Large Neighborhood Beam Search for Domain-Independent Dynamic Programming

Ryo Kuroiwa
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Abstract

Large neighborhood search (LNS) is an algorithmic framework that removes a part of a solution and performs search in the induced search space to find a better solution. While LNS shows strong performance in constraint programming, little work has combined LNS with state space search. We propose large neighborhood beam search (LNBS), a combination of LNS and state space search. Given a solution path, LNBS removes a partial path between two states and then performs beam search to find a better partial path. We apply LNBS to domain-independent dynamic programming (DIDP), a recently proposed generic framework for combinatorial optimization based on dynamic programming. We empirically show that LNBS finds better quality solutions than a state-of-the-art DIDP solver in five out of nine benchmark problem types with a total of 8570 problem instances. In particular, LNBS shows a significant improvement over the existing state-of-the-art DIDP solver in routing and scheduling problems.

2012 ACM Subject Classification Computing methodologies → Discrete space search

Keywords and phrases Large Neighborhood Search, Dynamic Programming, State Space Search, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2023.23

Supplementary Material *Software (Source Code)*: `https://github.com/domain-independent-dp/didp-rs/releases/tag/lnbs-cp23`
archived at `swh:1:dir:b531b883c4c9eda4e0452fe801a5b50f6a3970cc`

Funding This research is supported by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

In constraint programming (CP), large neighborhood search (LNS) [34] achieves strong performance in solving combinatorial optimization problems such as routing [24] and scheduling problems [27]. LNS is an algorithmic framework that removes a part of a solution and then performs search in the induced partial search space (neighborhood) to find a better solution. Typically, LNS uses tree search to find a better solution in a partial search space, where each search node represents a partial assignment of decision variables, and a solution of a problem corresponds to a leaf node, where all variables are assigned values.

Dynamic programming (DP) is a powerful method for multiple combinatorial optimization problems [14, 15], and the hybridization of CP, decision diagrams, and DP is a topic of active research [1, 22, 23, 5, 28, 19, 17]. Recently, domain-independent dynamic programming (DIDP), a model-based paradigm for combinatorial optimization based on DP, has been proposed [25]. In DIDP, a model of a problem is represented by a state transition system. A solution corresponds to a path in a state space graph, where each vertex represents a state and each edge represents a transition between two states. The current state-of-the-art DIDP solver is complete anytime beam search (CABS) [26], which is based on beam search, an algorithm that searches for a path in a state space graph by maintaining a fixed number of states at a time.

In this paper, we propose large neighborhood beam search (LNBS), a combination of LNS and beam search in a state space graph. LNBS tries to improve a solution path by removing a partial path between two states and then performing beam search to find a better partial path. While LNBS has the freedom to select a neighborhood (i.e., a partial path to remove), we propose a strategy that dynamically adjusts the size of a neighborhood based on a multi-armed bandit problem. With our strategy, LNBS is complete, i.e., it finds and proves an optimal solution given enough time, but, of course, it is aimed at problems where its solution quality is more important than proved optimality. We implement LNBS for DIDP and empirically evaluate its performance. The experimental results show that LNBS outperforms CABS in five out of nine benchmark problem types in terms of solution quality. In addition, LNBS performs better than a commercial CP solver, which uses LNS [27], in seven problems while CABS is better than CP in six problems. Since LNBS performs particularly well in routing and scheduling problems, we also investigate the reason for this performance and gain insight from empirical analysis.

2 Background

We first introduce domain-independent dynamic programming and complete anytime beam search. Then, we present large neighborhood search (LNS). We also describe LNS with decision diagrams [18], a recently proposed method for combinatorial optimization that can be considered a combination of LNS and state space search.

2.1 Domain-Independent Dynamic Programming

A combinatorial optimization problem is to find a set of discrete decisions, e.g., a permutation, to minimize or maximize an objective function. In dynamic programming (DP), a problem is recursively formulated by decomposing it into subproblems, represented by *states*, and the optimal objective value of each subproblem is represented by the *value function*, which maps a state to a real number.

Domain-independent dynamic programming (DIDP) is a model-based paradigm for combinatorial optimization based on DP [25]. In DIDP, a DP formulation of a combinatorial optimization problem is defined by a state-transition system in Dynamic Programming Description Language (DyPDL). A DyPDL model is a seven-tuple $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$ consisting of *state variables* $\mathcal{V}$, the *target state* S^0, *constants* $\mathcal{K}$, *transitions* $\mathcal{T}$, *base cases* $\mathcal{B}$, *state constraints* $\mathcal{C}$, and the *dual bound* h.

A state variable $v \in \mathcal{V}$ has a type of *set*, *element*, or *numeric*. Each set and element variable v_i is associated with a set of objects $N_i = \{0, ..., n_i - 1\}$. The domain of a set variable v_i is 2^{N_i}, and the domain of an element variable v_i is N_i. A numeric variable takes a real value. A constant in $\mathcal{K}$ is a value independent of the state variables.

A state is a complete value assignment to the state variables, and the target state S^0 is a state. We denote the value of a state variable v_i in a state S by $S[v_i]$. The value of a state S is represented by the value function $V(S)$, and the objective of a DyPDL model is to compute the value of the target state, $V(S^0)$. We can define dominance between states based on state variables. If a state S dominates another state S', denoted by $S' \preceq S$, then $V(S)$ is equal or better (less/greater for minimization/maximization) than $V(S')$. For the details of dominance in DyPDL, please refer to previous work [25].

A transition $\tau \in \mathcal{T}$ is a four-tuple $\langle \mathsf{eff}_\tau, \mathsf{cost}_\tau, \mathsf{pre}_\tau, \mathsf{forced}_\tau \rangle$ defining how a state S is transformed to a successor state (a subproblem) $S[\![\tau]\!]$, and how $V(S)$ is computed based on the value of $V(S[\![\tau]\!])$. The set of *effects* $\mathsf{eff}_\tau = \{(v_i, e_{v_i}) \mid v_i \in \mathcal{V}\}$ defines how each state variable v_i is updated by an *expression* e_{v_i}. The expression e_{v_i} consists of predefined operations on state variables and constants and returns $S[\![\tau]\!][v_i]$ given S. For example, for a set variable U, $U \setminus \{i\}$ is an expression representing removing an element i from U, which can be used as e_U and results in $S[\![\tau]\!][U] = S[U] \setminus \{i\}$. The *cost expression* cost_τ is an expression that takes $V(S[\![\tau]\!])$ in addition to S and returns a real number $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S)$. The preconditions pre_τ are conditions on the state variables, i.e., expressions returning a binary value $\top$ or $\bot$. The preconditions define when the transition is *applicable*. For example, if $\mathsf{pre}_\tau = \{i \in U\}$, then τ is applicable in S iff $i \in S[U]$. The flag forced_τ is a boolean value indicating whether the transition is a *forced transition*. Let $\mathcal{T}(S)$ be the set of applicable transitions in state S. If forced transitions are applicable, the first defined one τ is selected, and all other forced and non-forced transitions are ignored, i.e., $\mathcal{T}(S) = \{\tau\}$. The value of $V(S)$ is computed by taking the best $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S)$ over all $\tau \in \mathcal{T}(S)$.

Base cases $\mathcal{B}$ are sets of conditions. For any $B \in \mathcal{B}$, if a state S satisfies all conditions in B, denoted by $S \models B$, then it is called a *base state*, and $V(S)$ is defined non-recursively by an expression $e_B(S)$. State constraints $\mathcal{C}$ are conditions that must be satisfied by all states. If one of the state constraints $c \in \mathcal{C}$ is violated, denoted by $S \not\models \mathcal{C}$, then $V(S) = \infty$ ($V(S) = -\infty$) for minimization (maximization). The dual bound $h(S)$ is a lower (upper) bound on $V(S)$ for minimization (maximization).

Overall, if the problem is minimization, the DP formulation is defined as follows.

$$\text{compute } V(S^0) \tag{1}$$

$$\text{s.t. } V(S) = \begin{cases} \min_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S) & \text{if } S \models \mathcal{C} \land \forall B \in \mathcal{B}, S \not\models B \\ e_B(S) & \text{if } S \models \mathcal{C} \land \exists B \in \mathcal{B}, S \models B \\ \infty & \text{if } S \not\models \mathcal{C} \end{cases} \tag{2}$$

$$V(S) \leq V(S') \qquad \text{if } S' \preceq S \tag{3}$$

$$V(S) \geq h(S). \tag{4}$$

The first line states that the optimal objective value is $V(S^0)$. Equation (2) recursively defines the value function V. Inequalities (3) and (4) are bounds on the value function. For maximization, we replace min with max in the first line of Equation (2) and swap $\leq$ and $\geq$ in Inequalities (3) and (4). A *solution* for the DP formulation is a sequence of transitions that transforms the target state S^0 into a base state. Concretely, for a sequence of transitions $x = \langle x_1, ..., x_n \rangle$, let $S^{i+1} = S^i[\![x_{i+1}]\!]$ for $i = 0, ..., n-1$. Then, x is a solution if $x_{i+1} \in \mathcal{T}(S^i)$, $S^i \models \mathcal{C}$, and $\forall B \in \mathcal{B}, S^i \not\models B$ for $i = 0, ..., n-1$, $S^n \models \mathcal{C}$, and $\exists B \in \mathcal{B}, S^n \models B$. The solution is an *optimal solution* if $\mathsf{cost}_{\tau_i}(V(S^{i+1}), S^i) = V(S^i)$ for $i = 0, ..., n-1$ in addition.

