Domain-Independent Dynamic Programming for Combinatorial Optimization

J. Christopher Beck & Ryo Kuroiwa
Department of Mechanical & Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca
This is not a talk about Decision Diagrams

What it is about

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

Problem    Problem Definition
Model-and-Solve

Problem

Problem Definition

Models

General Purpose Solver

\[ \sum \]

Problem Definition

Models

CP, LP, MIP, MINLP, AI Planning, …

A Solution!
Model-and-Solve for DP

- Domain-independent dynamic programming (DIDP)

Define models using DP transition system

Solve models using heuristic state-based search
Open Source Software: didp-rs

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

- **Interface**
  - DIDPPy (Python)
  - DIDP-YAML

- **Core Libraries**
  - DyPDL (Modeling)
  - DyPDL Heuristic Search (Solver)

- Implemented in Rust
Outline

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

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

![Graph of a TSPTW problem](image)

- **Depot**: 0
- **Customers**: 1, 2, 3
- **Time Windows**:
  - Customer 1: [5, 16]
  - Customer 3: [8, 14]
  - Customer 2: [0, 10]
- **Travel Times** (in units of time):
  - 0 to 1: 4
  - 0 to 2: 4
  - 1 to 2: 3
  - 1 to 3: 5
  - 2 to 3: 3
Combinatorial Optimization

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

Total cost: 14
Recursive equations for the value function of a state (subproblem)

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

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

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

Visit a customer
Return to the depot
Base case
Recursive equations for the value function of a state (subproblem)

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

Visit a customer  
Return to the depot  
Base case

**State variables:**
- \( U \): unvisited customers  
- \( i \): current customer  
- \( t \): current time

**Constants**
- \( N \): all customers (0: depot)  
- \([a_i, b_i]\): time window for customer \( i \)  
- \( c_{ij} \): travel time from customer \( i \) to \( j \)
Recursive equations for the value function of a state (subproblem)

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

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

Visit a customer
Return to the depot
Base case

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

Constants
- $N$: all customers (0: depot)
- $[a_i, b_i]$: time window for customer $i$
- $c_{ij}$: travel time from customer $i$ to $j$
Recursive equations for the value function of a state (subproblem)

\
\begin{align*}
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}
\end{align*}

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

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

Visit a customer
Return to the depot
Base case
DP for Combinatorial Optimization

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

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

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

Visit a customer
Return to the depot
Base case

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

Constants
- $N$: all customers (0: depot)
- $[a_i, b_i]$: time window for customer $i$
- $c_{ij}$: travel time from customer $i$ to $j$
Recursive equations for the value function of a state (subproblem)

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

Visit a customer
Return to the depot
Base case

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

Constants
- \( N \): all customers (0: depot)
- \([a_i, b_i]\): time window for customer \( i \)
- \( c_{ij} \): travel time from customer \( i \) to \( j \)
Recursive equations for the value function of a state (subproblem)

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

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

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

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

**Constants**
- $N$ : all customers (0: depot)
- $[a_i, b_i]$ : time window for customer $i$
- $c_{ij}$ : travel time from customer $i$ to $j$
Recursive equations for the value function of a state (subproblem)

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

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

Visit a customer
Return to the depot
Base case

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

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

DP usually solved by problem-specific algorithm implementations
Our Modeling Interface: DIDPPy
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)
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)
import didppy as dp

model = dp.Model(maximize=False)

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

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

```python
import didppy as dp

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)
```

**Customers**
\[ N = \{0, 1, 2, 3\} \]

**Ready time**
\[ a_i \]

**Deadline**
\[ b_i \]

**Travel time**
\[ c_{ij} \]
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]])

To use state variable i for indexing

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)
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]])

State variables

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)

Unvisited \( U \subseteq N \)
Current \( i \in N \)
Time \( t \in \mathbb{Z} \)
import didppy as dp

model = dp.Model(maximize=False)

customer = model.add_object_type(number=4)

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

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

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

Questions?
Recursive Equation as Transitions

for j in range(1, 4):
    \[ V(U, i, t) = \min_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) \]
    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)
Recursive Equation as Transitions

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

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)

... for each value of \( i \in \mathbb{N} \)
Recursive Equation as Transitions

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

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)
Recursive Equation as Transitions

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

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)

When the transition is applicable
Recursive Equation as Transitions

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

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)

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

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

```python
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)
```
Base Cases: When to Stop Recursion

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

End of recursion on $V$
Better Model with Redundant Information

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

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

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

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

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

A dual bound is defined for a state

Other features not detailed here:

