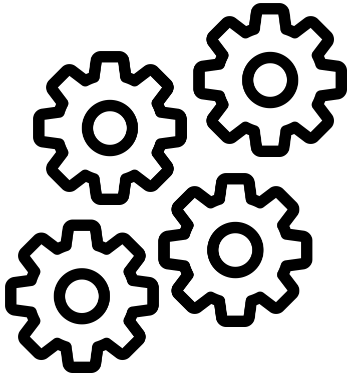
compute $V(N \setminus \{0\}, 0)$

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

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

Solve

DIDP solver



Current solvers are based on **heuristic search**

Our Modeling Interfaces

YAML (PDDL-like) **or** **Python library**

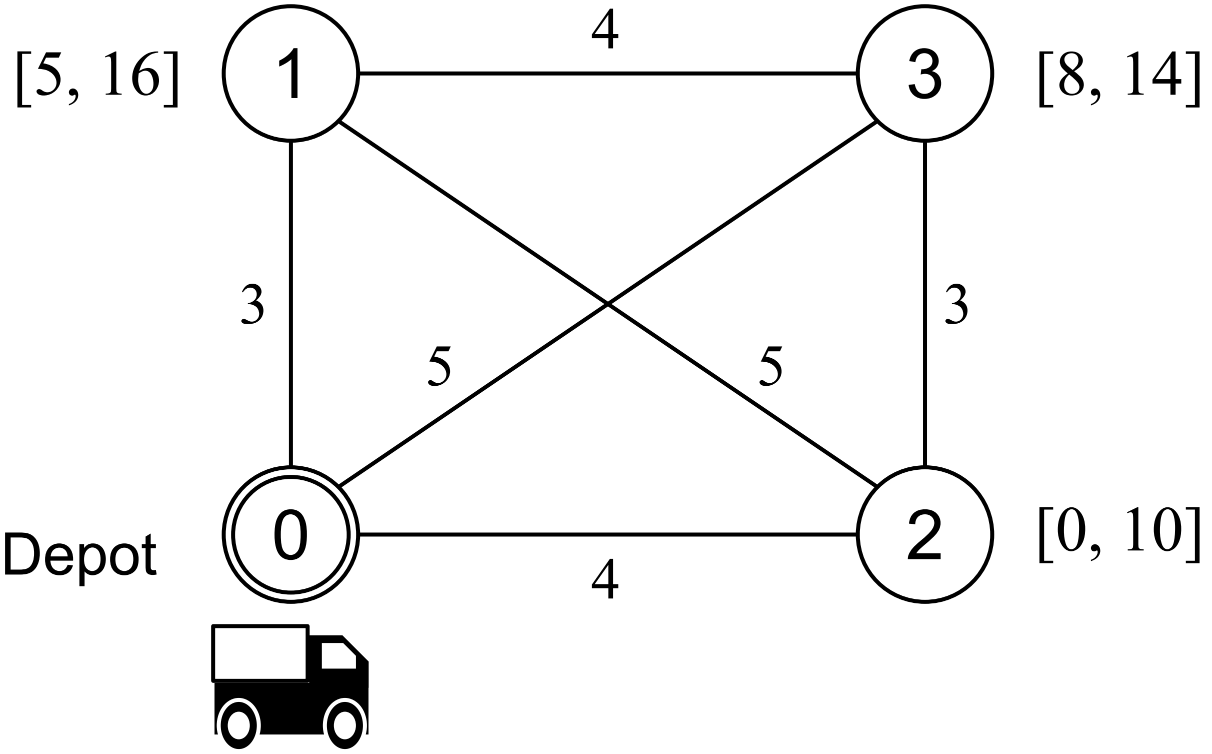
```yaml
objects:
  - customer
state_variables:
  - { name: unvisited, type: set, object: customer }
  - { name: location, type: element, object: customer }
tables:
  - name: travel_time
    type: integer
    args: [customer, customer]
transitions:
  - name: visit
    parameters: { name: j, object: unvisited }
    cost: (+ cost (travel_time location j))
    effect:
      unvisited: (remove j unvisited)
      location: j
  - name: return
    cost: (travel_time location 0)
    effect:
      location: 0
    preconditions:
      - (is_empty unvisited)
      - (!= location 0)
base_cases:
  - conditions:
      - (is_empty unvisited)
      - (= location 0)
```

```python
import didppy as dp

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

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