2.2 Complete Anytime Beam Search for DIDP

Previous research has shown that a subset of DyPDL models can be solved by cost-algebraic heuristic search [11], a generalized version of the shortest path algorithm [25]. Multiple DIDP solvers using cost-algebraic heuristic search algorithms have been proposed [25, 26]. These solvers perform *state space search*, which finds a path from the target state to a base state in a *state space graph*, a directed graph where each vertex is a state. In the state space graph, an edge from state S to $S[\![\tau]\!]$ exists if $\tau \in \mathcal{T}(S)$. Following the previous work, we assume that $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S)$ is expressed as $w_\tau(S) \times V(S[\![\tau]\!])$ where w_τ is an expression returning

Algorithm 1 Beam Search for DyPDL.

```
1: function BeamSearch($\overline{f}$, $b$)
2:     $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), x(S^0) \leftarrow \langle\rangle$.
3:     $O \leftarrow \{S^0\}, \overline{x} \leftarrow$ NULL, complete $\leftarrow \top$.
4:     while $O \neq \emptyset$ and $\overline{x} =$ NULL do
5:         $G \leftarrow \emptyset$.                                    ▷ A set of states in the next layer.
6:         for all $S \in O$ do
7:             if $\exists B \in \mathcal{B}$ such that $S \models B$ then                      ▷ A base state.
8:                 if $g(S) \times e_B(S) < \overline{f}$ then                       ▷ A better solution.
9:                     $\overline{f} \leftarrow g(S) \times e_B(S), \overline{x} \leftarrow x(S)$.
10:            else
11:                for all $\tau \in \mathcal{T}(S) : S[\![\tau]\!] \models \mathcal{C}$ do
12:                    if $\nexists S' \in G$ such that $S[\![\tau]\!] \preceq S'$ and $g(S) \times w_\tau(S) \geq g(S')$ then
13:                        $x(S[\![\tau]\!]) \leftarrow \langle x(S); \tau\rangle$.
14:                        $g(S[\![\tau]\!]) \leftarrow g(S) \times w_\tau(S), f(S[\![\tau]\!]) \leftarrow g(S[\![\tau]\!]) \times h(S[\![\tau]\!])$.
15:                        if $\exists S' \in G$ such that $S' \preceq S[\![\tau]\!]$ and $g(S[\![\tau]\!]) \leq g(S')$ then
16:                            $G \leftarrow G \setminus \{S'\}$.                ▷ Remove a dominated state.
17:                        if $f(S[\![\tau]\!]) < \overline{f}$ then                 ▷ Pruning by the primal bound.
18:                            $G \leftarrow G \cup \{S[\![\tau]\!]\}$.
19:        $O \leftarrow \{S \in G \mid f(S) < \overline{f}\}$.
20:        if $|O| > b$ then
21:            $O \leftarrow$ the best $b$ states in $G$ minimizing $f$.
22:            complete $\leftarrow \bot$.
23:    if $O \neq \emptyset$ then
24:        complete $\leftarrow \bot$.
25:    return $\overline{x}$, complete.
```

a real value and $\times$ is a binary operator satisfying a cost-algebra. In particular, we focus on nonnegative w_τ and binary operators $+$ or max, i.e., $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S) = w_\tau(S) + V(S[\![\tau]\!])$ or $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S) = \max\{w_\tau(S), V(S[\![\tau]\!])\}$. The weight of the edge $(S, S[\![\tau]\!])$ is defined as $w_\tau(S)$, and the cost of a path from the target state S^0, which corresponds to a sequence of the transitions $x = \langle x_1, ..., x_n\rangle$, is $\mathsf{cost}_x(S^0) = \bigtimes_{i=0}^{n-1} w_{x_{i+1}}(S^i)$ where $S^{i+1} = S^i[\![x_{i+1}]\!]$ for $i = 0, ..., n-1$. As each edge weight is nonnegative, the cost of a path is nonnegative and non-decreasing in length. In this paper, we focus on minimization while DyPDL and cost-algebraic heuristic search can handle both minimization and maximization.

The state-of-the-art cost-algebraic heuristic search solver is complete anytime beam search (CABS) [36, 26]. CABS performs beam search, which searches at most b states in the *open list* O at each layer of the state space graph. We show the pseudo-code of beam search in Algorithm 1. In addition to a DyPDL model and b, beam search takes the primal bound $\overline{f}$ as an input, which is the best-known objective value and could be infinity. With each state S, the best path $x(S)$ from S^0 and its cost $g(S) = \mathsf{cost}_{x(S)}(S^0)$ (the *g-value*) are maintained in lines 13 and 14. Starting from $O = \{S^0\}$, beam search processes a state S in O. If S is a base state, and the best path to S has a better cost than $\overline{f}$ in line 8, then $\overline{f}$ and the solution $\overline{x}$ are updated. Otherwise, $S[\![\tau]\!]$ is added to the candidate set G for each transition $\tau \in \mathcal{T}(S)$, which is called the *expansion* of S, and we say that S is expanded (lines 11– 18). When expanding $S \in O$, if there exists a state $S' \in G$ that dominates $S[\![\tau]\!]$ and $g(S') \leq g(S) \times w_\tau(S)$, then $S[\![\tau]\!]$ is not added to G. In addition to $g(S[\![\tau]\!])$, the priority

$f(S[\![\tau]\!]) = g(S[\![\tau]\!]) \times h(S[\![\tau]\!])$ (the *f-value*) is computed in line 14, which is a lower bound on the optimal path cost from S^0 to a base state via $S[\![\tau]\!]$. If $f(S[\![\tau]\!]) \geq \overline{f}$, the corresponding path does not lead to an improved solution, so $S[\![\tau]\!]$ is not added to G in line 18. After expanding all states, O is updated to the best b states in G according to f in lines 20-21. This procedure is repeated until a solution whose cost is better than the given primal bound is found, or no successor states are generated. The variable complete maintains whether the search is complete. If states in G are pruned due to the beam width b (line 22), or O is not empty when a solution is found (line 24), there may exist a better solution, so the search is not complete. If complete $= \top$, then $\overline{x}$ is the optimal solution if it is not NULL, or the model is infeasible if $\overline{x} =$ NULL. CABS performs a sequence of beam search with exponentially increasing beam width $b = 1, 2, 4, ...$ using the best objective value found so far as the primal bound $\overline{f}$ until the optimality of the best solution or the infeasibility is proved.

2.3 Large Neighborhood Search

Large neighborhood search (LNS) iteratively removes a part of a solution and solves the resulting subproblem (neighborhood) [34]. A solution for a CP problem is represented as a complete value assignment to decision variables. Given a solution, LNS removes a subset of the value assignments and solves the subproblem where the remaining variables are fixed to the values assigned in the original solution. Typically, a tree search algorithm is used. In a tree search algorithm, a search node is a partial value assignment to decision variables, successor nodes are generated by assigning a value to an unassigned variable, and a solution corresponds to a leaf node, where all variables are assigned values.

2.3.1 Large Neighborhood Search with Decision Diagrams

For combinatorial optimization, recent work proposed LNS with decision diagrams (DD-LNS) [18]. Although DD-LNS was not explicitly framed as state space search, we interpret it as a state space search algorithm. While DD-LNS is independent of DIDP, it also uses the DP formulation of a problem as input while assuming that the solution has n transitions. Given a sequence of transitions $\langle x_1, ..., x_n \rangle$, DD-LNS keeps the first d transitions, $\langle x_1, ..., x_d \rangle$, and searches for the remaining $n - d$ transitions. To find such a sequence, DD-LNS constructs a decision diagram (DD), a directed graph where nodes are partitioned into layers. In the constructed DD, each vertex corresponds to a state, and each edge corresponds to a transition, so it is a state space graph. The first layer in the DD contains only the node corresponding to S^d, where $S^i = S^{i-1}[\![x_i]\!]$ for $i = 1, ..., d$. DD-LNS iteratively constructs a layer by applying transitions to the states in the current layer until reaching layer $n - d + 1$.

Because constructing an exact DD is intractable, DD-LNS constructs a restricted DD, which keeps a subset of states in the exact DD. In each layer, states satisfying certain conditions are kept, some of which are selected randomly. Let the number of such states be K. If K is smaller than a parameter W, from the remaining states, DD-LNS also keeps the best $W - K$ states that minimize a priority function, the rough lower bound (RLB). The RLB of a state is a lower bound on the cost of a solution path via that state, which is the same as the f-value. If the RLB of a state is larger than the best solution cost, the state is removed from the DD as it does not lead to a better solution. Therefore, the procedure of constructing a restricted DD can be considered beam search with randomization. Indeed, the authors acknowledged that DD-LNS is a hybridization of LNS and beam search [18]. DD-LNS decreases d by 1 if a better solution is not found with d, starting from $d = n - 2$ and restarting from $d = n - 2$ if $d = 0$. When $d = 0$ and the restricted DD keeps all states except

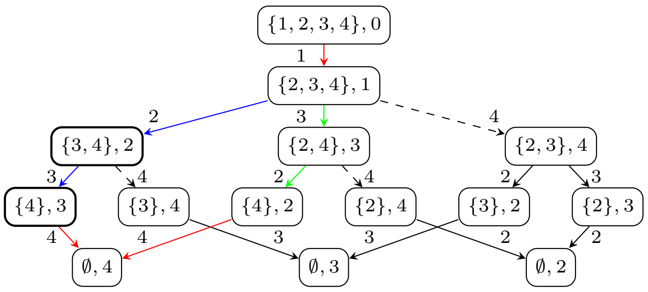


Figure 1 Partial state space graph induced by the prefix $\langle 1\rangle$ and the suffix $\langle 4\rangle$ (highlighted in red) in Example 1. The current partial path is highlighted in blue and an alternative partial path is highlighted in green. Dashed transitions conflict with the suffix (explained in Section 3.2).

for those removed based on RLB, then it is the exact DD, so DD-LNS proves the optimality of the solution. Since DD-LNS can be interpreted as a state space search algorithm and is designed for combinatorial optimization, we implement it for DIDP and experimentally compare it with our method.

