Large Neighborhood Beam Search 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)

CP-like **model & solve paradigm** based on dynamic programming (DP)

**Combinatorial optimization problem**

**State-based DP model**

-Compute $V(\{1,2,3,4\},0)$

$$V(U,i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\},j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

$$V(U,i) \geq h(U,i)$$

**DIDP solver**

Current solvers use state space search
import didpy 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 {j}".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()
Example 1: a TSP-Like Routing Problem

Visit all nodes starting from customer 0 (no need to return) while minimizing the total travel cost (visiting $j$ from $i$ requires cost $c_{ij}$)

![Graph Diagram]

1 to 2 to 3 to 4
DP Model for the Example Problem

Recursive decomposition into subproblems by visiting one customer
DP Model for the Example Problem

Recursively defined **value function** $V$ maps a **state** (subproblem), defined by unvisited customers $U$ and the current customer $i$, to the cost

$$V(U, i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

$$V(U, i) \geq h(U, i)$$

**Target state** (original problem)

**Transition** (visiting one customer)

**Base case** (all customers are visited)

**Dual bound function**
DP Model for the Example Problem

Recursively defined value function $V$ maps a state (subproblem), defined by unvisited customers $U$ and the current customer $i$, to the cost.

$$V(U, i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

Target state (original problem)

Transition (visiting one customer)

Base case (all customers are visited)

Dual bound function

compute $V(\{1, 2, 3, 4\}, 0)$
DP Model for the Example Problem

Recursively defined value function \( V \) maps a state (subproblem), defined by unvisited customers \( U \) and the current customer \( i \), to the cost

\[
V(U, i) = \begin{cases} 
\min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\
0 & \text{if } U = \emptyset 
\end{cases}
\]

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

Target state (original problem)
Transition (visiting one customer)
Base case (all customers are visited)
Dual bound function
DP Model for the Example Problem

Recursively defined value function $V$ maps a state (subproblem), defined by unvisited customers $U$ and the current customer $i$, to the cost

$$V(U, i) = \begin{cases} 
\min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\
0 & \text{if } U = \emptyset 
\end{cases}$$

compute $V(\{1, 2, 3, 4\}, 0)$

Target state (original problem)

Transition (visiting one customer)

Base case (all customers are visited)

Dual bound function

$V(U, i) \geq h(U, i)$
DP Model for the Example Problem

Recursively defined **value function** \( V \) maps a **state** (subproblem), defined by unvisited customers \( U \) and the current customer \( i \), to the cost

\[
V(U, i) = \begin{cases} 
\min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\
0 & \text{if } U = \emptyset 
\end{cases}
\]

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

**Target state** (original problem)

**Transition** (visiting one customer)

**Base case** (all customers are visited)

**Dual bound function**
$V(S)$: the shortest path cost from $S$ to a base case in a state space

Solution: a path from the target state
DP as State Space Search

Different paths can lead to the same state => store states in memory
Heuristic Search for DIDP

Guide the search using $f(S) = g(S) + h(S)$ (LB on the path cost via $S$)

Target state: $\{1, 2, 3, 4\}, 0$

Actual cost to state $S$
- $g(S)$
  - $\{2, 3, 4\}, 1$
  - $\{1, 3, 4\}, 2$
  - $\{1, 2, 4\}, 3$
  - $\{1, 2, 3\}, 4$

Estimated cost from $S$
- $h(S)$
  - $\{3, 4\}, 2$
  - $\{2, 4\}, 3$
  - $\{2, 3\}, 4$

Base cases
- $\emptyset, 1$
- $\emptyset, 2$
- $\emptyset, 3$
- $\emptyset, 4$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state

$\{1, 2, 3, 4\}, 0 \ f: 10$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state

$\{1, 2, 3, 4\}, 0 \quad f: 10$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state

- Visit 1: $\{2, 3, 4\}, 1$ with $f: 11$
- Visit 2: $\{1, 3, 4\}, 2$ with $f: 15$
- Visit 3: $\{1, 2, 4\}, 3$ with $f: 16$
- Visit 4: $\{1, 2, 3\}, 4$ with $f: 12$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state: $\{1, 2, 3, 4\}, 0$

Visit 1: $\{2, 3, 4\}, 1, f: 11$

Visit 2: $\{1, 3, 4\}, 2, f: 15$

Visit 3: $\{1, 2, 4\}, 3, f: 16$

Visit 4: $\{1, 2, 3\}, 4, f: 12$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state

{1, 2, 3, 4}, 0

Visit 1

{2, 3, 4}, 1

{1, 3, 4}, 2

{1, 2, 4}, 3

{1, 2, 3}, 4

Visit 2

{3, 4}, 2 $f$: 16

{2, 4}, 3 $f$: 15

{2, 3}, 4 $f$: 15

{1, 3}, 2 $f$: 15

{2, 3}, 1 $f$: 13

{1, 2}, 3 $f$: 15

{1, 3}, 2 $f$: 15

{1, 2}, 3 $f$: 15

{2, 3}, 4 $f$: 14

{2, 3}, 4 $f$: 14
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state: $\{1, 2, 3, 4\}, 0$