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

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

Background

Combinatorial Optimization

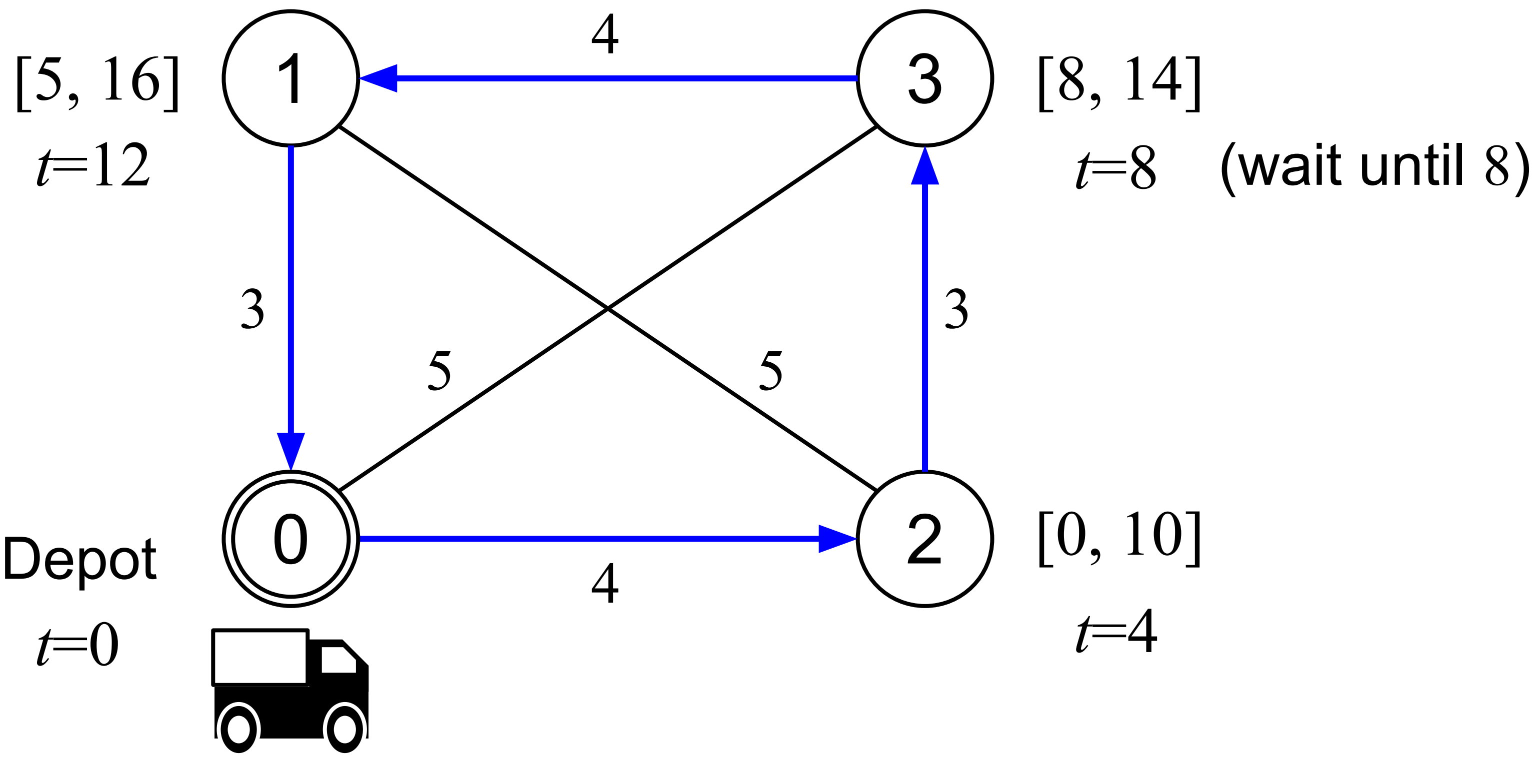
Traveling Salesperson Problem with Time Windows (TSPTW)
Minimize the travel time to visit all customers within time windows



Combinatorial Optimization

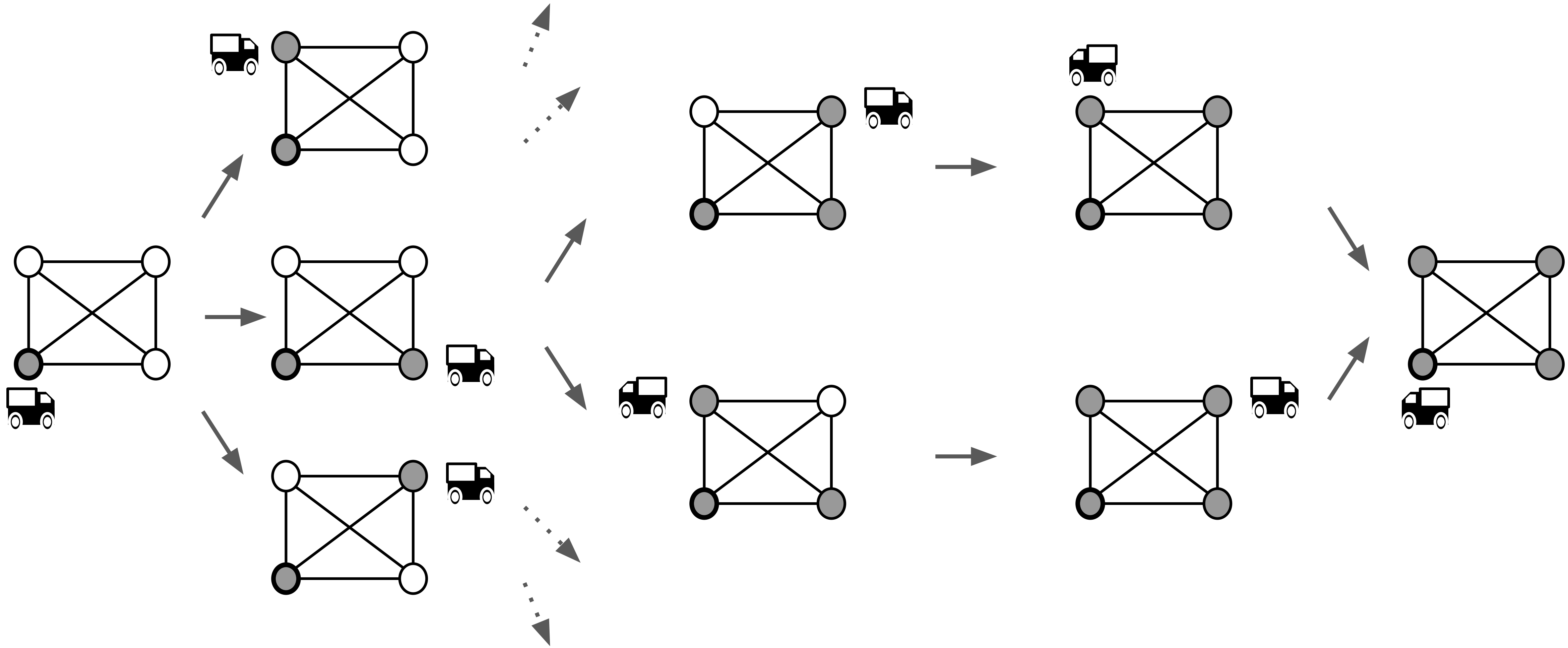
Traveling Salesperson Problem with Time Windows (TSPTW)
Minimize the travel time to visit all customers within time windows

Total cost: 14



DP for Combinatorial Optimization

State-based model: visit customers one by one



DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

Visit a customer

Return to the depot

Base case

State variables:

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

- Visit a customer
- Return to the depot
- Base case

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

Visit a customer

Return to the depot

Base case

State variables:

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

State variables:

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

State variables:

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

Visit a customer

Return to the depot

Base case