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

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

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

print("Solution:")

for transition in solution.transitions:
    print(transition.name)
DP as a Shortest Path Problem

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

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

$\{1, 2, 3\}, 0, 0$
CAASDy: Prototype Solver for DIDP

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

\[ \{1, 2, 3\}, 0, 0 \]

\[ V(U, i, t) \geq 0 \quad f: 0 \]
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

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

\[
\begin{align*}
\{1, 2, 3\}, 0, 0 \\
\{2, 3\}, 1, 5 \\
\{1, 3\}, 2, 4 \\
{1, 2}, 3, 8 \\
\{3\}, 2, 10 \\
\{2\}, 3, 9 \\
\{3\}, 1, 9 \\
\{1\}, 3, 8 \\
\{2\}, 1, 12 \\
\end{align*}
\]

\[f: 8 \quad f: 9 \quad f: 7 \quad f: 9\]

No transitions
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

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

\[
\begin{align*}
\{1\}, 3, 8 & \quad \{3\}, 1, 9 \\
\{2\}, 1, 12 & \quad \{1, 2\}, 3, 8 \\
\{2, 3\}, 1, 5 & \quad \{1, 3\}, 2, 4 \\
\{2\}, 3, 9 & \quad \{3\}, 2, 10 \\
\{3\}, 1, 10 & \quad \{2\}, 1, 12 \\
\emptyset, 3, 13 & \quad \emptyset, 1, 12
\end{align*}
\]
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

- Solves DP as a shortest path problem with A* search
- A* searches in the order of $f$ (path cost + dual bound of a state)
CAASDy: Prototype Solver for DIDP

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

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

$f: 14$
CAASDy: Prototype Solver for DIDP

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

\[
egin{array}{cccc}
\{1\}, 3, 8 & \{2\}, 1, 12 & \emptyset, 3, 13 & \emptyset, 0, 18 \\
\{1, 3\}, 2, 4 & \{2\}, 3, 9 & \{3\}, 1, 9 & \emptyset, 1, 12 \\
\{2, 3\}, 1, 5 & \{3\}, 2, 10 & \emptyset, 0, 18 & \emptyset, 0, 15 \\
\{1, 2, 3\}, 0, 0 & & & \\
\end{array}
\]
CAASDy: Prototype Solver for DIDP

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

```
{1, 2, 3}, 0, 0

{1, 2}, 3, 8
{3}, 2, 10
∅, 3, 13
∅, 0, 18

{2, 3}, 1, 5
{2}, 3, 9
∅, 0, 15

{1, 3}, 2, 4
{3}, 1, 9
∅, 1, 12

∅, 3, 13
{2}, 1, 12
∅, 1, 12
```

Dominated
CAASDy: Prototype Solver for DIDP

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

 Guaranteed to be an optimal solution

<table>
<thead>
<tr>
<th>State</th>
<th>Path</th>
<th>Travel time (w/o waiting)</th>
<th>Finishing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0, 18</td>
<td>0, 18</td>
<td>✓</td>
</tr>
<tr>
<td>${2}$, 1, 5</td>
<td>3</td>
<td>$f$: 14</td>
<td></td>
</tr>
<tr>
<td>${1, 2}$, 3, 8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1}$, 3, 8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${2}$, 1, 12</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\emptyset$, 1, 12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${3}$, 1, 9</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1, 3}$, 2, 4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${2, 3}$, 1, 5</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\emptyset$, 0, 13</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${3}$, 2, 10</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\emptyset$, 0, 15</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1, 2, 3}$, 0, 0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison of MIP, CP, and DIDP

<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 optimally solved instances with 8 GB and 30 minutes
Anytime DIDP Solvers
Anytime Solvers

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

Can we develop anytime solvers for DIDP?
## Anytime State Space Search Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth First Branch-and-Bound (DFBnB)</td>
<td>DFS</td>
<td></td>
</tr>
<tr>
<td>Cyclic Best-First Search (CBFS)</td>
<td>Hybrid of DFS and best-first search</td>
<td>Kao et al. 2009</td>
</tr>
<tr>
<td>Anytime Column Progressive Search (ACPS)</td>
<td>Hybrid of DFS and beam search</td>
<td>Vadlamudi et al. 2012</td>
</tr>
<tr>
<td>Discrepancy-Bounded DFS (DBDFS)</td>
<td>Discrepancy-based</td>
<td>Beck and Perron 2000</td>
</tr>
<tr>
<td>Complete Anytime Beam Search (CABS)</td>
<td>Iterative beam search</td>
<td>Zhang 1998</td>
</tr>
</tbody>
</table>

