# Parallel Beam Search Algorithms for Domain-Independent Dynamic Programming

**Ryo Kuroiwa, J. Christopher Beck**

Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada, ON M5S 3G8
ryo.kuroiwa@mail.utoronto.ca, jcb@mie.utoronto.ca

**Abstract**

Domain-independent dynamic programming (DIDP), a model-based paradigm based on dynamic programming, has shown promising performance on multiple combinatorial optimization problems compared with mixed integer programming (MIP) and constraint programming (CP). The current DIDP solvers are based on heuristic search, and the state-of-the-art solver, complete anytime beam search (CABS), uses beam search. However, the current DIDP solvers cannot utilize multiple threads, unlike state-of-the-art MIP and CP solvers. In this paper, we propose three parallel beam search algorithms and develop multi-thread implementations of CABS. With 32 threads, our multi-thread DIDP solvers achieve 9 to 39 times speedup on average and significant performance improvement over the sequential solver, finding the new best solutions for two instances of the traveling salesperson problem with time windows. In addition, our solvers outperform multi-thread MIP and CP solvers in four of the six combinatorial optimization problems evaluated.

## Introduction

Domain-independent dynamic programming (DIDP) has been proposed as a model-based paradigm for combinatorial optimization (Kuroiwa and Beck 2023b). In DIDP, a problem is formulated as a declarative dynamic programming model, and the model is solved by a general-purpose solver. Previous work has shown that DIDP solvers using heuristic search outperform existing model-based paradigms, mixed-integer programming (MIP) and constraint programming (CP), in multiple problems (Kuroiwa and Beck 2023b,c). The state-of-the-art DIDP solver uses complete anytime beam search (CABS) (Zhang 1998), which is based on beam search. However, the current DIDP solvers cannot utilize multiple threads, unlike state-of-the-art MIP and CP solvers.

Parallel heuristic search has been studied in previous work. For best-first search algorithms, multi-thread and distributed parallel algorithms have been proposed (Burns et al. 2010; Kishimoto, Fukunaga, and Botea 2013; Jinnai and Fukunaga 2017; Kuroiwa and Fukunaga 2019, 2020). They can be used to parallelize CAASDy (Kuroiwa and Beck 2023b), a DIDP solver based on best-first search. However, CAASDy is not an anytime solver and is outperformed by CABS in practice (Kuroiwa and Beck 2023c). Beam search has been parallelized for specific combinatorial optimization problems (Frohner et al. 2023), but DIDP is more general.

We propose three parallel beam search algorithms and develop corresponding multi-thread DIDP solvers. The experimental results show that our solvers achieve significant speedup and performance improvement compared to CABS and find the new best solutions for two instances of the traveling salesperson problem with time windows. In addition, our solvers outperform multi-thread MIP and CP solvers in four of six combinatorial optimization problems tested.

## Domain-Independent Dynamic Programming

Domain-independent dynamic programming (DIDP) is a paradigm for combinatorial optimization problems (Kuroiwa and Beck 2023b) in which a problem is described by a formalism called dynamic programming description language (DyPDL), based on a state transition system. In this paper, we focus on a subset of DyPDL that has been used in previous work (Kuroiwa and Beck 2023b,c).

A DyPDL model is a tuple $(\mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}, h)$. A *state* is defined by values of *state variables* $\mathcal{V}$. Each state variable has a type: *set*, *element*, or *numeric*. A set variable takes a subset of a given set, an element variable takes an element of a given set, and a numeric variable takes a number (integer in this paper). We denote the value of a state variable $v$ in state $S$ by $S[v]$. A *transition* $\tau \in \mathcal{T}$ is *applicable* in state $S$ if $S$ satisfies the *preconditions* of $\tau$. The set of applicable transitions in $S$ is denoted by $\mathcal{T}(S)$. When $\tau$ is applied, $S$ transitions to another state $S[\![\tau]\!]$ according to the *effects* of $\tau$ with the cost of $w_\tau(S) \geq 0$, where $w_\tau$ is a function called the *cost expression*. Preconditions, effects, and cost expressions are described by *expressions*, mathematical operations on state variables. For example, a transition $\tau$ may have an effect $U \leftarrow U \setminus \{j\}$ on a set variable $U$ and effect $i \leftarrow j$ on an element variable $i$ with precondition $j \in U$ and cost expression $c_{ij}$. A *solution* is a sequence of transitions that makes the *target state* $S^0$ transition to a *base state* $S^*$, which satisfies one of the *base cases* $\mathcal{B}$, denoted by $\exists B \in \mathcal{B}, S^* \models B$. In addition, each state $S$ resulting from applying a transition in a solution must satisfy the *state constraints* $\mathcal{C}$, denoted by $S \models \mathcal{C}$. An $S$-*solution* is a sequence of transitions satisfying the above conditions starting from $S$. Base cases and state constraints are also described by expressions. The cost of

Algorithm 1: Beam search.

```
Input: DyPDL model $(\mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}, h)$ and primal bound $\overline{f}$.
Parameters: beam width $b$.
Output: solution $x$ having lower cost than $\overline{f}$ and its optimality.
1:  $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), x(S^0) \leftarrow \langle\rangle$
2:  $O \leftarrow \{S^0\}, \overline{x} \leftarrow$ NULL, complete $\leftarrow \top$, $\underline{f} \leftarrow f(S^0)$
3:  while $O \neq \emptyset$ and $\overline{x} =$ NULL do
4:      $G \leftarrow \emptyset$
5:      for all $S \in O$ do
6:          if $\exists B \in \mathcal{B}, S \models B$ then                ▷ A solution.
7:              if $g(S) < \overline{f}$ then $\overline{x} \leftarrow x(S), \overline{f} \leftarrow g(S)$
8:              if $g(S) = \underline{f}$ then return $x(S)$, $\top$      ▷ Optimal.
9:              continue
10:         for all $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ do      ▷ Expand.
11:             $g^\tau \leftarrow g(S) \times w_\tau(S), f^\tau \leftarrow g^\tau \times h(S[\![\tau]\!])$
12:             if $f^\tau \geq \overline{f}$ then continue
13:             INSERT$(S, \tau, g^\tau, f^\tau, G)$
14:     $O \leftarrow \{S \in G \mid f(S) < \overline{f}\}$
15:     if complete and $O \neq \emptyset$ then $\underline{f} \leftarrow \max\{\underline{f}, \min_{S \in O} f(S)\}$
16:     if $|O| > b$ then
17:         $O \leftarrow$ the best $b$ states in $O$, complete $\leftarrow \bot$
18: return $\overline{x}$, complete $\wedge\ O = \emptyset$
```

a solution $x = \langle \tau_1, ..., \tau_n \rangle$ is $w_{\tau_1}(S^0) \times ... \times w_{\tau_n}(S^{n-1})$, where $S^i = S^{i-1}[\![\tau_i]\!]$ for $i = 1, ..., n$ and $\times$ is a binary operator, either of $+$ or $\max$, specified by the model. An *optimal solution* minimizes the cost.