State variables:

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

State variables:

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

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

DP for Combinatorial Optimization

Recursive equations of a value function of a state

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

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

Visit a customer

Return to the depot

Base case

State variables:

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

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

Solved by problem-specific algorithm implementations before DIDP

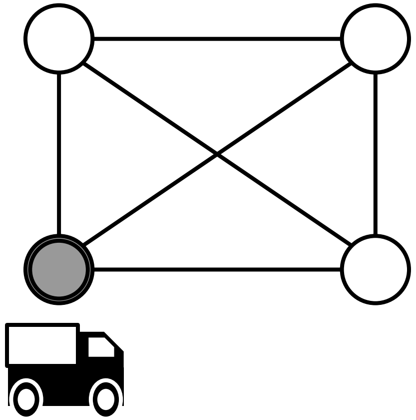
Our Modeling Formalism: DyPDL

State Variables

- Types: set, element, numeric
- Objective: compute the value of the **target state** (initial state)

Variable	Type	Domain	Target
U	set	$U \subseteq N$	$N \setminus \{0\}$
i	element	$i \in N$	0
t	numeric	$t \in \mathbb{Z}_0^+$	0

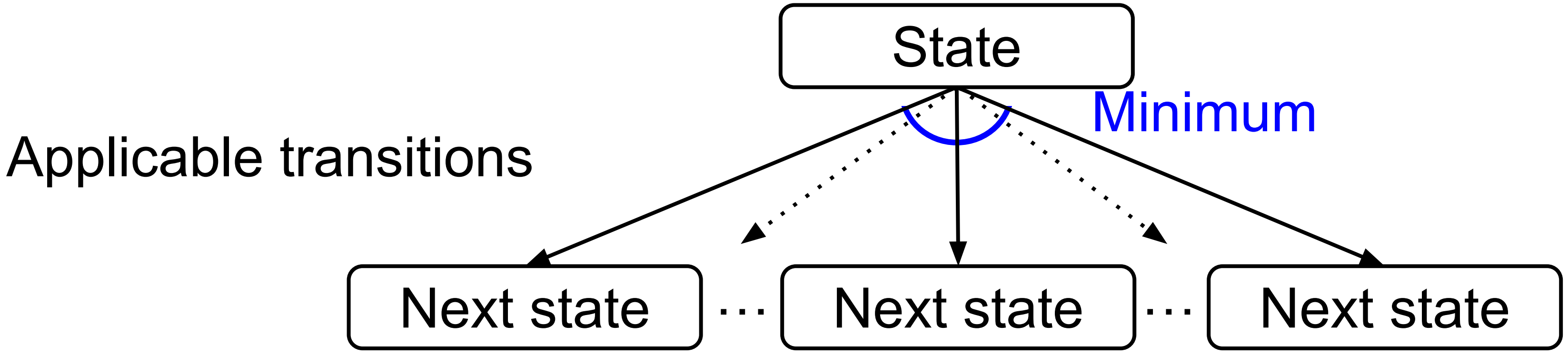
$\text{compute } V(N \setminus \{0\}, 0, 0)$



Transitions

- Define recursive equations by state transitions (actions)
- Value of a state: the minimum over all applicable transitions