Implemented in [https://github.com/domain-independent-dp/didp-rs](https://github.com/domain-independent-dp/didp-rs)
## Anytime State Space Search Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth First Branch-and-Bound (DFBnB)</td>
<td>DFS</td>
<td>Kao et al. 2009</td>
</tr>
<tr>
<td>Cyclic Best-First Search (CBFS)</td>
<td>Hybrid of DFS and best-first search</td>
<td>Kao et al. 2009</td>
</tr>
<tr>
<td>Anytime Column Progressive Search (ACPS)</td>
<td>Hybrid of DFS and beam search</td>
<td>Vadlamudi et al. 2012</td>
</tr>
<tr>
<td>Discrepancy-Bounded DFS (DBDFS)</td>
<td>Discrepancy-based</td>
<td>Beck and Perron 2000</td>
</tr>
<tr>
<td>Complete Anytime Beam Search (CABS)</td>
<td>Iterative beam search</td>
<td>Zhang 1998</td>
</tr>
</tbody>
</table>

Implemented in [https://github.com/domain-independent-dp/didp-rs](https://github.com/domain-independent-dp/didp-rs)
Beam Search

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

\[
b = 2 \quad \begin{cases} \{1, 2, 3\}, 0, 0 \\ f: 0 \end{cases}
\]
Beam Search

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

$b = 2$  
\[ \{1, 2, 3\}, 0, 0 \]  
\[ f: 0 \]
Beam Search

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

$b = 2$

- $\{2, 3\}, 1, 5$ with $f: 3$
- $\{1, 2, 3\}, 0, 0$ with $f: 4$
- $\{1, 3\}, 2, 4$ with $f: 5$
- $\{1, 2\}, 3, 8$ with $f: 5$
Beam Search

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

$b = 2$

\[
\begin{align*}
{2, 3}, 1, 5 &\quad f: 3 \\
{1, 2, 3}, 0, 0 &\quad f: 4 \\
{1, 3}, 2, 4 &\quad f: 5 \\
{1, 2}, 3, 8 &
\end{align*}
\]
Beam Search

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

$b = 2$

- {2, 3}, 1, 5
  - {3}, 2, 10, $f$: 8
  - {2}, 3, 9, $f$: 7

- {1, 3}, 2, 4
  - {3}, 1, 9, $f$: 9
  - {1}, 3, 8, $f$: 7

- {1, 2, 3}, 0, 0
  - {1, 2}, 3, 8
Beam Search

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

$b = 2$

$\{2, 3\}, 1, 5$

$\{2\}, 3, 9$

$\{3\}, 2, 10$

$\{2\}, 3, 9$

$\{3\}, 1, 9$

$\{1\}, 3, 8$

$\{3\}, 1, 9$

$\{1\}, 3, 8$

$\{1, 2\}, 3, 8$

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

$\{1\}, 3, 8$

$\{1\}, 3, 8$

$\{1\}, 3, 8$

$\{2\}, 3, 9$

$\{1\}, 3, 8$
Beam Search

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

$b = 2$

<table>
<thead>
<tr>
<th>State</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2, 3}</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>{1, 2}</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>{1}</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>{1, 3}</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>{3}</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

No transitions
Beam Search

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

$b = 2$

No transitions

$f$: 11

\[
\begin{array}{c|c|c}
\{3\}, & 2, & 10 \\
\{2\}, & 3, & 9 \\
\{3\}, & 1, & 9 \\
\{1\}, & 3, & 8 \\
\emptyset, & 1, & 12 \\
\end{array}
\]
Beam Search

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

$b = 2$

\[
\begin{align*}
\{2, 3\}, & \quad 1, 5 \\
\{3\}, & \quad 2, 10 \\
\{2\}, & \quad 3, 9 \\
\{3\}, & \quad 1, 9 \\
\{1, 3\}, & \quad 2, 4 \\
\{3\}, & \quad 1, 9 \\
\{1\}, & \quad 3, 8 \\
\{1\}, & \quad 3, 8 \\
\emptyset, & \quad 1, 12 \\
\emptyset, & \quad 0, 15 \\
\end{align*}
\]
Beam Search