DyPDL allows a user to model implied information. Element and numeric variables can be defined as *resource variables*, with a preference of less or more. A state $S$ *dominates* another state $S'$, denoted by $S' \preceq S$, if $S[v] \leq S'[v]$ ($S[v] \geq S'[v]$) for all resource variables $v$ where less (more) is preferred, and $S[v] = S'[v]$ for all non-resource variables $v$. If $S' \preceq S$, it is guaranteed that an optimal $S$-solution has a lower or equal cost than an optimal $S'$-solution. The *dual bound function* $h$ returns $h(S)$, a lower bound on the $S$-solution cost, given a state $S$.

## Beam Search for DIDP

A DyPDL model can be solved by heuristic search in a *state transition graph*, where nodes are states, the weight of an edge $(S, S[\![\tau]\!])$, corresponding to transition $\tau$, is $w_\tau(S)$, and an $S$-solution corresponds to a path from $S$ to a base state (Kuroiwa and Beck 2023b). The path cost is computed by taking the sum or maximum of the edge weights. Taking the maximum is different from usual settings for heuristic search but can be handled by generalizing it (Edelkamp, Jabbar, and Lafuente 2005). We assume an acyclic state transition graph.

The current state-of-the-art solver (Kuroiwa and Beck 2023c) uses complete anytime beam search (CABS) (Zhang 1998), which is based on beam search (see Algorithm 1). As part of its input, beam search takes the, possibly infinite, primal bound, $\overline{f}$, an upper bound on the optimal solution cost. For each state $S$, it maintains $x(S)$, the best path found so far from the target state $S^0$. The $g$-value, $g(S)$, is the cost of $x(S)$. The $h$-value, $h(S)$, is a lower bound on the optimal path cost from $S$ to a base state. The $f$-value,

Algorithm 2: Insert a successor to a set if it is not dominated.

```
Input: state $S$, transition $\tau$, $g$-value $g^\tau$, $f$-value $f^\tau$, and set $G$
1: function INSERT$(S, \tau, g^\tau, f^\tau, G)$
2:     if $\nexists S' \in G$ with $S[\![\tau]\!] \preceq S'$ and $g^\tau \geq g(S')$ then
3:         if $\exists S' \in G$ with $S' \preceq S[\![\tau]\!]$ and $g^\tau \leq g(S')$ then
4:             $G \leftarrow G \setminus \{S'\}$          ▷ Remove a dominated state.
5:         $g(S[\![\tau]\!]) \leftarrow g^\tau, f(S[\![\tau]\!]) \leftarrow f^\tau$     ▷ Update $g$ and $f$.
6:         $x(S[\![\tau]\!]) \leftarrow \langle x(S); \tau \rangle$              ▷ Update the path.
7:         $G \leftarrow G \cup \{S[\![\tau]\!]\}$          ▷ Insert the state into the set.
```

$f(S) = g(S) \times h(S)$ is the lower bound on the cost of a path to a base state extending $x(S)$. Beam search also maintains the *global dual bound* $\underline{f}$, the lower bound on the solution cost, initialized with $\underline{f} = f(S^0)$ (line 2).

At each iteration, beam search processes all states in the *open list* $O$, which is initilized with $S^0$ in line 2. For each state $S$ in $O$, if $S$ is a base state and the solution cost is better than the primal bound $\overline{f}$, the solution $\overline{x}$ and $\overline{f}$ are updated (line 7). If the solution cost matches the global dual bound, the solution is optimal and beam search terminates (line 8). Otherwise, beam search *expands* $S$, i.e., it applies transitions to $S$ and inserts the resulting *successor states* into the open list (lines 10-13). It only considers applicable transitions and successor states satisfying the state constraints (line 10). The $g$-value and $f$-value of each successor state are computed and saved to $g^\tau$ and $f^\tau$ (line 11). If $f^\tau$ is greater than or equal to $\overline{f}$, the state is pruned because it does not lead to a better solution (line 12). Otherwise, it is inserted into the set of states $G$ if it is not dominated (line 13).

The function INSERT in Algorithm 2 checks if any state $S'$ in $G$ dominates the successor state $S[\![\tau]\!]$ and has a lower or equal $g$-value (line 2). In such a case, a path cost to $S'$ is lower or equal, and $S'$-solution has a better or equal cost by definition of dominance. Thus, the current successor state can be ignored without loss of optimality. Conversely, if the current successor state and its $g$-value dominate an existing state, the dominated state is removed from $G$ (lines 3 and 4).

After expanding all states, $O$ is updated to $G$ ignoring states with $f$-values greater than or equal to the primal bound (line 14). The open list $O$ contains only states in the same *layer*, i.e., states reached from the target state with the same number of transitions. If $|O| > b$, the best $b$ states are kept, and the flag 'complete' is set to $\bot$ (lines 17). Here, $b$ is called a *beam width*. When a solution is found or $O$ becomes empty, beam search terminates (line 18). If complete $= \top$, i.e., no state is discarded, and $O$ is empty, we have exhausted all solutions having lower costs than $\overline{f}$, which is indicated by the second return value. CABS repeats beam search while doubling $b$ and updating $\overline{f}$ to the best solution cost until the second return value becomes $\top$. At such a point, the best found solution is optimal, or the model is infeasible if no solution is found. Our version of CABS follows Kuroiwa and Beck (2023c), but the original one by Zhang (1998) is more general; beam search selects states in $O$ using some criterion, and CABS relaxes such a criterion at each iteration.