Visit 1: $\{2, 3, 4\}, 1$
- $\{1, 3, 4\}, 2$
- $\{1, 2, 4\}, 3$
- $\{1, 2, 3\}, 4$

Visit 2:
- $\{3, 4\}, 2$: $f: 16$
- $\{2, 4\}, 3$: $f: 15$
- $\{2, 3\}, 4$: $f: 14$
- $\{2, 3\}, 1$: $f: 13$
- $\{1, 3\}, 2$: $f: 15$
- $\{1, 2\}, 3$: $f: 15$

- $\{1, 2, 3\}, 4$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state: $\{1, 2, 3, 4\}, 0$

Visit 1:
- $\{2, 3, 4\}, 1$
- $\{2, 3\}, 1$
- $\{1, 2\}, 3$

Visit 2:
- $\{1, 3, 4\}, 2$
- $\{2, 4\}, 3$
- $\{1, 3\}, 2$

Visit 3:
- $\{2, 3\}, 4$
- $\{2, 3\}, 4$

Visit 4:
- $\{1, 2\}, 3$
- $\{1, 2\}, 3$

Target state $\{2\}, 3$ with $f = 15$

Target state $\{3\}, 2$ with $f = 14$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state

Visit 1

$\{1, 2, 3, 4\}, 0$

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

$\{1, 2\}, 3$

$\{1\}, 3$

$\{2\}, 4$

$\{3\}, 2$

$\{2, 3\}, 1$

$\{2, 4\}, 3$

$\{2, 3\}, 4$

$\{3, 4\}, 2$

$\{2, 4\}, 3$

$\{2, 3\}, 4$

$\{1, 3\}, 2$

$\{3, 4\}, 2$

$\{2, 4\}, 3$

$\{2, 3\}, 4$

$\{1, 3\}, 2$

$\{3, 2\} f: 15$

$\{3\}, 2 f: 14$

$\{1, 2\}, 3$

$\{1\}, 3$

$\{2\}, 4$

$\{1, 2\}, 3$

$\{1\}, 3$

$\{2\}, 4$

$\{1, 2\}, 3$

$\{1\}, 3$

$\{2\}, 4$

$\{1, 2\}, 3$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state: $\{1, 2, 3, 4\}, 0$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

Target state

{1, 2, 3, 4}, 0

Visit 1

{2, 3, 4}, 1

3

{2, 4}, 3

{1, 3, 4}, 2

2

{2, 4}, 3

{1, 3}, 3

{2, 3}, 2

1

{1, 2, 4}, 3

4

{1, 2, 3}, 4

4

{1, 2, 3}, 4

4

{1, 2}, 3

3

{2, 3}, 1

2

{3, 2}, 3

3

{3, 4}, 2

2

{2, 2}, 2

∅, 2

∅, 3

∅, 3

f: 15

f: 14
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$
Beam Search

Keep the best $b$ states according to the $f$-value in each layer

Beam width: $b = 2$

The best solution: $(4, 1, 2, 3)$
SOTA: Complete Anytime Beam Search (CABS)

- Beam search with $b = 1, 2, 4, 8, \ldots$, until exhausting the state space
- Prune a state $S$ if $f(S) \geq$ the incumbent solution cost

$b = 4$, incumbent: 14

All states are searched or pruned, so the incumbent is optimal

Zhang 1998; Kuroiwa and Beck ICAPS 2023b
Large Neighborhood Beam Search
Large Neighborhood Search (LNS)

LNS for CP: remove value assignments to some variables from a solution and perform tree search to find a better solution

Current solution: $x = 1, y = 0, z = 0, w = 0$

Removed assignments: $z = 0, w = 0$

```
x = 1, y = 0, z = 0, w = 0
x = 1, y = 0, z = 1, w = 0
x = 1, y = 0, z = 0, w = 1
x = 1, y = 0, z = 1, w = 1
```
LNS for DIDP

Remove a partial path and search in a partial state space

Current: (1, 2, 3, 4)

Removed: (2, 3)

Better: (1, 3, 2, 4)
Large Neighborhood Beam Search (LNBS)

1. Find an initial solution
2. Select the length of the partial path $d$
3. Select the start of the partial path $i$
4. Select beam width $b$
5. Remove the partial path and do beam search with $b$
6. Check time limit
   - If time limit is reached, stop
   - Otherwise, repeat from step 2

---

Example:
- $i=2$
- $d=2$
- Removed: $(2, 3)$

Initial solution: $\{1, 2, 3, 4\}, 0$

Steps:
- $\{1, 2, 3, 4\}, 0 \rightarrow \{2, 3, 4\}, 1$
- $\{2, 3, 4\}, 1 \rightarrow \{3, 4\}, 2$
- $\{3, 4\}, 2 \rightarrow \{4\}, 3$
- $\{4\}, 3 \rightarrow \emptyset, 4$
Multi-Armed Bandit-Based Length Selection

How many transitions to remove given the remaining time 0.8?