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

$b = 2$

$\{2, 3\}, 1, 5$

$\{3\}, 2, 10$

$\{2\}, 3, 9$

$\{3\}, 1, 9$

$\{1, 3\}, 2, 4$

$\{1\}, 3, 8$

$\{1\}, 3, 8$

$\{1, 2\}, 3, 8$

$\emptyset, 1, 12$

$\emptyset, 0, 15$

$f: 14$
Beam Search

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

$$b = 2$$

$$\{2, 3\}, 1, 5\quad 3\quad \{1, 2, 3\}, 0, 0\quad 5\quad \{1, 2\}, 3, 8$$

$$\{2\}, 3, 9\quad 4\quad \{1, 3\}, 2, 4\quad 5\quad \emptyset, 1, 9\quad 3\quad \emptyset, 3, 8\quad 3\quad \emptyset, 0, 15\quad 3$$

$$f: 14$$
Beam Search

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

$b = 2$

$\{2, 3\}, 1, 5$

$\{3\}, 2, 10$
$\{2\}, 3, 9$

$\{1, 3\}, 2, 4$

$\{3\}, 1, 9$
$\{1\}, 3, 8$

$\emptyset, 1, 12$
$\emptyset, 0, 15$

$f: 14$
Complete Anytime Beam Search (CABS)

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

\( b = 8, \text{ incumbent: 14} \)

\[
\{1, 2, 3\}, \ 0, \ 0
\]

\( f: 0 \)

[Zhang 1998]
Complete Anytime Beam Search (CABS)

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

$b = 8$, incumbent: 14

[Zhang 1998]
Complete Anytime Beam Search (CABS)

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

$b = 8$, incumbent: 14

\[
\begin{align*}
\{2, 3\}, 1, 5 & \quad 3 \quad \{1, 2, 3\}, 0, 0 \\
\{3\}, 2, 10 & \quad 4 \quad \{1, 3\}, 2, 4 \\
\{2\}, 3, 9 & \quad 5 \quad \{3\}, 1, 9 \\
 & \quad 5 \quad \{1\}, 3, 8 \\
 & \quad 4 \quad \{2\}, 1, 12 \\
\end{align*}
\]

\[
\begin{align*}
f: 8 & \quad f: 7 & \quad f: 9 & \quad f: 7 & \quad f: 9 \\
\end{align*}
\]

[Zhang 1998]
Complete Anytime Beam Search (CABS)

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

$b = 8$, incumbent: 14

[Zhang 1998]
Complete Anytime Beam Search (CABS)

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

$b = 8$, incumbent: 14

\[
\begin{array}{c|c|c}
\{3\}, 2 & 10 & 3 \\
\{2\}, 3 & 9 & 4 \\
\{3\}, 1 & 9 & 5 \\
\emptyset, 3 & 13 & 5 \\
\emptyset, 0 & 18 & f: 16 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\{1, 2, 3\}, 0 & 0 & 3 \\
\{1, 3\}, 2 & 4 & 5 \\
\{1\}, 3 & 8 & 5 \\
\{2\}, 1 & 12 & 5 \\
\{2\}, 1 & 12 & 4 \\
\emptyset, 1 & 12 & 3 \\
\emptyset, 0 & 15 & f: 14 \\
\end{array}
\]

[Zhang 1998]
Complete Anytime Beam Search (CABS)

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

$\begin{align*}
\{1\}, 3, 8 & \quad \{1\}, 3, 8 \\
\{3\}, 1, 9 & \quad \{1\}, 3, 8 \\
\{2\}, 3, 9 & \quad \{2\}, 1, 12 \\
\{3\}, 2, 10 & \quad \{1, 2\}, 3, 8 \\
\{2, 3\}, 1, 5 & \quad \{1, 2, 3\}, 0, 0
\end{align*}$

Proved the optimality

$\begin{align*}
\{\emptyset\}, 1, 12 & \quad \{\emptyset\}, 0, 15 \\
\{\emptyset\}, 3, 13 & \quad \{\emptyset\}, 0, 18
\end{align*}$

$\begin{align*}
\text{ incumbent: } & 14 \\
f: & 16 \\
f: & 14
\end{align*}$

[Zhang 1998]
Experimental Evaluation of CABS
# of Optimally Solved by CABS

<table>
<thead>
<tr>
<th>Description</th>
<th>MIP</th>
<th>CP</th>
<th>CAASDy</th>
<th>CABS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSPTW (340)</td>
<td>227</td>
<td>47</td>
<td>257</td>
<td>259</td>
</tr>
<tr>
<td>CVRP (207)</td>
<td>26</td>
<td>0</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>SALBP-1 (2100)</td>
<td>1357</td>
<td>1584</td>
<td>1653</td>
<td>1801</td>
</tr>
<tr>
<td>Bin Packing (1615)</td>
<td>1157</td>
<td></td>
<td>1234</td>
<td>922</td>
</tr>
<tr>
<td>MOSP (570)</td>
<td>225</td>
<td>437</td>
<td>483</td>
<td>527</td>
</tr>
<tr>
<td>Graph-Clear (135)</td>
<td>24</td>
<td>4</td>
<td>76</td>
<td>103</td>
</tr>
<tr>
<td>Talent Scheduling (1000)</td>
<td>6</td>
<td>7</td>
<td>224</td>
<td>253</td>
</tr>
<tr>
<td>m-PDTSP (1117)</td>
<td>945</td>
<td></td>
<td>1049</td>
<td>947</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>∑w_iT_i (375)</td>
<td>109</td>
<td>150</td>
</tr>
</tbody>
</table>