If complete $= \top$ and the primal bound is greater than the optimal cost, there exists a state $S \in O$ such that an optimal solution is an extension of $x(S)$; $\{x(S) \mid S \in O\}$ is the set of all non-dominated paths from the target state to a certain layer. For paths removed due to dominance in Algorithm 2, it is guaranteed that a better or equal path exists in the set. For paths removed due to the primal bound, their costs are greater than or equal to the optimal cost. Therefore, we update the global dual bound in line 15 by taking the minimum $f$-value of states in $O$.

**Implementation of Beam Search** In our implementation, a data structure called a *search node* has a pointer to a state and its $g$-, $h$-, and $f$-values. The set $G$ is represented by a hash table where the key is the values of non-resource variables and the hash-table entry is a list of search nodes. In Algorithm 2, a new search node is compared with each node in the list. If the new node dominates an existing one in the list, it replaces the dominated node. Otherwise, the new node is appended to the list if it is not dominated.

To save memory and computation, selecting the best $b$ states in $O$ in line 17 of Algorithm 1 is performed incrementally. Search nodes are stored in a binary heap in descending order of $f$-values, and ties are broken by $h$-values. Before checking dominance in Algorithm 2, if the number of states in $G$ is equal to $b$, and $(f^\tau, h(S[\![\tau]\!]))$ is greater than or equal to that of the top of the binary heap, $S[\![\tau]\!]$ is ignored. A search node also has a binary flag indicating if the state is included in $G$. If a state $S'$ in $G$ is dominated by the successor state in line 3 of Algorithm 2, the flag of $S'$ is set to be false, and the number of states in $G$ is decremented. The dominated search node is removed as soon as it becomes the top of the binary heap. When inserting the successor state, if $G$ still contains $b$ states, the top of the binary heap is removed.

## Parallel Beam Search for DIDP

We propose three parallel beam search algorithms, all of which expand states from the open list in parallel.

### Shared Beam Search (SBS)

The main issue of expanding states in parallel is how to check the dominance of successor states; a successor state generated in one thread might be dominated by one generated in another thread, so communication across multiple threads is necessary. As we have described, sequential beam search performs the dominance check using a hash table. Our first algorithm, shared beam search (SBS), uses a *concurrent hash table*: a hash table designed for multi-thread access. Otherwise, SBS is similar to an existing problem-specific beam search method (Frohner et al. 2023). While there are multiple implementations of concurrent hash tables, we use DashMap 5.4.0,[1] in which a hash table is divided into multiple *shards*, and each shard is protected by a lock. When a thread accesses a shard, it locks the shard, performs the operation, and unlocks the shard. A shard is uniquely determined by the hash value of a key, so the same key is always stored in the same shard.

We show pseudo-code of SBS in Algorithm 3. The concurrent hash table is divided into multiple shards

[1]https://crates.io/crates/dashmap/5.4.0

Algorithm 3: Shared beam search (SBS).

```
Input: DyPDL model $(\mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}, h)$, primal bound $\overline{f}$.
Parameters: beam width $b$ and shard amount $m$.
Output: solution $x$ having lower cost than $\overline{f}$ and its optimality.
1: $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), x(S^0) \leftarrow \langle\rangle$
2: $O \leftarrow \{S^0\}, \overline{x} \leftarrow$ NULL, complete $\leftarrow \top, \underline{f} \leftarrow f(S^0)$
3: while $O \neq \emptyset$ and $\overline{x} =$ NULL do
4:     for all $S \in O$ with $\exists B \in \mathcal{B}, S \models B$ in parallel do
5:         if $g(S) < \overline{f}$ then $\overline{x} \leftarrow x(S), \overline{f} \leftarrow g(S)$
6:     if $\overline{x} \neq$ NULL then
7:         if $\overline{f} = \underline{f}$ then return $\overline{x}, \top$.
8:         if not complete then return $\overline{x}, \bot$.
9:     $G_i \leftarrow \emptyset$ for $i = 1, ..., m$
10:    for all $S \in O$ with $\nexists B \in \mathcal{B}, S \models B$ in parallel do
11:        for all $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ do
12:            $g^\tau \leftarrow g(S) \times w_\tau(S), f^\tau \leftarrow g^\tau \times h(S[\![\tau]\!])$.
13:            if $f^\tau \geq \overline{f}$ then continue.
14:            determine shard $j$ for $S[\![\tau]\!]$.
15:            lock $G_j$
16:            INSERT$(S, \tau, g^\tau, f^\tau, G_j)$
17:            unlock $G_j$
18:    $O \leftarrow \{S \in \bigcup_{i=1}^m G_i \mid f(S) < \overline{f}\}$
19:    if complete and $O \neq \emptyset$ then $\underline{f} \leftarrow \max\{\underline{f}, \min_{S \in O} f(S)\}$
20:    if $|O| > b$ then
21:        $O \leftarrow$ the best $b$ states in $O$, complete $\leftarrow \bot$
22: return $\overline{x}$, complete $\wedge\ O = \emptyset$
```

$G_1, ..., G_m$. SBS checks base states in parallel (lines 4-8) before expanding states. While the sequential implementation incrementally selects the best $b$ states in line 17 of Algorithm 1, parallelizing this step is not straightforward. In the implementation of SBS, for the open list $O$, we use an array of search nodes instead of a binary heap. In line 21, search nodes are sorted in parallel by the ascending order of $f$-values and $h$-values, and the best $b$ are selected. This difference from the sequential implementation may change the search behavior as a tie may be broken differently. If the flag of a search node indicates that the state is not included in $G$, that search node is ignored before sorting. In the implementation, the global dual bound computation in line 19 is performed only when the sorting is performed to avoid the overhead of computing the minimum $f$-value of states in $O$. The parallel operations in lines 4-5, 10-19, and 21 are performed by a thread pool using Rayon 1.7.0.[2] With $k$ threads, we use $m = 4k$ shards, following the default of DashMap.