3 Large Neighborhood Beam Search

We start with a simple idea of LNS for state space search: given a solution path, $x = \langle x_1, ..., x_n\rangle$ which connects states $\langle S^0, ..., S^n\rangle$, we remove a partial path $\langle x_i, ..., x_{i+d-1}\rangle$ and search for a better partial path from S^{i-1} to S^{i+d-1}. If we find a better solution $\langle x_1, ..., x_{i-1}, x'_i, ..., x'_{i+d'-1}, x_{i+d} ..., x_n\rangle$, we repeat this procedure with the new solution. While the overview of the algorithm is simple, there are design choices on how to select a partial path to remove and how to search for a better partial path. The novelty of our method compared to existing methods arises from such choices in addition to the fact that it is used for DIDP. First, we describe the modifications of beam search for DyPDL to search for a partial path from S^{i-1} to S^{i+d-1}. Then, we propose strategies to select a partial path to remove.

3.1 Beam Search for DyPDL in a Partial State Space Graph

We want to find a path from S^{i-1} to S^{i+d-1} instead of from S^0 to a base state. We could modify line 8 in Algorithm 1 so that it checks if $S = S^{i+d-1}$ instead of $\exists B \in \mathcal{B}, S \models B$. However, in DyPDL, it may not be desirable as shown in the following example.

▶ **Example 1.** Consider the following DP formulation, where $U \subseteq \{0, 1, 2, 3, 4\}$ is a set variable, $k \in \{0, 1, 2, 3, 4\}$ is an element variable, and c_{lj} for $l, j \in \{0, 1, 2, 3, 4\}$ is a constant.

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

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

Each transition in the DyPDL model has precondition $j \in U$ and effect $(U, U \setminus \{j\})$ for some j, and so we denote each transition by j. Each solution corresponds to a permutation of the transitions $1, 2, 3, 4$. A solution $\langle 1, 2, 3, 4\rangle$ connects a sequence of states $\langle(\{1, 2, 3, 4\}, 0)$, $(\{2, 3, 4\}, 1)$, $(\{3, 4\}, 2)$, $(\{4\}, 3)$, $(\emptyset, 4)\rangle$. Consider removing $\langle 2, 3\rangle$ from the solution. We visualize the partial state space graph in Figure 1. An algorithm tries to find a path from $(\{2, 3, 4\}, 1)$ to $(\{4\}, 3)$. The original one, $\langle 2, 3\rangle$, is only the path. However, a partial path $\langle 3, 2\rangle$ from $(\{2, 3, 4\}, 1)$ to $(\{4\}, 2)$ also results in a valid solution $\langle 1, 3, 2, 4\rangle$.

Algorithm 2 Beam Search in a partial state space graph.

```
1:  function BeamSearchForPartialPath($\overline{f}$, $b$, prefix, suffix)
2:      $\hat{S} \leftarrow S^0[\![\mathsf{prefix}]\!], g(\hat{S}) \leftarrow \mathsf{cost}_{\mathsf{prefix}}(S^0), f(\hat{S}) \leftarrow g(\hat{S}) \times h(\hat{S}), x(\hat{S}) \leftarrow \mathsf{prefix}$.
3:      $O \leftarrow \{\hat{S}\}$, $\overline{x} \leftarrow \mathrm{NULL}, \mathsf{complete} \leftarrow \top$.
4:      while $O \neq \emptyset$ and $\overline{x} = \mathrm{NULL}$ do
5:          $G \leftarrow \emptyset$.                                        ▷ A set of states in the next layer.
6:          for all $S \in O$ do
7:              $S' \leftarrow S, \mathsf{success} \leftarrow \bot$.
8:              if $\exists B \in \mathcal{B}$ and $S' \models B$ then
9:                  $\mathsf{success} \leftarrow \top$.                          ▷ A base state, success.
10:             else
11:                 for $\tau \leftarrow \mathsf{suffix}_1, ..., \mathsf{suffix}_{n-i-d+1}$ do          ▷ Rollout of the suffix.
12:                     if $S' \not\models \mathsf{pre}_\tau$ then
13:                         break.                     ▷ Preconditions are not satisfied, fail.
14:                     $S' \leftarrow S'[\![\tau]\!], g(S') \leftarrow g(S') \times w_\tau(S'), x(S') \leftarrow \langle x(S'); \tau \rangle$.
15:                     if $S' \not\models \mathcal{C}$ then
16:                         break.                     ▷ State constraints are not satisfied, fail.
17:                     if $\exists B \in \mathcal{B}$ such that $S' \models B$ then
18:                         $\mathsf{success} \leftarrow \top$.                  ▷ A base state, success.
19:                         break.
20:             if success then
21:                 if $g(S') \times e_B(S') < \overline{f}$ then                    ▷ A better solution.
22:                     $\overline{f} \leftarrow g(S') \times e_B(S'), \overline{x} \leftarrow x(S')$.
23:             else
24:                 for all $\tau \in \mathcal{T}(S) : S[\![\tau]\!] \models \mathcal{C}$ do
25:                     if $\nexists S' \in G$ such that $S[\![\tau]\!] \preceq S'$ and $g(S) \times w_\tau(S) \geq g(S')$ then
26:                         $x(S[\![\tau]\!]) \leftarrow \langle x(S); \tau \rangle$.
27:                         $g(S[\![\tau]\!]) \leftarrow g(S) \times w_\tau(S), f(S[\![\tau]\!]) \leftarrow g(S[\![\tau]\!]) \times h(S[\![\tau]\!])$.
28:                         if $\exists S' \in G, S' \preceq S[\![\tau]\!] \wedge g(S[\![\tau]\!]) \leq g(S')$ then
29:                             $G \leftarrow G \setminus \{S'\}$.              ▷ Remove a dominated state.
30:                         if $f(S[\![\tau]\!]) < \overline{f}$ then             ▷ Pruning by the primal bound.
31:                             $G \leftarrow G \cup \{S[\![\tau]\!]\}$.
32:         $O \leftarrow \{S \in G \mid f(S) < \overline{f}\}$.
33:         if $|G| > b$ then
34:             $O \leftarrow$ the best $b$ states in $G$ minimizing $f$.
35:             $\mathsf{complete} \leftarrow \bot$.
36:     if $O \neq \emptyset$ then
37:         $\mathsf{complete} \leftarrow \bot$.
38:     return $\overline{x}$, complete.
```

Considering the above example, instead of focusing on a partial path to a state, we focus on a partial path to a *suffix* of the solution path. Given a solution path $\langle x_1, ..., x_n \rangle$, if we remove a partial path $\langle x_i, ..., x_{i+d-1} \rangle$, then $\langle x_1, ..., x_{i-1} \rangle$ is the prefix, and $\langle x_{i+d}, ..., x_n \rangle$ is the suffix. For a partial path $\langle x'_i, ..., x'_{i+d'-1} \rangle$, we want to check if $\langle x_1, ..., x_{i-1}, x'_i, ..., x'_{i+d'-1}, x_{i+d}, ..., x_n \rangle$ is a valid solution. Therefore, for a state S found by a search algorithm, we perform a rollout of the suffix from S and check if each of resulting states satisfies the state constraints and a

Algorithm 3 Large Neighborhood Beam Search (LNBS).

Input: initial feasible solution $\overline{x}$.
Output: solution $\overline{x}$ and if the optimality or infeasibility is proved.

```
1: while the time limit is not reached do
2:     $n \leftarrow |\overline{x}|, \overline{f} \leftarrow \mathsf{cost}_{\overline{x}}(S^0)$.
3:     Select $d$ such that $2 \leq d \leq n$.
4:     Select $i$ such that $1 \leq i \leq n - d + 1$.
5:     Select beam width $b$.
6:     $\mathsf{prefix} \leftarrow \langle \overline{x}_1, ..., \overline{x}_{i-1} \rangle$, $\mathsf{suffix} \leftarrow \langle \overline{x}_{i+d}, ..., \overline{x}_n \rangle$.
7:     $x, \mathsf{complete} \leftarrow$ BeamSearchForPartialPath($\overline{f}$, $b$, prefix, suffix)
8:     if $x \neq$ NULL then
9:         $\overline{x} \leftarrow x$.
10:    if $i = 0 \wedge d = n \wedge \mathsf{complete}$ then
11:        return $\overline{x}$, $\top$.
12: return $\overline{x}$, $\bot$.
```

base case. We show the modified version of beam search in Algorithm 2. This algorithm takes a prefix prefix and a suffix suffix as input. In line 2, we denote the state resulting from applying the prefix to the target state by $\hat{S} = S^0[\![\mathsf{prefix}]\!]$ and initialize the open list O with $\hat{S}$ in line 3. In lines 8–22, the algorithm performs a rollout of the suffix from S and checks if it results in a better solution. Other parts are the same as Algorithm 1.

