Parallel Beam Search Algorithms for Domain-Independent Dynamic Programming

Ryo Kuroiwa and J. Christopher Beck
Toronto Intelligent Decision Engineering Laboratory (TIDEL)
Department of Mechanical and Industrial Engineering,
University of Toronto
Background
Domain-Independent Dynamic Programming (DIDP)

A user can solve a combinatorial optimization problem by formulating a Dynamic Programming (DP) model

**Combinatorial Optimization Problem**

**DP Model**

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

**General-Purpose DP Solver**

Current solvers (in Rust) use **heuristic search**

**Solution**

Recently proposed by us [Kuroiwa and Beck 2023b]
Example: DP Model for TSP

- TSP: Minimize the total travel cost to visit all customers and return
- DP: State space representation of the problem
  Customers are visited one by one in TSP
Example of DIDP with Python

```python
import didppy as dp

model = dp.Model(maximize=False)
customer = model.add_object_type(number=4)
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)

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)],
        preconditions=[u.contains(j)],
    )
    model.add_transition(visit)

model.add_base_case([u.is_empty()], cost=c[i, 0])
model.add_dual_bound(0)
solver = dp.CABS(model, threads=32)
solution = solver.search()
```

Define constants and state variables

Define transitions between states

Define goal conditions

Call a solver
Example of DIDP with Python

```python
import didppy as dp

model = dp.Model(maximize=False)
customer = model.add_object_type(number=4)
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)

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)],
        preconditions=[i.contains(j)],
    
model.add_transition(visit)

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

model.add_dual_bound(0)  # Contribution of this paper

solver = dp.CABS(model, threads=32)
solution = solver.search()
```

Define constants and state variables

Define transitions between states

Define goal conditions

Call a solver
Solving the DP model by finding a path in the state transition graph

\( f \)-value: priority to expand, \( g + h \)

\( g \)-value: actual path cost

\( h \)-value: estimation by a heuristic function (given with a DP model in current DIDP)
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]

\[ b = 2 \]

Initial state  

\[ \bigcirc \]
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer.
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]

\[ b = 2 \]
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer.
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998].

$$b = 2$$
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]

\[ b = 2 \]
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer.
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998].

$b = 2$
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer.
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]

$b = 2$
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]

$b = 2$

![Diagram](image)
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

• Beam search expands the $b$ states minimizing $f$-values in each layer
• Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the $b$ states minimizing $f$-values in each layer.
- Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998].

$$b = 2$$
CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

• Beam search expands the $b$ states minimizing $f$-values in each layer
• Complete Anytime Beam Search (CABS) repeats beam search with increasing $b$ until finding an optimal solution [Zhang 1998]

$b = 2$
Parallel Beam Search Algorithms
Approach 1: Shared Beam Search (SBS)

• Expand the best $b$ states (obtained by parallel sort) in parallel
• Use a concurrent hash table for duplicate detection

Divided into multiple shards, and each shard has a lock

Shard 1

<table>
<thead>
<tr>
<th>key1</th>
<th>value1</th>
</tr>
</thead>
<tbody>
<tr>
<td>key4</td>
<td>value4</td>
</tr>
<tr>
<td>key7</td>
<td>value7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Shard 2

<table>
<thead>
<tr>
<th>key2</th>
<th>value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>key5</td>
<td>value5</td>
</tr>
<tr>
<td>key8</td>
<td>value8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Shard 3

<table>
<thead>
<tr>
<th>key3</th>
<th>value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>key6</td>
<td>value6</td>
</tr>
<tr>
<td>key9</td>
<td>value9</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Similar to problem-specific parallel beam search by Frohner+ (2023)
Approach 2: Hash Distributed Beam Search (HDBS)

• Send a state to a thread determined by its hash value using message passing (duplicate states sent to the same thread)

• Each thread locally detects duplicates and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

\[b = 2, \#\text{threads} = 2\]

Assignment to a thread

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

• Send a state to a thread determined by its hash value using message passing (duplicate states sent to the same thread)

• Each thread locally detects duplicates and expands \( \frac{b}{\#\text{threads}} \) states

\[ b = 2, \#\text{threads} = 2 \]