### Hash Distributed Beam Search (HDBS)

Another approach to parallel beam search is to use message passing as in previous work on parallel best-first search (BFS) (Kishimoto, Fukunaga, and Botea 2013; Jinnai and Fukunaga 2017; Kuroiwa and Fukunaga 2019). In this approach, a state is uniquely assigned to a worker[3] by a hash function, and each worker independently maintains an open list and a hash table to check dominance. When a worker

[2]https://crates.io/crates/rayon/1.7.0

[3]We use 'worker' instead of 'thread' or 'process' as message passing can be used with multi-thread and multi-process settings.

Algorithm 4: Hash distributed beam search 1 (HDBS1).

```
Input: DyPDL model $(\mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}, h)$ and primal bound $\overline{f}$.
Parameters: beam width $b$ and number of workers $k$.
Output: solution $x$ having lower cost than $\overline{f}$ and its optimality.
 1: $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), x(S^0) \leftarrow \langle\rangle$
 2: $\underline{f} \leftarrow f(S^0), Q_i \leftarrow []$ for $i = 1, ..., k$
 3: for $i = 1, ..., k$ in parallel do
 4:     $O_i \leftarrow \emptyset, \overline{x}_i \leftarrow$ NULL, complete$_i \leftarrow \top, \overline{f}_i \leftarrow \overline{f}$
 5:     if $i =$ HASH$(S^0)$ then $O_i \leftarrow O_i \cup \{S^0\}$
 6:     loop
 7:         $G_i \leftarrow \emptyset$, $c_i \leftarrow 0$, sent_all$_i \leftarrow \bot$
 8:         while $c_i < k$ do
 9:             RECV_STATE$(Q_i, G_i, c_i)$
10:             if $\exists S \in O_i$ then
11:                 $O_i \leftarrow O_i \setminus \{S\}$
12:                 if $\exists B \in \mathcal{B}, S \models B$ then
13:                     if $g(S) < \overline{f}_i$ then $\overline{x}_i \leftarrow x(S), \overline{f}_i \leftarrow g(S)$
14:                     continue
15:                 for all $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ do
16:                     $g^\tau \leftarrow g(S) \times w_\tau(S), f^\tau \leftarrow g^\tau \times h(S[\![\tau]\!])$
17:                     if $f^\tau \geq \overline{f}_i$ then continue
18:                     $j \leftarrow$ HASH$(S[\![\tau]\!])$
19:                     PUSH$(Q_j, (S, \tau, g^\tau, f^\tau))$
20:             else if not sent_all then
21:                 PUSH$(Q_j$, NULL) for $j = 1, ..., k$
22:                 sent_all $\leftarrow \top$
23:         $O_i \leftarrow \{S \in G_i \mid f(S) < \overline{f}_i\}$, $\underline{f}_i \leftarrow \min_{S \in O_i} f(S)$
24:         send $\overline{x}_i$, $\overline{f}_i$ $\underline{f}_i$, $|O_i|$, and complete$_i$ to worker 1
25:         if $i = 1$ then AGGREGATE$(\underline{f})$
26:         receive $\underline{f}$, $l$, and is_optimal from worker 1
27:         if $i = l$ then
28:             if $\overline{x}_i \neq$ NULL and $\overline{f}_i = \underline{f}$ then return $\overline{x}_i$, $\top$
29:             return $\overline{x}_i$, is_optimal.
30:         else if $l \neq$ NULL then
31:             break
32:         if $|O_i| > b/k$ then
33:             $O_i \leftarrow$ the best $b/k$ states in $O_i$, complete$_i \leftarrow \bot$
```

generates a search node, it is sent to the worker responsible for the corresponding state. We adapt this idea to beam search and propose hash distributed beam search (HDBS).

Unlike BFS, beam search explores layer by layer: the set of generated states $G$ contains only states in the same layer. In HDBS, when a worker inserts a state received from another worker into $G$, the state must belong to the same layer as the other states in $G$. In other words, we need to synchronize the layer searched by each worker. We propose two synchronization variants, HDBS1 and HDBS2.

**HDBS1** HDBS1 takes a straightforward approach: each worker proceeds to the next layer when all workers finish the current layer (Algorithm 4). Each worker $i$ maintains an open list $O_i$ and set $G_i$. A state is assigned a worker by a hash function HASH based on the values of non-resource variables. *Channels* used for message passing are represented by $Q_i$. When a worker generates a search node, it is sent to the worker responsible for the corresponding state using PUSH (line 19). This operation is non-blocking, i.e.,

Algorithm 5: Receive states from a channel.

```
Input: channel $Q$, set $G$, and counter $c$
1: function RECV_STATE$(Q, G, c)$
2:     while $Q$ contains a message do
3:         $(S, \tau, g^\tau, f^\tau) \leftarrow$ POP$(Q)$
4:         if $(S, \tau, g^\tau, f^\tau) =$ NULL then
5:             $c \leftarrow c + 1$
6:         else
7:             INSERT$(S, \tau, g^\tau, f^\tau, G)$
```

Algorithm 6: Aggregate information of the current layer.

```
Input: current dual bound $\underline{f}$.
 1: function AGGREGATE$(\underline{f})$
 2:     receive $\overline{x}_j$, $\overline{f}_j$, $\underline{f}_j$, $|O_j|$, and complete$_j$ from $j = 1, ..., k$
 3:     empty $\leftarrow \sum_{j=1}^k |O_j| = 0$, complete $\leftarrow \bigwedge_{j=1}^k$ complete$_j$
 4:     if $\forall j = 1, ..., k, \overline{x}_j =$ NULL then                ▷ No solution.
 5:         if empty then                                  ▷ No states to search.
 6:             $l \leftarrow 1$
 7:         else
 8:             $l \leftarrow$ NULL
 9:     else
10:         let $l \in \arg\min_{j=1,...,k} \overline{f}_j$
11:     if complete and not empty then
12:         $\underline{f} \leftarrow \max\{\underline{f}, \min\{\min_{j=1,...,k} \overline{f}_j, \min_{j=1,...,k:|O_j|>0} \underline{f}_j\}\}$
13:     is_optimal $\leftarrow$ complete $\wedge$ empty
14:     broadcast $\underline{f}$, $l$, is_optimal
```

the worker does not wait until the message is received. Messages are received in the order of sending. A worker receives a state using POP, which is also non-blocking, and inserts it into $G$ after a dominance check (Algorithm 5). HDBS1 alternately receives states and expands a state (lines 9 and 10).