We use this modified version of beam search in large neighborhood beam search (LNBS) as shown in Algorithm 3. In line 2, $|\overline{x}|$ denotes the length of the current solution $\overline{x}$. In lines 3–5, LNBS selects parameters d, i, and b. In line 7, LNBS performs beam search in the neighborhood. If an improving solution is found, LNBS updates the current solution $\overline{x}$ in line 9. If the searched neighborhood is the original search space, i.e., $i = 1$ and $d = n$, and beam search proves the optimality or infeasibility, LNBS terminates in line 11. Therefore, if it is guaranteed to select $i = 1$ and $d = n$ with sufficiently large b given enough time, LNBS is guaranteed to find the optimal solution or prove the infeasibility, i.e., it is complete. CABS can be considered a configuration of LNBS, where $i = 1$, $d = n$, and b increases exponentially. DD-LNS can also be considered a configuration of LNBS, where i ranges from $n - 2$ to 1, d is $n - i + 1$, and b is fixed to W while beam search is extended with the randomization mechanism. We will describe the strategies that we use to select d, i, and b below.

3.2 Removing Conflicting Transitions

In Example 1, consider finding a partial path from a prefix $\langle 1 \rangle$, which results in state $\hat{S} = (\{2, 3, 4\}, 1)$, to a suffix $\langle 4 \rangle$ using beam search. In $\hat{S}$, three transitions 2, 3, and 4 are applicable. However, applying transition 4 does not lead to a feasible solution because it is already used in the suffix and cannot be applied twice: it requires $4 \in U$ and removes 4 from U, but no other transition adds 4 to U, so applying 4 makes the suffix inapplicable. Generalizing this example, if we know that a transition τ makes a transition τ' in the suffix inapplicable, then we can ignore τ when searching for a partial path. In particular, we focus on the effects of τ that add/remove an element to/from a set variable and the preconditions of τ' that require the element to be/not to be in that set variable.

▶ **Proposition 2.** *Suppose that a DyPDL model $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$ has a set variable $U \in \mathcal{V}$ whose domain is 2^N, where N is a set of objects. There does not exist a solution $\langle x_1, ..., x_n \rangle$ such that any pair of $\tau = x_i$ and $\tau' = x_j$ for $1 \leq i < j \leq n$ satisfy either of the following* *conditions:*

1. *There exists $k \in N$ such that $(U, U \setminus \{k\}) \in \mathsf{eff}_\tau$, $(k \in U) \in \mathsf{pre}_{\tau'}$, and each $\tau'' \in \mathcal{T} \setminus \{x_i, x_j\}$ does not change U or $(U, U \setminus \{l\}) \in \mathsf{eff}_{\tau''}$ for some $l \in N$.*
2. *There exists $k \in N$ such that $(U, U \cup \{k\}) \in \mathsf{eff}_\tau$, $(k \notin U) \in \mathsf{pre}_{\tau'}$, and each $\tau'' \in \mathcal{T} \setminus \{x_i, x_j\}$ does not change U or $(U, U \cup \{l\}) \in \mathsf{eff}_{\tau''}$ for some $l \in N$.*

Proof. For the first case, x_i removes k from U, and no other transition adds k to U. Since x_j requires k to be in U, once we apply x_i, we cannot apply x_j later. The other case is proved similarly. ◀

Before starting beam search in a neighborhood, we remove a transition τ from the model if there exists a transition τ' in the suffix such that τ and τ' satisfy one of the conditions in Proposition 2. Detecting such a pair of transitions is done once at the beginning by checking the expression trees representing the preconditions and effects of transitions.

3.3 Bandit-Based Depth Selection

Selecting the depth of a neighborhood, d, in line 3 of Algorithm 3 is non-trivial. If d is too small, it is unlikely that an improving solution exists. However, if d is too large, each neighborhood search takes a long time. We want to select d such that the total cost improvement is maximized within the time limit.

We formulate the depth selection as the budgeted multi-armed bandit problem with continuous random costs [35]. We have the set of depths $D \subseteq \{2, ..., n\}$. If we select a depth $d \in D$ and perform search in line 7 of Algorithm 3, we obtain a new solution x by spending search time t. As t is assumed to take a value in $[0, 1]$ in the budgeted multi-armed bandit problem, we divide the actual time by the time limit T. If the cost $\mathsf{cost}_x(S^0)$ is smaller than the current best solution cost $\overline{f}$, the reward is $r = (\overline{f} - \mathsf{cost}_x(S^0))/\overline{f}$. If no better solution is found, the reward is $r = 0$. We call this process a *round*, and we repeat rounds until reaching the time limit T. We do not know the reward r and time t before finishing a round, so we use random variables r_{dk} and t_{dk} representing the reward and time if depth d is used at round k. Let a be a strategy that selects a depth a_k in the round k. The number of rounds performed by a by the time limit, K_{aT}, is also a random variable. The objective is to find a strategy a that maximizes the total expected reward $\mathbb{E}[\sum_{k=1}^{K_{aT}} r_{a_k k}]$.

We use Budgeted-UCB [35]. At each round, if some depths in D have not been selected before, Budgeted-UCB selects one of them. Otherwise, let m_{dk} be the number of rounds where the depth d is selected up to round $k-1$, and let $\bar{r}_{dk}$ and $\bar{t}_{dk}$ be the average reward and search time for d up to round $k-1$. Budgeted-UCB selects the depth d that maximizes

$$\frac{\bar{r}_{dk}}{\bar{t}_{dk}} + \frac{\epsilon_{dk}}{\bar{t}_{dk}} + \frac{\epsilon_{dk}}{\bar{t}_{dk}} \frac{\min\{\bar{r}_{dk} + \epsilon_{dk}, 1\}}{\max\{\bar{t}_{dk} - \epsilon_{dk}, \lambda\}} \tag{5}$$

where $\epsilon_{dk} = \sqrt{\frac{2\log(k-1)}{m_{dk}}}$ and λ is a positive lower bound of the search time of each round.

In practice, we initialize $D = \{2, 4, 8, ..., 2^l, n\}$, where n is the length of the initial feasible solution and l is the maximum integer such that $2^l < n$. If we get a solution whose length n' is different from n at round k, we replace n with n' in D using $m_{n',k+1} = m_{n,k+1}$, $\bar{r}_{n',k+1} = \bar{r}_{n,k+1}$, and $\bar{t}_{n',k+1} = \bar{t}_{n,k+1}$ and ignore depths greater than n' in D. If multiple depths have not been selected before or have the same value, we select the minimum depth among them. For λ, we use the time of the first round divided by 10 while there is no guarantee that it is a lower bound. Thus, the theoretical analysis of Budgeted-UCB studied in the original paper [35] does not necessarily apply to our setting. In addition, while Xia et al. [35] assumed that the pairs $\{(r_{dk}, t_{dk})\}_{k=1}^{\infty}$ are i.i.d., we do not have such a guarantee.

3.4 Start Selection

Once LNBS determines the depth d to use, it selects a starting point i, which induces the prefix and the suffix, in line 4 of Algorithm 3. We select i considering the cost change in a neighborhood. Concretely, the cost change by partial path $\langle x_i, ..., x_{i+d-1} \rangle$ is defined as

$$\delta_{di} = \mathsf{cost}_{\langle x_1,...,x_{i+d-1} \rangle}(S^0) - \mathsf{cost}_{\langle x_1,...,x_{i-1} \rangle}(S^0). \tag{6}$$

Since the path cost is non-decreasing, $\delta_{di} \geq 0$. We ignore i with $\delta_{di} = 0$[1] and select one uniformly at random from the remaining options.

Our second approach is to select i based on the probability biased by δ_{di}. As we explain in the next subsection, for each d and i, the beam width b is maintained. Since smaller b_{di} leads to a shorter search time, we discount the probability of selecting i by b_{di}. Concretely, given the depth d, we can select the starting point i with the probability

$$p_{di} = \frac{\delta_{di}/b_{di}}{\sum_{j=1}^{n-d+1} \delta_{dj}/b_{dj}}. \tag{7}$$

3.5 Beam Width Selection

Given the depth d and the starting point i, LNBS selects a beam width b in line 5 of Algorithm 3. Here, we use a similar strategy to CABS: for each d and i, we initialize the beam width b_{di} to be 1 and update it to $2b_{di}$ after each round with d and i. If we find an improved solution in line 7, we reset $b_{d'i'} = 1$ only for d' and i' such that $i' > i$ or $i' + d' < i + d$; if $i' \leq i$ and $i' + d' \geq i + d$, the prefix and the suffix for the neighborhood induced by i' and d' do not change, and we know that a better partial path was not found with beam widths smaller than $b_{d'i'}$.[2] If the neighborhood is exhausted, i.e., $\mathsf{complete} = \top$ in line 7, we ignore the combination of d and i in lines 3 and 4 until a new solution is found and b_{di} is reset to 1. Since the number of neighborhoods is finite, LNBS eventually exhausts all the neighborhoods and finds the optimal solution, which guarantees completeness.

4 Experimental Evaluation

We compare LNBS with CABS, DD-LNS, CP, and mixed-integer programming (MIP).