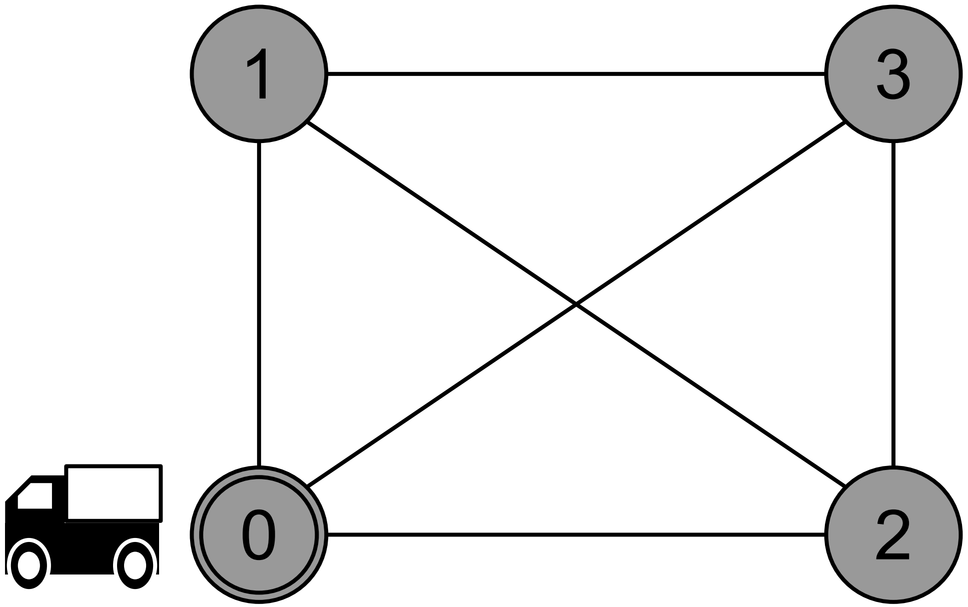
Transition to visit a customer	Expression
Cost expression and effects	$c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\})$
Preconditions to apply	$j \in U \wedge t + c_{ij} \leq b_j$



Base Cases

- Conditions to stop recursion (goal conditions)
- Value of a satisfying state: defined non-recursively

$$U = \emptyset \wedge i = 0 \rightarrow V(U, i, t) = 0$$



What DyPDL Can Do but PDDL Cannot

What DyPDL Can Do but PDDL Cannot

Explicitly modeling implications of the problem definition
(very useful and common in OR!)

What DyPDL Can Do but PDDL Cannot

Explicitly modeling implications of the problem definition (very useful and common in OR!)

Dominance based on **resource variables**

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

What DyPDL Can Do but PDDL Cannot

Explicitly modeling implications of the problem definition (very useful and common in OR!)

Dominance based on **resource variables**

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

Dual bound (LB in minimization)

$$V(U, i, t) \geq 0$$

What DyPDL Can Do but PDDL Cannot

Explicitly modeling implications of the problem definition (very useful and common in OR!)