When a worker has expanded all states in the open list, it notifies other workers that it has sent all states by sending a special message, NULL (line 21). Here, the flag 'sent_all' is used to send the special message only once. Using $c_i$, worker $i$ maintains the number of workers from which it has received all successor states, which is incremented in line 5 of Algorithm 5. When $c_i$ becomes the number of workers, $k$, worker $i$ has finished the current layer. The worker then sends the best solution found ($\overline{x}_i$), its cost ($\overline{f}_i$), the minimum $f$-value in the open list ($\underline{f}_i$), the number of states in the open list ($|O_i|$), and if states are discarded (complete$_i$) to worker 1 (line 24). Worker 1 aggregates this information and broadcasts the global dual bound ($\underline{f}$), the worker index which has found the best solution ($l$), and the solution optimality (is_optimal) in Algorithm 6. When the open lists of all workers are empty with no solution found, HDBS1 sets $l$ to 1. The global dual bound is computed from $\underline{f}_j$ only when no state is discarded up to this point in each worker $j$ (line 11 of Algorithm 6). In line 12, it is possible that $\min_{j=1,..,k:|O_j|>0} \underline{f}_j > \overline{f} = \min_{j=1,..,k} \overline{f}_j$. In such a case, all states in the next layer can be pruned, and the current solution is optimal, so the global dual bound becomes $\overline{f}$.

If $l \neq$ NULL, a solution is found or all open lists are

Algorithm 7: Hash distributed beam search 2 (HDBS2).

**Input**: DyPDL model $(\mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}, h)$ and primal bound $\overline{f}$.
**Parameters**: beam width $b$ and number of workers $k$.
**Output**: solution $x$ having lower cost than $\overline{f}$ and if $x$ is optimal.

```
1:  $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), x(S^0) \leftarrow \langle\rangle$
2:  $Q_i, R_i \leftarrow []$ for $i = 1, ..., k$
3:  for $i = 1, ..., k$ in parallel do
4:      $O_i \leftarrow \emptyset, \overline{x}_i \leftarrow$ NULL, complete$_i \leftarrow \top$
5:      $\overline{f}_i \leftarrow \overline{f}, \underline{f}_i \leftarrow \infty$
6:      if $i =$ HASH$(S^0)$ then $O_i \leftarrow O_i \cup \{S^0\}, \underline{f}_i \leftarrow f(S^0)$
7:      for $j = 1, ..., k$ do
8:          PUSH$(R_j, ($NULL$, \overline{f}_i, \underline{f}_i, |O_i|,$ complete$_i))$
9:      loop
10:         $G_i \leftarrow \emptyset, c_i \leftarrow 0,$ sent_all$_i \leftarrow \bot$
11:         $P_{ij} \leftarrow []$ for $j = 1, ..., k$
12:         $L_i \leftarrow \emptyset, l_i \leftarrow$ NULL$, \hat{f}_i, f'_i \leftarrow \infty,$ empty$_i \leftarrow O_i = \emptyset$
13:         while $c_i < k$ do
14:             RECV_INFO$(R_i, L_i, l_i, \hat{f}_i, f'_i,$ empty$_i,$ complete$_i)$
15:             $\overline{f}_i = \min\{\overline{f}_i, \hat{f}_i\}$
16:             if $|L_i| = k$, complete$_i$, and not empty$_i$ then
17:                 $\underline{f}_i = \max\{\underline{f}_i, \min\{\hat{f}_i, f'_i\}\}$
18:             for all $j \in L_i$ such that $P_{ij}$ contains a message do
19:                 APPEND$(Q_j, P_{ij}), P_{ij} \leftarrow []$
20:             RECV_STATE$(Q_i, G_i, c_i)$
21:             if $\exists S \in O_i$ then
22:                 $O_i \leftarrow O_i \setminus \{S\}$
23:                 if $\exists B \in \mathcal{B}, S \models B$ then
24:                     if $g(S) < \overline{f}_i$ then $\overline{x}_i \leftarrow x(S), \overline{f}_i \leftarrow g(S)$
25:                     continue
26:                 for all $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ do
27:                     $g^\tau \leftarrow g(S) \times w_\tau(S), f^\tau \leftarrow g^\tau \times h(S[\![\tau]\!])$
28:                     if $f^\tau \geq \overline{f}_i$ then continue
29:                     $j \leftarrow$ HASH$(S[\![\tau]\!])$
30:                     if $j \in L_i$ then
31:                         PUSH$(Q_j, (S, \tau, g^\tau, f^\tau))$
32:                     else
33:                         PUSH$(P_{ij}, (S, \tau, g^\tau, f^\tau))$
34:             else if not sent_all and $|L_i| = k$ then
35:                 PUSH$(Q_j,$ NULL$)$ for $j = 1, ..., k$
36:                 sent_all $\leftarrow \top$
37:         if $i = l_i \vee ($empty$_i \wedge i = 1)$ then
38:             if $\overline{x}_i \neq$ NULL and $\overline{f}_i = \underline{f}_i$ then return $\overline{x}_i, \top$
39:             return $\overline{x}_i,$ empty$_i \wedge$ complete$_i$
40:         else if $l_i \neq$ NULL $\vee$ empty$_i$ then
41:             break
42:         $O_i \leftarrow \{S \in G_i \mid f(S) < \overline{f}_i\}, \underline{f}_i \leftarrow \min_{S \in O_i} f(S)$
43:         for $j = 1, ..., k$ do
44:             PUSH$(R_j, (\overline{x}_i, \overline{f}_i, \underline{f}_i, |O_i|,$ complete$_i))$
45:         if $|O_i| > b/k$ then
46:             $O_i \leftarrow$ the best $b/k$ states in $O_i$, complete$_i \leftarrow \bot$
```

empty, so all workers terminate, and worker $l$ returns $\overline{x}_l$ (lines 27-31). Otherwise, each worker knows that all workers finished the current layer, so it proceeds to the next layer. In line 33, each worker keeps the best $b/k$ states. As a result, the set of kept states is not necessarily the same as the best $b$ states across all open lists. For example, if states with high $f$-values are concentrated in a single worker, states that would have been discarded by sequential beam search will be kept. HDBS1 can behave differently from sequential beam search.