4.1 Experimental Settings

As benchmarks, we use the same problems and instances used by previous work [26]: the traveling salesperson problem with time windows (TSPTW), the capacitated vehicle routing problem (CVRP), the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP), the single machine total weighted tardiness problem ($1||\sum w_i T_i$), the talent scheduling problem (Talent), the simple assembly line balancing problem to minimize the number of stations (SALBP-1), the bin packing problem (BPP), and the graph-clear problem (GCP). We use the same DP, CP, and MIP models as the previous work [26].[3]

[1] Theoretically, a better solution may be found with such i if the suffix is not empty because a partial path may change the state from which the suffix is applied, which may change the cost of the suffix.

[2] Theoretically, a better solution may be found with beam width smaller than $b_{d'i'}$ if the updated primal bound changes the search behavior.

[3] `https://github.com/Kurororoko/didp-models`

LNBS, CABS, and DD-LNS are implemented in didp-rs v0.3.2[4] using Rust 1.65.0. In LNBS and DD-LNS, CABS is run first to find a feasible solution, and then LNBS and DD-LNS are run to improve the solution. For DD-LNS, we use $W = 1000$ and $p = 0.1$ following the original paper [18]. As we described above, LNBS removes conflicting transitions from a suffix, selects the depth using Budgeted-UCB, and geometrically increases the beam width for each neighborhood. To select the starting point of a partial path, we consider two approaches, uniform and biased sampling. While biased sampling achieves better solution quality in CVRP and m-PDTSP, uniform sampling is better in TSPTW, $1||\sum w_i T_i$, SALBP-1, MOSP, and GCP (see Appendix A). In what follows, we only show results from uniform sampling. In Appendix A, we also evaluate the importance of removing conflicting transitions and Budgeted-UCB. We confirm that these mechanisms significantly improve the solution quality. A more comprehensive ablation study is left for future work.

We implemented the DP models using didppy, a Python interface for didp-rs. We use IBM ILOG CP Optimizer 22.1.0 for the CP models and Gurobi Optimizer 9.5.0 for the MIP models. CP Optimizer is known to use LNS [27]. The DP, CP, and MIP models are implemented in Python 3.10.2. All experiments are run on an Intel Xeon Gold 6148 processor with a single thread, an 8 GB memory limit, and a time limit of 1800 seconds. For LNBS and DD-LNS, we take the median of 5 runs.

Following the previous work [26], we use the primal gap and the primal integral to measure the performance [7]. If an algorithm finds a solution x^t with the cost $f(x^t)$ at time t, and the optimal or best-known solution cost is f^*, the primal gap at t for the algorithm is

$$p(t) = \begin{cases} 1 & \text{if } f(x^t) \cdot f^* < 0 \\ 0 & \text{if } f(x^t) = f^* = 0 \\ \frac{|f(x^t) - f^*|}{\max\{|f(x^t)|, |f^*|\}} & \text{otherwise.} \end{cases} \tag{8}$$

If the algorithm does not find a solution at time t, then $p(t) = 1$. Let $t_1, ..., t_{l-1}$ be time points where a better solution is found, $t_0 = 0$, and $t_l = T$ where T is the time limit. Then, the primal integral, $P(T)$ is defined as

$$P(T) = \sum_{i=1}^{l} p(t_{i-1}) \cdot (t_i - t_{i-1}). \tag{9}$$

We use the primal gap at the time limit, $p(T)$, and the primal integral, $P(T)$, as the measures.

4.2 Experimental Results

We show the number of instances where the optimality of a solution is proved, the average primal gap at the time limit, and the average primal integral for each problem in Table 1. We omit MIP in Table 1 because it is outperformed by CP in the primal gap and the primal integral for all problem types (see Appendix A). In the number of optimally solved instances, MIP is the best in CVRP (with 26 problems solved). As reported by previous work [26], MIP is good at finding an optimal solution for small instances, while it fails to find a feasible solution for larger instances, which results in poor performance in the primal gap and the primal integral on average. In other problems, LNBS solves more instances optimally than MIP except for BPP, where LNBS solves 1139 and MIP solves 1157.

[4] `https://didp.ai`

Table 1 Summary of the experimental result. “#” is the number of optimally solved instances, “gap” is the average primal gap at the time limit, and “p.i.” is the average primal integral.

	CP			CABS			DD-LNS			LNBS		
	#	gap	p.i.	#	gap	p.i.	#	gap	p.i.	#	gap	p.i.
TSPTW (340)	47	0.0259	49.0	**257**	0.0033	9.0	109	0.0100	23.7	241	**0.0016**	**5.7**
CVRP (207)	0	0.3174	601.1	**6**	0.1772	339.5	0	0.2504	461.6	**6**	**0.1640**	**316.8**
m-PDTSP (1178)	**1049**	0.0122	25.5	1030	0.0023	5.1	459	0.0102	21.7	1029	**0.0022**	**5.0**
$1\|\|\sum w_iT_i$ (375)	150	**0.0009**	**3.5**	**286**	0.0346	74.4	100	0.0409	83.5	275	0.0051	13.0
Talent (1000)	0	0.0081	29.3	**232**	0.0173	38.2	0	0.0602	114.6	**232**	**0.0041**	**11.1**
SALBP-1 (2100)	1584	0.0046	28.4	**1799**	**0.0003**	**2.2**	1507	0.0067	13.5	1682	0.0022	7.3
BPP (1615)	**1234**	**0.0014**	7.7	1159	0.0017	**6.1**	775	0.0190	35.4	1139	0.0021	8.1
MOSP (570)	437	0.0044	13.0	**526**	**0.0000**	**0.4**	353	0.0203	37.5	523	0.0002	0.7
GCP (135)	1	0.0151	44.3	**103**	**0.0000**	**0.6**	3	0.0009	2.7	102	0.0001	**0.6**
Larger Instances												
m-PDTSP (240)	77	0.1491	285.9	**101**	0.0694	153.2	79	0.1345	265.7	98	**0.0652**	**146.6**
MOSP (760)	0	0.0676	150.6	**150**	**0.0002**	**4.4**	0	0.0402	72.7	148	0.0025	10.4
GCP (50)	**0**	0.5289	1268.3	**0**	**0.0013**	**10.8**	**0**	0.0764	137.8	**0**	0.0038	19.5

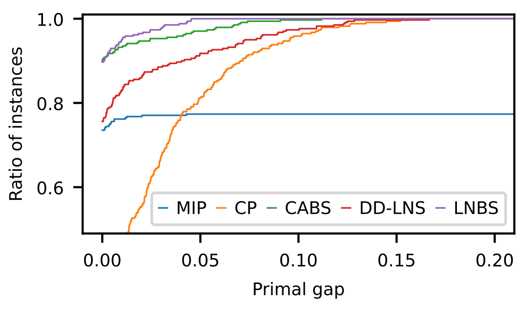
Compared to CABS, LNBS achieves the better primal gap and the primal integral in TSPTW, CVRP, m-PDTSP, $1||\sum w_iT_i$, and Talent. We show the distribution of the primal gap in TSPTW, CVRP, $1||\sum w_iT_i$, and Talent in Figure 2. The primal integral has a similar trend to the primal gap in these problems (see Appendix A). In contrast, CABS is better than LNBS in SALBP-1, BPP, MOSP, and GCP. These results are consistent with the observation that LNS is effective for routing and scheduling problems in CP [24, 27]. In the routing problems (TSPTW, CVRP, and m-PDTSP), CABS is already better than CP, and LNBS is even better than CABS. In $1||\sum w_iT_i$, while LNBS shows a significant improvement from CABS (0.0051 from 0.0346) outperforming MIP (0.0188), CP is still the best. However, in Talent, LNBS outperforms CP while CABS does not. Overall, LNBS is better than CP in seven problems while CABS is better than CP in six problems.

In TSPTW, the difference in the primal gap between LNBS and CABS (0.0016 and 0.0033) seems small, but it is because they achieve almost the same primal gap in many instances: they optimally solve all 135 instances in the Dumas set [10] and achieve almost the same average primal gap in the AFG set [2], which has 50 instances, and GendreauDumas Extended set [16], which has 130 instances. However, in the OhlmannThomas set [31], which has 25 instances, no instance is optimally solved, and LNBS shows a significant improvement in the primal gap (0.0184 from 0.0395).

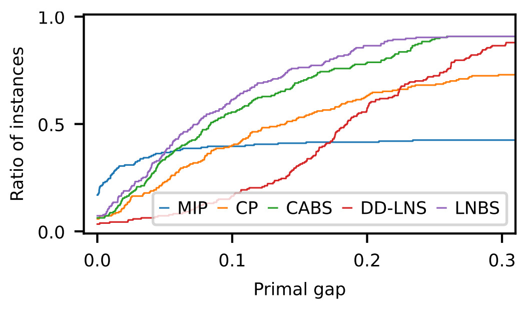
In the number of optimally solved instances, CABS is equal to or better than LNBS in all problems. While CABS always searches the entire state space graph, LNBS searches multiple neighborhoods, and the entire state space graph is just one of them. Nevertheless, LNBS proves the optimality of more than 93% of instances that are optimally solved by CABS.