Dominance based on **resource variables**

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

Dual bound (LB in minimization)

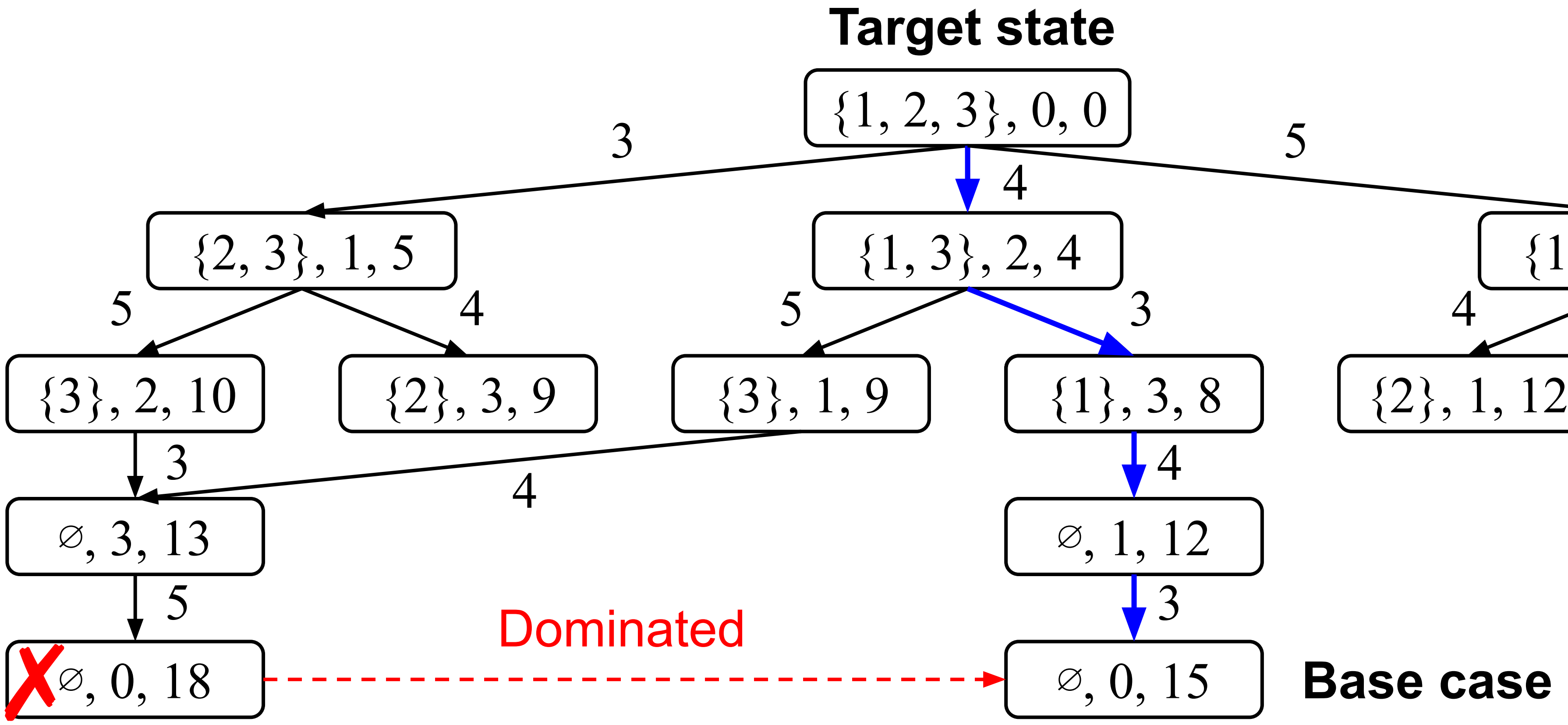
$$V(U, i, t) \geq 0$$

Other features skipped in this talk:

- State constraints
- Forced transitions

Our DIDP Solver: CAASDy

- Solve **DP as a shortest path** in the state space using A*
- **Heuristic: dual bound** defined in a DP model



Experimental Results

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

# of optimality solved instances with 8GB and 30-min

Future Work

We need your ideas to advance DIDP!

- Visit our website: https://didp.ai
- Start DIDP with Python: `pip install didppy`
  Tutorials and API Reference: https://didppy.rtfd.io
- Start DIDP with YAML: `cargo install didp-yaml`
- Clone the repository:
  `git clone https://github.com/domain-independent-dp/didp-rs`
  Everything in Rust 

Time vs. Coverage (Mean over All Problems)