<table>
<thead>
<tr>
<th>length $d = 2$ (arm)</th>
<th>length $d = 4$ (arm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average cost improvement: 0.5</td>
<td>Average cost improvement: 0.8</td>
</tr>
<tr>
<td>Average time: 0.01</td>
<td>Average time: 0.02</td>
</tr>
<tr>
<td># of times $d = 2$ used: 10</td>
<td># of times $d = 4$ used: 5</td>
</tr>
</tbody>
</table>

Random variables

Cost improvement (reward) $r_2$

Time $t_2$

Cost improvement (reward) $r_4$

Time $t_4$

=> Use Budgeted-UCB [Xia et al. 2015] to decide
Start Selection

Given length $d$, sample start $i$ uniformly at random

$d = 2$

\[
\begin{align*}
\{1, 2, 3, 4\}, 0 & \rightarrow \{2, 3, 4\}, 1 \\
\{2, 3, 4\}, 1 & \rightarrow \{3, 4\}, 2 \\
\{3, 4\}, 2 & \rightarrow \{4\}, 3 \\
\{4\}, 3 & \rightarrow \emptyset, 4
\end{align*}
\]
Beam Width Selection

Double beam width $b_{di}$ for length $d$ and start $i$ after each beam search starting from $b_{di}=1$
Beam Width Selection

- Reset $b_{di}$ to 1 if the partial state space changes
- Prove optimality if $i=1$, $d$ is the solution length, and $b$ is large enough

Examples:

$d=2$, $i=1$, $b=4$

$d=2$, $i=2$, $b=1$

$d=2$, $i=3$, $b=2$

$d=4$, $i=1$, $b=1$

The partial state space changes

$d=2$, $i=1$, $b=1$

$d=2$, $i=2$, $b=1$

$d=2$, $i=3$, $b=1$

$d=4$, $i=1$, $b=2$

The state space doesn't change

$d=2$, $i=1$, $b=1$

$d=2$, $i=2$, $b=1$

$d=2$, $i=3$, $b=1$

$d=4$, $i=1$, $b=2$
Experimental Evaluation
Distribution of Primal Gap in Routing Problems

Primal gap: relative gap to the best known solution (smaller is better) achieved with 30 min

TSP with Time Windows (TSPTW)

Pick-and-Delivery TSP (m-PDTSP)
Distribution of Primal Gap in Scheduling Problems

Primal gap: relative gap to the best known solution (smaller is better) achieved with 30 min

Single Machine Total Weighted Tardiness

Talent Scheduling
Distribution of Primal Gap in Other Problems

Primal gap: relative gap to the best known solution (smaller is better) achieved with 30 min

Simple Assembly Line Balancing (SALBP-1)

Minimization of Open Stacks (MOSP)
Why LNBS is Worse in Some Problems?

Diverse => easy to find a better one?

Not diverse => difficult?

Diverse

Not diverse

# of partial paths

Partial path cost

Better  Current  Worse

Better  Current  Worse

# of partial paths

Partial path cost
Why LNBS is Worse in Some Problems?

- Hypothesis: when partial path costs are not diverse (low entropy), finding a better solution in a partial state space is difficult.
- Not much difference if a problem is easy (the solution length is small).

Routing and scheduling problems:

- Solution length vs. Cost entropy

Other problems:

- Solution length vs. Cost entropy
Conclusion

- DIDP: a model & solve paradigm based on DP

- LNBS is effective particularly in routing and scheduling problems such as TSPTW, m-PDTSP, and talent scheduling, which seems to be related to the diversity of partial path costs

- Start DIDP with Python: `pip install didppy`
  Tutorials and API Reference: [https://didppy.rtfd.io](https://didppy.rtfd.io)
Beam Width Selection

Double beam width $b_{di}$ for length $d$ and start $i$ after each beam search starting from $b_{di} = 1$.

$d=2, i=1, b=2$

$d=2, i=2, b=1$

$d=2, i=3, b=2$

$d=4, i=1, b=1$

$d=2, i=1, b=4$

$d=2, i=2, b=1$

$d=2, i=3, b=2$

$d=4, i=1, b=1$
Definition of Entropy

\[ H(Y) = \sum_{c \in C} \frac{|\{ y \in Y \mid \text{cost}_y = c \}|}{|Y|} \log_2 \frac{|\{ y \in Y \mid \text{cost}_y = c \}|}{|Y|} \]

\( Y \) : set of partial paths
\( C \) : set of partial path costs

We enumerate all feasible prefixes for an initial solution where first eight transitions are removed