DD-LNS performs worse than CABS and LNBS in all the problems. Previous work has reported that DD-LNS is effective for TSPTW [18]. Note however that the DD-LNS and LNBS results in Table 1 are not the results reported by Gillard and Schaus. To validate that our implementation and experimental settings do not handicap DD-LNS, we compare the results of DD-LNS with those of the original paper in TSPTW.[5] Our DD-LNS implementation

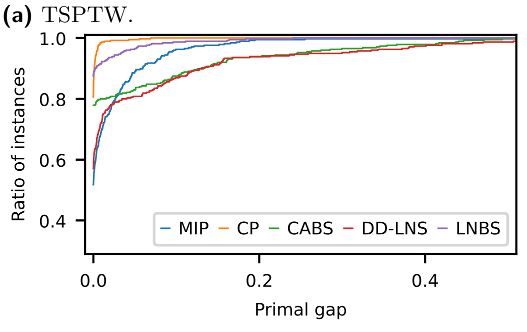
[5] `https://github.com/xgillard/ijcai_22_DDLNS/blob/main/results/tsptw/ddlns/results_w1000_`



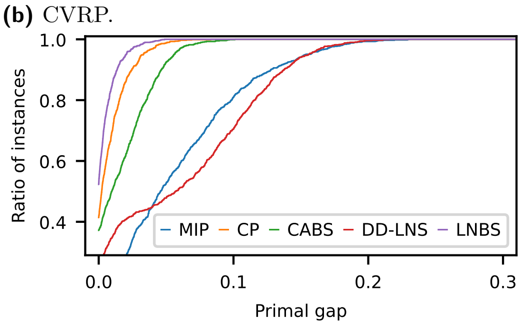
(a) TSPTW.



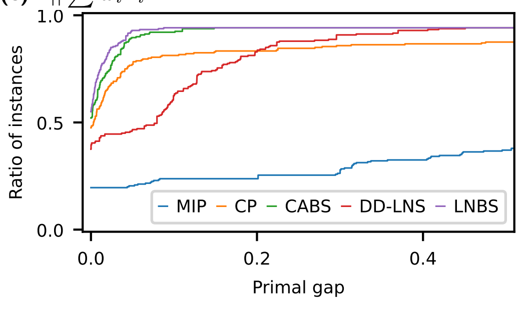
(b) CVRP.



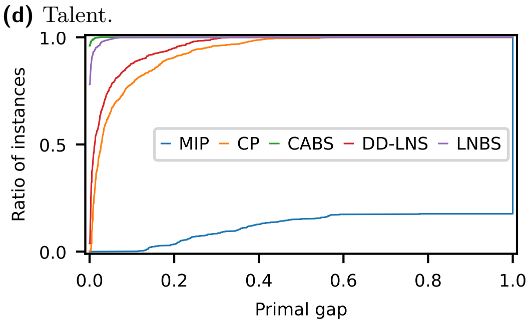
(c) $1||\sum w_i T_i$.



(d) Talent.



(e) Large instances in m-PDTSP.



(f) Large instances in MOSP.

Figure 2 Distribution of the primal gap at the time limit. Higher and left is better.

finds a better solution than the original in all instances with the time limit of 600 seconds. This difference is likely due to the difference between the DP models used by us (from Kuroiwa and Beck [26]) and Gillard and Schaus. In TSPTW, a solution is a tour that starts from a depot, visits each customer j within the time window $[a_j, b_j]$, and returns to the depot, and an optimal solution minimizes the total travel time. In both DP models, state variables are the set of unvisited customers U, the current customer i, and the current time t, and each transition corresponds to visiting a customer or the depot. Each DP model has a dual bound (called RLB by Gillard and Schaus), a lower bound on the optimal solution cost. While Gillard and Schaus use a dual bound based on a minimum spanning tree, Kuroiwa and Beck use a simpler one based on the minimum travel time between customers. In addition, Kuroiwa and Beck use information that was not considered by Gillard and Schaus. First, they use dominance between states based on the current time: a state S dominates another state S' if $S[U] = S'[U]$, $S[i] = S'[i]$, and $S[t] \leq S'[t]$. Furthermore, since the time to visit customer j is underestimated by $t + c^*_{ij}$, where c^*_{ij} is the shortest travel time from i to j, they

`t600.txt`

define state constraints $\forall j \in U, t + c^*_{ij} \leq b_j$. The dominance and the state constraints are useful to prune states, which potentially explains the performance gap. Another difference is whether considering the time window constraints at the depot or not. In the benchmark instances used above[6] (and in Kuroiwa and Beck [26]), a time window $[a_0, b_0]$ is defined for the depot. Gillard and Schaus explicitly model a required return to the depot within $[a_0, b_0]$ while Kuroiwa and Beck do not. However, in the benchmark instances, $b_0 \geq b_j + c_{j0}$ holds for all customers j, where c_{j0} is the travel time from j to the depot. Thus, if all customers are visited within the time windows, the depot can be reached within the time window, and explicit modeling of the depot return window is unnecessary.

4.3 Larger Instances

LNBS performs better than CABS in m-PDTSP and worse in MOSP and GCP, but the difference in the average primal gap is small. To evaluate the difference more clearly, we use larger instances for these problems.

In m-PDTSP, a vehicle visits all nodes in a graph, picks up some commodities at some nodes, and delivers them to others. Each commodity has a weight and the total weight of commodities that a vehicle can carry is limited by the capacity. In the benchmark set for m-PDTSP, three types of instances are used: Class 1, Class 2, and Class 3 [21], and Class 1 instances are generated from instances of the sequential ordering problem (SOP) [3]. We generate larger Class 1 instances by using 30 SOP instances in TSPLIB[7] that were not used by the previous work. The original instances have at most 47 nodes, and the new instances have 42 to 378 nodes. We use the same methods as the previous work [21] with the maximum weight $q \in \{1, 5\}$ and the capacity $Q \in \{5q, 10q, 20q, 100q\}$, resulting in 240 instance in total.

In MOSP, an instance is represented by a matrix, and the original set uses at most 125×125 matrices. We add instances using 150×150 to 1000×1000 matrices [8, 12].

In GCP, an instance is represented by a graph. The original instance set uses random and random planar graphs with 20, 30, and 40 nodes. We generate 50 instances using random graphs with 100 and 200 nodes following the method used by previous work [29].

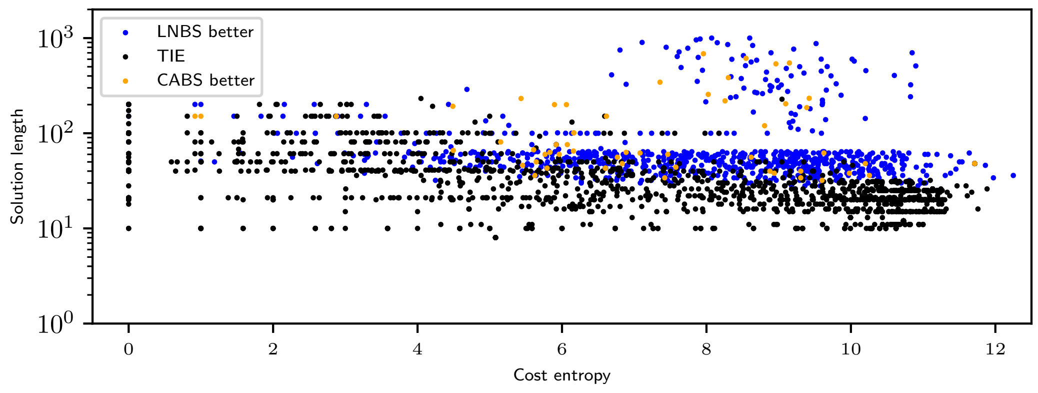
As shown in Table 1, LNBS clearly outperforms CABS in m-PDTSP in the primal gap and the primal integral, but CABS is better in MOSP and GCP. We also show the distribution of the primal gap in m-PDTSP and MOSP in Figure 2. The primal integral has a similar tendency to the primal gap, and the result for GCP is qualitatively similar to that of MOSP.

4.4 Analysis of Problem Characteristics

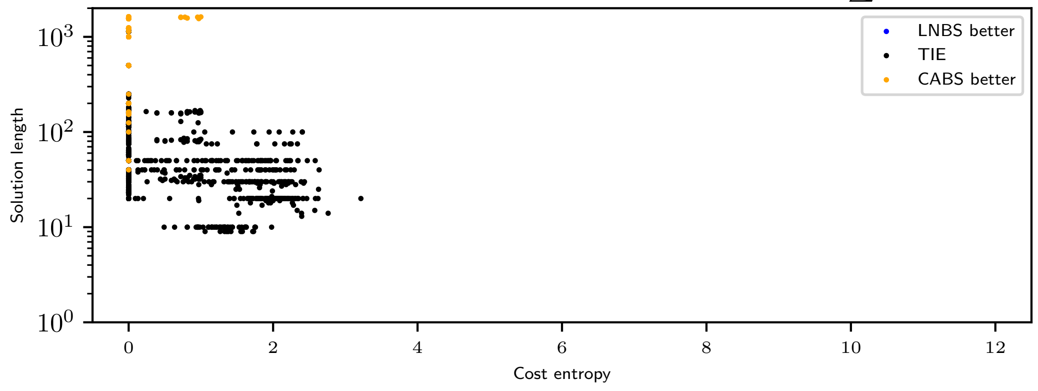
In routing problems, a solution is a route visiting all nodes in a graph, and its cost is the length of the route. In the DP models for these problems, each transition corresponds to visiting one node, and the cost of a partial path increases when a transition is applied. We expect that different partial solutions tend to have different costs, and it is relatively easy to find a better partial path; because the path costs are diverse, unless the current partial path is optimal, better partial paths are included in a partial state space graph with high density. In such a case, beam search is likely to find a better partial path although it searches in a fraction of the partial state space graph restricted by the beam width.