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

• Send a state to a thread determined by its hash value using message passing (duplicate states sent to the same thread)

• Each thread locally detects duplicates and expands \( \frac{b}{\text{#threads}} \) states

\[ b = 2, \text{#threads} = 2 \]

Sent to the assigned thread

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{#\text{threads}}$ states

$b = 2, #\text{threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\# \text{threads}}$ states

$b = 2, \# \text{threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its hash value using message passing (duplicate states sent to the same thread)
- Each thread locally detects duplicates and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its hash value using message passing (duplicate states sent to the same thread)
- Each thread locally detects duplicates and expands $\frac{b}{\text{#threads}}$ states

$b = 2, \text{#threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its hash value using message passing (duplicate states sent to the same thread)
- Each thread locally detects duplicates and expands $\frac{b}{\# \text{threads}}$ states

$b = 2, \# \text{threads} = 2$

Expanded states are different from sequential beam search

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands \( \frac{b}{\text{#threads}} \) states

\[ b = 2, \#\text{threads} = 2 \]

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search
HDBS1: Immediate Layer Synchronization

Thread 1

Expand the best states in layer $i$

Send successors in layer $i + 1$

Receive successors in layer $i + 1$

Go to the next layer ($i \leftarrow i + 1$)

Thread 2

Expand the best states in layer $i$

Send successors in layer $i + 1$

Receive successors in layer $i + 1$

Go to the next layer ($i \leftarrow i + 1$)

Synchronize
HDBS2: Delayed Layer Synchronization

Thread 1

1. Expand the best states in layer $i$
2. Check if all threads finish layer $i - 1$
3. Send successors in layer $i + 1$
4. Receive successors in layer $i + 1$
5. Notify that layer $i$ is finished
   Go to the next layer ($i \leftarrow i + 1$)

Thread 2

1. Expand the best states in layer $i$
2. Check if all threads finish layer $i - 1$
3. Send successors in layer $i + 1$
4. Receive successors in layer $i + 1$
5. Notify that layer $i$ is finished
   Go to the next layer ($i \leftarrow i + 1$)
Experimental Evaluation
SBS vs. HDBS: Mean Speedup against Single Thread

TSP with Time Windows (TSPTW)

Assembly Line Balancing (SALBP-1)

Used with CABS and measure the time to solve optimally (limits: 5-min and 188GB)
SBS vs. HDBS: Mean Speedup against Single Thread

Minimization of Open Stacks (MOSP)

Graph-Clear (building security problem)

Used with CABS and measure the time to solve optimally (limits: 5-min and 188GB)
### DIDP vs. Commercial Parallel Optimization Solvers

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Gurobi</th>
<th>CPO</th>
<th>DIDP (HDBS2)</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>262/13.3</td>
</tr>
<tr>
<td>CVRP (207)</td>
<td>vehicle routing</td>
<td>29/5.3</td>
<td>0/</td>
<td>-</td>
</tr>
<tr>
<td>SALBP-1 (2100)</td>
<td>line balancing</td>
<td>1351/1.3</td>
<td>1581/1.4</td>
<td>1826/18.8</td>
</tr>
<tr>
<td>Bin Packing (1615)</td>
<td>bin packing</td>
<td>1192/6.4</td>
<td>1251/9.2</td>
<td>1239/39.6</td>
</tr>
<tr>
<td>MOSP (570)</td>
<td>manufacturing</td>
<td>238/3.1</td>
<td>397/0.3</td>
<td>531/ 9.0</td>
</tr>
<tr>
<td>Graph-Clear (135)</td>
<td>building security</td>
<td>16/2.0</td>
<td>4/3.2</td>
<td>113/10.3</td>
</tr>
</tbody>
</table>

#optimally solved / mean speed up

- Resources: 32 threads, 5-min, and 188GB
- Gurobi: mixed-integer programming solver
- CPO: IBM ILOG CP Optimizer (constraint programming solver)
Conclusion

• A parallel beam search algorithm, HDBS2, shows good speedup and yields a high-performance parallel combinatorial optimization solver

• Start DIDP with Python: pip install didppy