# of optimally solved instances with 8 GB and 30 minutes
Primal Integral

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

[Berthold 2013]
## Mean Primal Gap/Primal Integral

<table>
<thead>
<tr>
<th>Description</th>
<th>MIP</th>
<th>CP</th>
<th>CABS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSPTW (340)</td>
<td>0.227/484.1</td>
<td>0.026/49.0</td>
<td>0.003/9.0</td>
</tr>
<tr>
<td>CVRP (207)</td>
<td>0.585/1157.4</td>
<td>0.317/601.2</td>
<td>0.185/351.2</td>
</tr>
<tr>
<td>SALBP-1 (2100)</td>
<td>0.345/634.6</td>
<td>0.005/28.5</td>
<td>0.000/1.9</td>
</tr>
<tr>
<td>Bin Packing (1615)</td>
<td>0.039/86.2</td>
<td>0.002/8.0</td>
<td>0.002/5.3</td>
</tr>
<tr>
<td>MOSP (570)</td>
<td>0.039/100.4</td>
<td>0.004/13.0</td>
<td>0.000/0.4</td>
</tr>
<tr>
<td>Graph-Clear (135)</td>
<td>0.110/311.8</td>
<td>0.015/44.3</td>
<td>0.000/0.5</td>
</tr>
<tr>
<td>Talent Scheduling (1000)</td>
<td>0.051/142.7</td>
<td>0.002/18.1</td>
<td>0.011/26.4</td>
</tr>
<tr>
<td>m-PDTSP (1178)</td>
<td>0.078/180.0</td>
<td>0.013/26.0</td>
<td>0.002/5.3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>∑wiTi (375)</td>
<td>0.018/74.6</td>
</tr>
</tbody>
</table>

Best paper runner-up at ICAPS 2023 [Kuroiwa and Beck 2023b]
Current & Future Work
## DIDP Papers at CP

### Tuesday, August 29th

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:00-16:50</td>
<td>14B</td>
<td>Applications 3</td>
</tr>
</tbody>
</table>
| 16:00    |           | **Amoosh Golestanian, Giovanni Lo Bianco, Chengyu Tao and J. Christopher Beck**  
Optimization models for pickup and delivery problems with reconfigurable capacities  |

### Thursday, August 31st

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:20-14:20</td>
<td>27A</td>
<td>Search 3</td>
</tr>
</tbody>
</table>
| 13:50    |           | **Ryo Kuroiwa and J. Christopher Beck**  
Large Neighborhood Beam Search for Domain-Independent Dynamic Programming  |
Building a Parallel Solver

- **Parallel**
  - Gurobi
  - CPLEX
  - SCIP
  - OR Tools
  - Xpress
  - CP Optimizer

- **Sequential**
  - CAASDy
  - CABS

- **Open source**

- **Not anytime**

- **Anytime**
### Comparison of Solvers with 32 Threads

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

# of optimally solved instances/speedup with 32 threads, 19 2GB, and 5 minutes
What Makes a Good Model?

What DP models are good/bad?

**DP Model 1**
- Compute $V(N \setminus \{0\}, 0)$
- $V(U, i) = \min_{j \in U} c_{ij} + V(U \setminus \{i\}, j)$
- $V(\emptyset, i) = c_{i0}$

**DP Model 2**
- Compute $V(\emptyset, 0)$
- $V(U, i) = \min_{j \notin U} c_{ji} + V(U \cup \{i\}, j)$
- $V(N \setminus \{0\}, i) = c_{0i}$
Can we automatically derive a dual bound function from a model?

Domain-Independent Dual Bound Function

Problem

Model

DP Model

DIDP solver

Derive

Solve

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

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

\[ V(U, i) \geq h(U, i) \]

Dual bound function
Empirical Analysis of DIDP Search

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

![Figure 1. Random 4-SAT problems, tested using ASAT, mean (solid), median (dashed) branches, N = 75](image)

[Gent and Walsh 1994]
Please Use DIDP on Your Problems!

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

Questions?