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

State-based PDDL model

AI planner

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

(:action pick-up
  ..................
  ..................
)

Heuristic search is popular
What We Propose: DIDP

Domain-Independent Dynamic Programming (DIDP)

Any combinatorial optimization problem

State-based DP model

\[
\begin{align*}
\text{compute } & V(N \setminus \{0\}, 0) \\
V(U, i) & = \min_{j \in U} c_{ij} + V(U \setminus \{i\}, j) \\
V(\emptyset, i) & = c_{i0}.
\end{align*}
\]

DIDP solver

Current solvers are based on heuristic search
Our Modeling Interfaces

YAML (PDDL-like)

```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)
```

or

Python library

```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
Recursive equations of a value function of a state

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

- 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 \(j\)
Recursive equations of a value function of a state

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

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

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

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

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

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 $j$
Recursive equations of a value function of a state

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

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 \( j \)
Recursive equations of a value function of a state

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

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

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 $j$
Recursive equations of a value function of a state

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

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

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

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

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)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Domain</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U$</td>
<td>set</td>
<td>$U \subseteq N$</td>
<td>$N \setminus {0}$</td>
</tr>
<tr>
<td>$i$</td>
<td>element</td>
<td>$i \in N$</td>
<td>0</td>
</tr>
<tr>
<td>$t$</td>
<td>numeric</td>
<td>$t \in \mathbb{Z}_0^+$</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Transition to visit a customer</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost expression and effects</td>
<td>$c_{ij} + V(U \setminus {j}, j, \max{t + c_{ij}, a_j})$</td>
</tr>
<tr>
<td>Preconditions to apply</td>
<td>$j \in U \land t + c_{ij} \leq b_j$</td>
</tr>
</tbody>
</table>

Applicable transitions

State

Minimum

Next state

...
Base Cases

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

\[ U = \emptyset \land i = 0 \rightarrow V(U, i, t) = 0 \]
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  Dual bound (LB in minimization)

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

\[ V(U, i, t) \geq 0 \]

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

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

# 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)