Algorithm 8: Receive the information of the previous layer.

**Input**: channel $R$, set $L$, index $l$, bounds $\hat{f}$ and $f'$, and two flags

```
1: function RECV_INFO$(R, L, l, \hat{f}, f',$ empty, complete$)$
2:     while $R$ contains a message do
3:         $(\overline{x}_j, \overline{f}_j, \underline{f}_j, |O_j|,$ complete$_j) \leftarrow$ POP$(R)$
4:         $L \leftarrow L \cup \{j\}$
5:         if $\overline{x}_j \neq$ NULL $\wedge (\overline{f}_j < \hat{f} \vee (\overline{f}_j = \hat{f} \wedge j < l))$ then
6:             $\hat{f} \leftarrow \overline{f}_j, l \leftarrow j$
7:         empty $\leftarrow$ empty $\wedge |O_j| = 0$
8:         complete $\leftarrow$ complete $\wedge$ complete$_j$
9:         if compelete and $|O_j| > 0$ then $f' = \min\{f', \underline{f}_j\}$
```

**HDBS2** In HDBS1, each worker needs to wait until all workers finish the current layer. However, a worker can actually expand states in the next layer as long as it does not send successor states to unfinished workers. Based on this idea, we propose HDBS2. We show pseudo-code of HDBS2 in Algorithm 7. Instead of aggregating all information to worker 1, each worker sends the information to all workers using channels $R_j$ in line 44 and immediately proceeds to the next layer. When worker $i$ receives this information from worker $j$ (line 14), worker $i$ knows that worker $j$ has finished the previous layer. When worker $i$ generates a search node assigned to $j$, if $j$ finished the previous layer, it is immediately pushed to the channel $Q_j$ (line 31). Otherwise, it is stored in a local buffer $P_{ij}$ (line 33), and all search nodes in $P_{ij}$ are pushed to $Q_j$ once $i$ receives the information from $j$ (line 19). Here, $L_i$ maintains the set of workers from which worker $i$ has received the information of the previous layer.

The information of the previous layer is aggregated by each worker in Algorithm 8. It keeps updating the worker index which has found the best solution in the previous layer ($l$), the best solution cost in the previous layer ($\hat{f}$), the minimum $f$-value in the current layer ($f'$), the flag indicating if the current layer is empty (empty), and the flag indicating if any worker discarded a state (complete). For $\hat{f}$, when multiple workers find the best solution, the one with the smallest index is selected (line 5). In line 24 of Algorithm 7, $\hat{f}$ is not updated by the solution found in the current layer. Thus, $\hat{f}$ and $l$ only depend on the information shared with all workers, and $l$ is uniquely determined. Worker $l$ may return a solution having a better cost than $\hat{f}_l$ in line 39, since $\overline{x}_l$ can be updated. However, even if worker $j \neq l$ finds a better solution than $\hat{f}_j = \hat{f}_l$, worker $j$ does not return a solution.

In line 34, before sending the special message NULL, each worker checks if all workers have finished the previous layer. This condition ensures the following property: when worker $i$ receives the special message NULL from worker $j$ in channel $Q_i$, worker $i$ knows that worker $j$ already received the information of the previous layer from all workers. Therefore, when worker $i$ sends the information of the current layer in line 44, worker $i$ is sure that all workers have already processed the information of the previous layer, and the information of the current layer will not be mixed with the information of the previous layer. In other words, worker $i$ exits the while loop in lines 13-36 after receiving exactly one message from each worker in channel $R_i$ and all assigned states in the next layer. For the first layer, HDBS2 sends dummy messages (line 8).

**Implementation** For the open list and hash table, we use the same implementation as sequential beam search. While a worker sends a message to itself in the pseudo-code, such a message is processed without being sent to a channel. For HASH, we use FxHash from rustc-hash 1.1.0[4], take the remainder of the hash value divided by $k$, and increase it by 1 so that the resulting value is in $\{1, ..., k\}$. Although HDBS can be implemented for multi-process (distributed) environments, we focus on the multi-thread implementation in this paper. For message passing, we use Crossbeam Channel 0.5.8.[5] For broadcast in HDBS1, we use bus 2.4.0.[6]

## Empirical Evaluation

We implement multi-thread DIDP solvers using CABS with SBS (CASBS), HDBS1 (CAHDBS1), and HDBS2 (CAHDBS2) in an existing DIDP framework, didp-rs,[7] with Rust 1.65.0. We use up to 32 threads, and CABS starts from $b = 32$. We evaluate the DIDP solvers in six problems used by Kuroiwa and Beck (2023b): the traveling salesperson problem with time windows (TSPTW), the capacitated vehicle routing problem (CVRP), the simple assembly line balancing problem (SALBP-1), bin packing, the minimization of open stacks problem (MOSP) (Yuen and Richardson 1995), and graph-clear (Kolling and Carpin 2007). We also evaluate MIP and CP models using Gurobi 10.0.1 and IBM ILOG CP Optimizer 22.1.0. We use the DyPDL models from Kuroiwa and Beck (2023c) and MIP and CP models from Kuroiwa and Beck (2023b)[8] with Python 3.11.2. For a memory allocator, jemalloc 5.2.1[9] is used. All experiments are run on a machine with 2 Intel Xeon Gold 6148 CPUs (40 cores in total) using 188 GB memory and 300 seconds.

### Evaluation Measures

We evaluate the coverage: the number of optimally solved problem instances within the time and memory limits. We measure speedup by the time that the sequential solver takes to optimally solve an instance divided by that of a parallel solver, excluding the time to create a model. As discussed, parallelization may change the behavior of beam search, so we also evaluate the speedup of search time per expansion.