In contrast, in SALBP-1 and BPP, the problem is to pack weighted items into capacitated bins while minimizing the number of bins. In the DP models, each transition packs one item into a bin, and the cost increases only when a new bin is opened. In the DP models for

[6] `https://lopez-ibanez.eu/tsptw-instances`
[7] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/sop/`



(a) Problem types where LNBS has lower mean gap: TSPTW, CVRP, m-PDTSP, $1||\sum w_i T_i$, and Talent.



(b) Problem types where CABS has lower mean gap: SALBP-1, BPP, MOSP, and GCP.

Figure 3 Entropy of the cost distribution over partial paths vs. the solution length in each problem instance. 'LNBS better' means LNBS finds a better solution, 'TIE' means that LNBS and CABS achieve the same solution cost, and 'CABS better' means that CABS finds a better solution.

MOSP and GCP, the cost is computed by taking the maximum weights of edges in a path, and it does not increase unless a new edge has a higher weight than the current maximum. Therefore, we expect that many partial paths tend to have the same cost in these problems, making it difficult to improve a solution by searching only a partial state space graph.

Based on these observations, we hypothesize that LNBS tends to perform better than CABS when path costs in a partial state space graph are diverse. To test this hypothesis, we evaluate the diversity of costs in a partial state space graph using entropy in information theory. Given a solution for a DyPDL model $x = \langle x_1, ..., x_n \rangle$, let $Y_{di}(x)$ be the set of solution paths whose prefix is $\langle x_1, ..., x_{i-1} \rangle$ and the suffix is $\langle x_{i+d}, ..., x_n \rangle$. Let $C = \{\mathsf{cost}_y(S^0) \mid y \in Y_{di}(x)\}$ be the set of the path costs. Then, the entropy of the path costs is defined as follows:

$$H(Y_{di}(x)) = -\sum_{c \in C} \frac{|\{y \in Y_{di}(x) \mid \mathsf{cost}_y(S^0) = c\}|}{|Y_{di}(x)|} \log_2 \frac{|\{y \in Y_{di}(x) \mid \mathsf{cost}_y(S^0) = c\}|}{|Y_{di}(x)|}. \quad (10)$$

As this value gets larger, the cost distribution becomes more diverse, and we expect that LNBS will perform better than CABS. However, even if the entropy is large, if the problem itself is easy, both CABS and LNBS will find optimal or near-optimal solutions. To consider such cases, we also evaluate the length of the initial solution found by CABS.

We evaluate entropy and the length of the initial solution found for each problem instance used in Section 4.2. We first run CABS until it finds a feasible solution and record the length of the solution. Then, we remove the first eight transitions from the solution and enumerate all feasible prefixes of the solution, i.e., we use $d = 8$ and $i = 1$. In Figure 3, we show a scatter plot of entropy and the solution length divided into two plots to emphasize the differences between the problem types where LNBS has a lower primal gap on average (Figure 3a) and those where CABS is better (Figure 3b). With low entropy, CABS dominates. For higher entropy, the solution length begins to play a factor: for short solutions, CABS and LNBS perform equally but for longer solutions, LNBS tends to perform better. Indeed, for the problems where CABS performs better on average (Figure 3b), the entropy is quite low (less than 3.5). This result suggests that the entropy of the cost distribution over partial paths is related to the performance of LNBS. Since this analysis is based on path costs, it is not directly applicable to LNS with tree search, where a solution corresponds to a leaf node. However, if we consider the factors of neighborhood size and cost distribution over leaf nodes, we may be able to apply this analysis to LNS for CP and MIP.

5 Related Work

As we discussed, LNBS can be considered a generalization of DD-LNS [18], which combines DDs and LNS. DP is closely related to DDs [23], and DDs have been actively used for combinatorial optimization [9]. For example, DDs are used to obtain bounds on the optimal objective value [4, 33], and heuristics based on DDs have been proposed [6]. Moreover, ddo, a general-purpose DD solver for combinatorial optimization has been developed [5, 19]. In CP, DDs are used for constraint propagation [1, 22]. Recently, HADDOCK, a modeling language of a DD based on a state transition system, was proposed for CP [17].

In state space search, there exist several methods that improve a solution path by searching in a partial state space graph, but they were not framed as LNS. In classical planning, plan neighborhood graph search (PNGS) first constructs a partial state space graph by performing local search from each state in a solution path and then finds the shortest path in the graph [30]. In sliding tile puzzles, iterative tunneling search with A* (ITSA*) iteratively expands a partial state space graph, which includes states close to a given path, and finds the shortest path in that graph [13]. Unlike the above two algorithms, Joint and local path A* (LPA*) [32] try to find a better partial path between two states in a given path using A* [20]. While Joint and LPA* fix the length of a partial path to remove and deterministically select a neighborhood, LNBS dynamically adjusts them and uses beam search instead of A*.

6 Conclusion

We proposed large neighborhood beam search (LNBS), a state space search algorithm based on large neighborhood search (LNS) and beam search for domain-independent dynamic programming (DIDP). Our configuration of LNBS exploits the multi-armed bandit problem and random sampling to select a neighborhood. We proved that LNBS is complete. LNBS finds better quality solutions on average than the state-of-the-art DIDP solver, complete anytime beam search (CABS), in five out of the nine benchmark problems. In particular, LNBS performs well in routing and scheduling problems, and our analysis suggests that this performance is related to the diversity of the cost distribution over partial paths. A deeper investigation of the characteristics of the problems that make LNS effective in state space search and tree search is an interesting direction for future work. Based on such analysis, developing better configurations for LNBS may also be possible.

References

1 H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming – CP 2007*, pages 118–132, 2007. doi:10.1007/978-3-540-74970-7_11.

2 Norbert Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, Technische Universität Berlin, 1995.

3 Norbert Ascheuer, Michael Jünger, and Gerhard Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraint. *Computational Optimization and Applications*, 17:25–42, 2000. doi:10.1023/A:1008779125567.

4 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014. doi:10.1287/ijoc.2013.0561.

5 David Bergman, Andre A. Cire, Willem Jan Van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, December 2016. doi:10.1287/ijoc.2015.0648.

6 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and Tallys Yunes. Bdd-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014. doi:10.1007/s10732-014-9238-1.

7 Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41:611–614, 2013. doi:10.1016/j.orl.2013.08.007.

8 Marco Antonio Moreira De Carvalho and Nei Yoshihiro Soma. A breadth-first search applied to the minimization of the open stacks. *Journal of the Operational Research Society*, 66:936–946, June 2015. doi:10.1057/jors.2014.60.

9 Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. Decision diagrams for discrete optimization: A survey of recent advances. *INFORMS Journal on Computing*, 34(4):2271–2295, 2022. doi:10.1287/ijoc.2022.1170.

10 Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995. doi:10.1287/opre.43.2.367.

11 Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch Lafuente. Cost-algebraic heuristic search. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pages 1362–1367, 2005.

12 Rafael de Magalhães Dias Frinhani, Marco Antonio Moreira de Carvalho, and Nei Yoshihiro Soma. A pagerank-based heuristic for the minimization of open stacks problem. *PLoS ONE*, 13(8):1–24, 2018. doi:10.1371/journal.pone.0203076.

13 David A Furcy. ITSA*: Iterative tunneling search with A*. In *Proceedings of AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, pages 21–26, 2006.

14 Maria Garcia de la Banda and Peter J. Stuckey. Dynamic programming to minimize the maximum number of open stacks. *INFORMS Journal on Computing*, 19(4):607–617, 2007. doi:10.1287/ijoc.1060.0205.

15 Maria Garcia de la Banda, Peter J. Stuckey, and Geoffrey Chu. Solving talent scheduling with dynamic programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011. doi:10.1287/ijoc.1090.0378.

16 Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–346, 1998. doi:10.1287/opre.46.3.330.

17 Rebecca Gentzel, Laurent Michel, and W.-J. van Hoeve. HADDOCK: A language and architecture for decision diagram compilation. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming – CP 2020*, pages 531–547, 2020. doi:10.1007/978-3-030-58475-7_31.

18 Xavier Gillard and Pierre Schaus. Large neighborhood search with decision diagrams. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 4754–4760, 2022. doi:10.24963/ijcai.2022/659.

19 Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5243–5245, 2020. doi:10.24963/ijcai.2020/757.

20 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

21 Hipólito Hernández-Pérez and Juan José Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196:987–995, August 2009. doi:10.1016/j.ejor.2008.05.009.

22 Samid Hoda, Willem-Jan van Hoeve, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *Principles and Practice of Constraint Programming – CP 2010*, pages 266–280, 2010.

23 John N. Hooker. Decision diagrams and dynamic programming. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–110, 2013.

24 Siddhartha Jain and Pascal Van Hentenryck. Large neighborhood search for dial-a-ride problems. In *Principles and Practice of Constraint Programming – CP 2011*, pages 400–413, 2011.

25 Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023. doi:10.1609/icaps.v33i1.27200.

26 Ryo Kuroiwa and J. Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023. doi:10.1609/icaps.v33i1.27201.

27 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018. doi:10.1007/s10601-018-9281-x.

28 Shu Lin, Na Meng, and Wenxin Li. Optimizing constraint solving via dynamic programming. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1146–1154, July 2019. doi:10.24963/ijcai.2019/160.

29 Michael Morin, Margarita P. Castro, Kyle E.C. Booth, Tony T. Tran, Chang Liu, and J. Christopher Beck. Intruder alert! Optimization models for solving the mobile robot graph-clear problem. *Constraints*, 23(3):335–354, 2018. doi:10.1007/s10601-018-9288-3.