To evaluate the behavior difference between sequential and parallel CABS, we compute the expansion ratio, the number of expansions by a parallel solver divided by that of the sequential solver. A similar metric, search overhead, was used by previous work (Kishimoto, Fukunaga, and Botea 2013), which is our expansion ratio minus 1.[10]

We use the primal gap and primal integral (Berthold 2013) following Kuroiwa and Beck (2023c). When a solver finds a solution with cost $f(x)$, and the best-known solution cost is $f(x^*)$, the primal gap is $(f(x) - f(x^*))/f(x)$. When no solution is found, the primal gap is 1. The primal integral measures the solution quality over time. Let $p(t)$ be the primal gap achieved at time $t$. When the time limit is $T$, and a solver finds a better solution at time $t_i$ for $i = 1, ..., l-1$, the primal integral is $\sum_{i=0}^{l} p(t_i)$, where $t_0 = 0$ and $t_l = T$.

### Experimental Results

In Figure 1, we show the speedup of parallel DIDP solvers with 8, 16, and 32 threads using the problems where DIDP outperforms MIP and CP in coverage according to previous work (Kuroiwa and Beck 2023c): TSPTW, SALBP-1, MOSP, and graph-clear. All problems solved by sequential CABS are solved by the parallel solvers except for one instance of SALBP-1 unsolved by CASBS with 8 threads.

For 32 threads, the speedup is shown in Table 1 with the coverage, the primal gap, the primal integral, and the expansion ratio. Table 1 includes results for all problems studied by Kuroiwa and Beck (2023b). HDBS achieves better speedup than SBS. CAHDBS2 is better than CAHDBS1 in TSPTW and SALBP-1, and the difference is small in MOSP and graph-clear. We compare 1 and 32 thread solvers for each instance in the appendix (Kuroiwa and Beck 2023a).

In TSPTW, CVRP, MOSP, and graph-clear, the expansion ratio is close to 1, and the speedup is close to the speedup per expansion, indicating that the effect on the search behavior change is small. However, in SALBP-1, the expansion ratio is below 1, and the speedup is larger than the speedup per expansion, particularly in HDBS. In the DyPDL model for SALBP-1, the dual bound is strong, and beam search proves the optimality as soon as it finds an optimal solution in some instances. Thus, the result suggests that the search behavior change helps to find a good solution in SALBP-1. We observe a more extreme tendency in bin packing, where a similar DyPDL model is used, resulting in a super linear speedup of HDBS. CASBS solves fewer instances than sequential CABS. While CAHDBS1 and CAHDBS2 solve more instances in total, they do not solve 36 and 41 instances solved by sequential CABS. We compare the numbers of expanded states for each problem in the appendix.

Comparing coverage, the primal gap, and the primal integral, parallel DIDP solvers with 32 threads are significantly better than sequential CABS. CAHDBS1 and CAHDBS2 find new best solutions for rbg193.2.tw (12138) and rbg233.2.tw (14492) in the AFG set of TSPTW.

MIP and CP achieve a smaller speedup than DIDP. In TSPTW and MOSP, while parallel CP has the better primal gap, it is slower and solves fewer instances than sequential CP. In terms of overall performance, parallel DIDP is better (worse) than parallel MIP and CP if the sequential DIDP is better (worse) than the sequential MIP and CP.

[4]https://crates.io/crates/rustc-hash/1.1.0

[5]https://crates.io/crates/crossbeam-channel/0.5.8

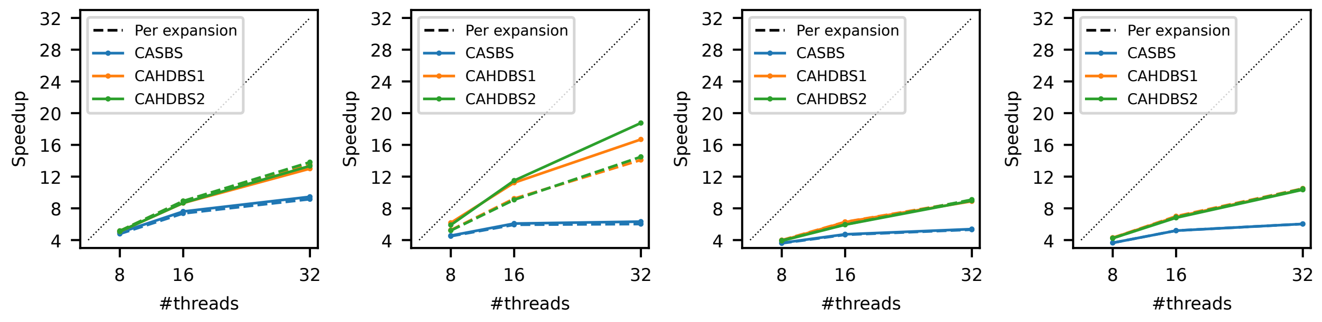
[6]https://crates.io/crates/bus/2.4.0

[7]https://github.com/domain-independent-dp/didp-rs/releases/tag/parallel-aaai24

[8]https://github.com/Kurorororo/didp-models

[9]https://jemalloc.net/

[10]We keep the metric positive to take the geometric mean.



(a) TSPTW (b) SALBP-1 (c) MOSP (d) Graph-Clear

Figure 1: Speedup of parallel CABS methods over sequential CABS (the geometric mean over instances optimally solved by sequential CABS in 10-300 seconds). 'Per expansion' shows the speedup of search time per expansion.

| | | 1 thread | | | 32 threads | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Method | # | Gap | Integral | # | Gap | Integral | Speedup | Speedup/expansion | Expansion ratio |
| TSPTW (340) | MIP | 192 | 0.313 | 108.3 | 239 | 0.210 | 77.5 | 4.2 | - | - |
| | CP | 42 | 0.028 | 9.6 | 27 | 0.025 | 17.1 | 0.1 | - | - |
| | CASBS | | | | 260 | 0.003 | 1.5 | 9.5 | 9.2 | 0.968 |
| | CAHDBS1 | 235 | 0.006 | 2.9 | **262** | **0.002** | **1.1** | 13.0 | 13.5 | 1.036 |
| | CAHDBS2 | | | | **262** | **0.002** | **1.1** | **13.3** | **13.8** | 1.035 |
| CVRP (90) | MIP | 13 | 0.232 | 121.8 | **29** | 0.114 | 81.5 | 5.3 | - | - |
| | CP | 0 | 0.079 | 28.0 | 0 | 0.060 | 24.6 | - | - | - |
| | CASBS | | | | 6 | 0.055 | 18.5 | 5.2 | 5.2 | 0.983 |
| | CAHDBS1 | 5 | 0.055 | 18.6 | 8 | **0.040** | **13.8** | **9.3** | **9.3** | 0.999 |
| | CAHDBS2 | | | | 8 | **0.040** | **13.8** | **9.3** | **9.3** | 1.000 |
| SALBP-1 (2100) | MIP | 1322 | 0.332 | 107.4 | 1351 | 0.206 | 69.6 | 1.3 | - | - |
| | CP | 1563 | 0.011 | 16.5 | 1581 | 0.009 | 15.9 | 1.4 | - | - |
| | CASBS | | | | 1818 | 0.001 | 1.0 | 6.3 | 6.0 | 0.952 |
| | CAHDBS1 | 1714 | 0.002 | 2.0 | **1824** | **0.000** | **0.5** | 16.7 | 14.1 | 0.845 |
| | CAHDBS2 | | | | **1824** | **0.000** | **0.5** | **18.8** | **14.5** | 0.773 |
| Bin Packing (1615) | MIP | 1122 | 0.047 | 19.3 | 1192 | 0.034 | 15.5 | 6.4 | - | - |
| | CP | 1189 | 0.006 | 4.5 | **1251** | **0.002** | **1.3** | 9.2 | - | - |
| | CASBS | | | | 1077 | 0.006 | 3.6 | 3.9 | 4.2 | 1.089 |
| | CAHDBS1 | 1110 | 0.003 | 4.9 | 1239 | 0.003 | 1.8 | 36.4 | 10.5 | 0.288 |
| | CAHDBS2 | | | | 1239 | 0.003 | 1.8 | **39.6** | **11.0** | 0.278 |
| MOSP (570) | MIP | 216 | 0.059 | 26.4 | 238 | 0.050 | 22.9 | 3.1 | - | - |
| | CP | 421 | 0.008 | 4.2 | 397 | 0.005 | 4.8 | 0.3 | - | - |
| | CASBS | | | | 526 | **0.000** | 0.3 | 5.4 | 5.3 | 0.987 |
| | CAHDBS1 | 507 | **0.000** | 0.2 | **531** | **0.000** | **0.1** | 8.9 | 9.0 | 1.012 |
| | CAHDBS2 | | | | **531** | **0.000** | **0.1** | **9.0** | **9.1** | 1.015 |
| Graph-Clear (135) | MIP | 6 | 0.214 | 80.4 | 16 | 0.187 | 72.6 | 2.0 | - | - |
| | CP | 4 | 0.048 | 22.6 | 3 | 0.040 | 22.0 | 3.2 | - | - |
| | CASBS | | | | 103 | **0.000** | 0.2 | 6.0 | 6.0 | 0.999 |
| | CAHDBS1 | 92 | **0.000** | 0.3 | **113** | **0.000** | **0.1** | **10.4** | **10.5** | 1.011 |
| | CAHDBS2 | | | | **113** | **0.000** | **0.1** | 10.3 | **10.5** | 1.011 |

Table 1: Comparison of solvers with 1 and 32 threads. '#' is the coverage, 'Gap' is the primal gap, and 'Integral' is the primal integral (arithmetic mean). We show the geometric mean speedup in instances solved by a sequential solver in 10-300 seconds.

## Conclusion and Future Work

We proposed parallel beam search algorithms for domain-independent dynamic programming (DIDP). Hash distributed beam search (HDBS) distributes states to threads based on hash values using message passing, adapted from parallel best-first search (BFS). HDBS achieves significant speedup and performance improvement over the state-of-the-art sequential DIDP solver. A potential bottleneck of HDBS is frequent communication between threads. In parallel BFS, such overhead can be reduced by abstracted hashing (Jinnai and Fukunaga 2017), which tends to assign successors of state $S$ to the same thread assigned to $S$, exploiting problem structure. Designing such a hash function for DIDP is future work.

## Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

## References

Berthold, T. 2013. Measuring the Impact of Primal Heuristics. *Operations Research Letters*, 41: 611–614.

Burns, E.; Lemons, S.; Ruml, W.; and Zhou, R. 2010. Best-First Heuristic Search for Multicore Machines. *Journal of Artificial Intelligence Research*, 39: 689–743.

Edelkamp, S.; Jabbar, S.; and Lafuente, A. L. 2005. Cost-Algebraic Heuristic Search. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 1362–1367.

Frohner, N.; Gmys, J.; Melab, N.; Raidl, G.; and Talbi, E.-G. 2023. Parallel Beam Search for Combinatorial Optimization. In *Workshop Proceedings of the 51st International Conference on Parallel Processing*.

Jinnai, Y.; and Fukunaga, A. 2017. On Hash-Based Work Distribution Methods for Parallel Best-First Search. *Journal of Artifiicial Intelligence Research*, 60: 491–548.

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2013. Evaluation of a Simple, Scalable, Parallel Best-First Search Strategy. *Artificial Intelligence*, 195: 222–248.

Kolling, A.; and Carpin, S. 2007. The GRAPH-CLEAR Problem: Definition, Theoretical Properties and Its Connections to Multirobot Aided Surveillance. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1003–1008.

Kuroiwa, R.; and Beck, J. C. 2023a. Appendix for Parallel Beam Search Algorithms for Domain-Independent Dynamic Programming. https://tidel.mie.utoronto.ca/pubs/Appendix_Parallel_AAAI24.pdf. Accessed: 2023-12-13.

Kuroiwa, R.; and Beck, J. C. 2023b. Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 236–244.

Kuroiwa, R.; and Beck, J. C. 2023c. Solving Domain-Independent Dynamic Programming Problems with Anytime Heuristic Search. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 245–253.

Kuroiwa, R.; and Fukunaga, A. 2019. On the Pathological Search Behavior of Distributed Greedy Best-First Search. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, 255–263.

Kuroiwa, R.; and Fukunaga, A. 2020. Analyzing and Avoiding Pathological Behavior in Parallel Best-First Search. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 175–183.

Yuen, B. J.; and Richardson, K. V. 1995. Establishing the Optimality of Sequencing Heuristics for Cutting Stock Problems. *European Journal of Operational Research*, 84: 590–598.

Zhang, W. 1998. Complete Anytime Beam Search. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI-98/IAAI-98)*, 425–430.