30 Hootan Nakhost and Martin Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 121–128, 2010. doi:10.1609/icaps.v20i1.13402.

31 Jeffrey W. Ohlmann and Barrett W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1):80–90, 2007. doi:10.1287/ijoc.1050.0145.

32 Daniel Ratner and Ira Pohl. Joint and LPA*: Combination of approximation and search. In *Proceedings of the fifith National Conference on Artificial Intelligence (AAAI).*, pages 173–177, 1986. doi:10.5555/2887770.2887798.

33 Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Peel-And-Bound: Generating Stronger Relaxed Bounds with Multivalued Decision Diagrams. In *Principles and Practice of Constraint Programming – CP 2022*, pages 35:1–35:20, 2022. doi:10.4230/LIPIcs.CP.2022.35.

34 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming – CP98*, volume 1520, pages 417–431, 1998. `doi:10.1007/3-540-49481-2_30`.

35 Yingce Xia, Xu-Dong Zhang, Nenghai Yu, Geoffrey Holmes, and Yan Liu. Budgeted bandit problems with continuous random costs. In *Proceedings of the Seventh Asian Conference on Machine Learning*, pages 317–332, 2015.

36 Weixiong Zhang. Complete anytime beam search. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI-98/IAAI-98)*, pages 425–430, 1998.

A Additional Experimental Results

Table 2 Comparison of CP, MIP, and two configurations of LNBS. "#" is the number of optimally solved instances, "gap" is the average primal gap at the time limit, and "p.i." is the average primal integral.

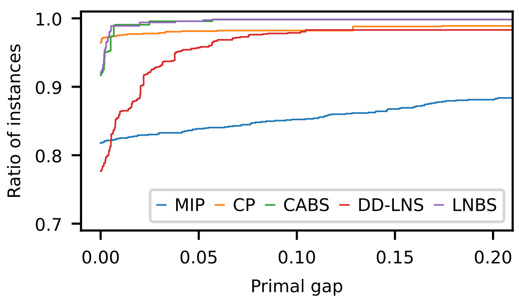
	CP			MIP			LNBS			LNBS/bias		
	#	gap	p.i.	#	gap.	p.i.	#	gap.	p.i.	#	gap.	p.i.
TSPTW (340)	46	0.0275	52.3	227	0.2268	484.1	241	**0.0016**	**5.7**	**242**	0.0019	6.2
CVRP (207)	0	0.3174	601.1	**26**	0.5845	1157.4	6	0.1640	316.8	6	**0.1633**	**314.8**
m-PDTSP (1178)	**1050**	0.0121	25.4	945	0.0858	180.0	1029	0.0022	5.0	1029	**0.0021**	**4.7**
$1\|\|\sum w_iT_i$ (375)	150	**0.0003**	**2.4**	109	0.0188	75.6	**275**	0.0051	13.0	**275**	0.0056	14.5
Talent (1000)	7	0.0072	27.6	0	0.0573	152.6	**232**	**0.0041**	**11.1**	231	0.0042	**11.1**
SALBP-1 (2100)	1584	0.0046	28.4	1357	0.3447	634.6	**1682**	**0.0022**	**7.3**	1675	**0.0022**	7.5
BPP (1615)	**1234**	**0.0014**	**7.7**	1157	0.0385	85.9	1139	0.0021	8.1	1129	0.0021	9.0
MOSP (570)	437	0.0044	13.0	224	0.0394	100.4	**523**	**0.0002**	**0.7**	**523**	**0.0002**	**0.7**
GCP (135)	1	0.0151	44.3	23	0.1102	311.9	**102**	**0.0001**	**0.6**	**102**	**0.0001**	0.7
Larger Instances												
m-PDTSP (240)	77	0.1481	284.1	47	0.5811	1096.8	**98**	0.0652	**146.6**	97	**0.0647**	147.0
MOSP (760)	0	0.0675	150.4	0	0.8806	1599.4	**148**	**0.0025**	**10.4**	**148**	0.0027	10.7
GCP (50)	**0**	0.5287	1268.1	**0**	0.5306	977.8	**0**	**0.0038**	**19.5**	**0**	0.0061	21.2

Table 3 Comparison of LNBS variants. "No removing conflicts" does not remove conflicting transitions in the suffix. "No Budgeted-UCB" selects the depth uniformly at random instead of using Budgeted-UCB. "#' is the number of optimally solved instances, "gap" is the average primal gap at the time limit, and "p.i." is the average primal integral.

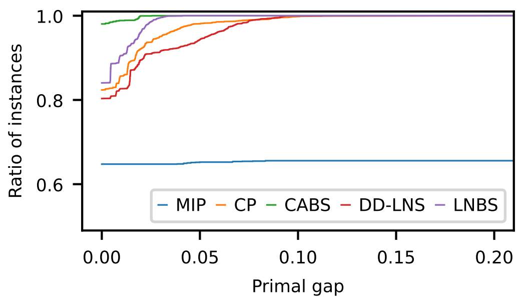
	LNBS			No removing conflicts			No Budgeted-UCB		
	#	gap.	p.i.	#	gap.	p.i.	#	gap.	p.i.
TSPTW (340)	241	**0.0016**	**5.7**	234	0.0035	10.6	**256**	0.0034	9.4
CVRP (207)	**6**	**0.1640**	**316.8**	5	0.1682	322.5	**6**	0.1767	338.7
$1\|\|\sum w_iT_i$ (375)	275	**0.0051**	**13.0**	268	0.0224	56.0	**287**	0.0340	74.1

We show the result of MIP in Table 2. MIP solves more instances than CP in TSPTW, CVRP, and GCP, but CP is better in the primal gap and the primal integral.

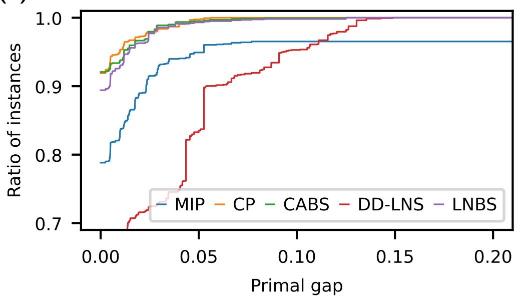
In addition, we compare two configurations of LNBS in Table 2. One selects the starting point of a partial path uniformly at random (LNBS), and another selects it according to the probability distribution biased by partial path costs in Equation (7) (LNBS/bias). LNBS/bias solves one more instance in TSPTW and outperforms LNBS in CVRP and m-PDTSP in the primal gap.



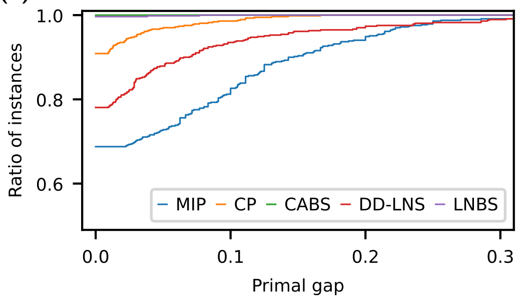
(a) m-PDTSP.



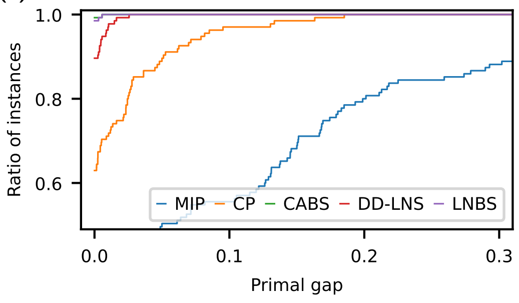
(b) SALBP-1.



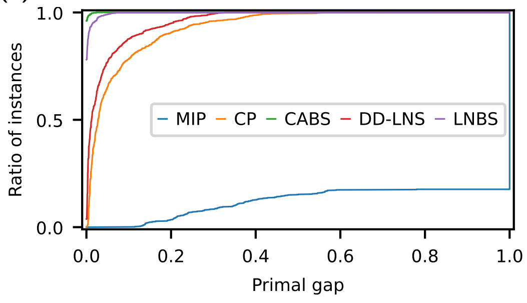
(c) BPP.



(d) MOSP.



(e) GCP.

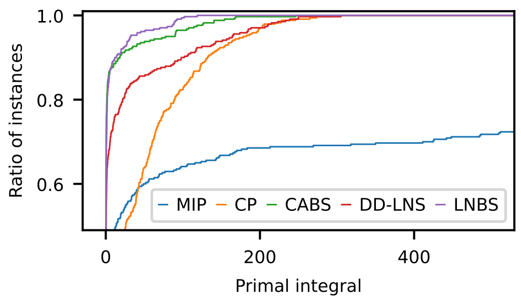


(f) Large instances in GCP.

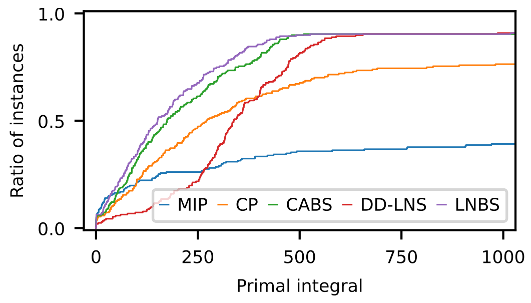
Figure 4 Distribution of the primal gap at the time limit. Higher and left is better.

We evaluate the importance of other components of LNBS using a subset of the problems: TSPTW, CVRP, and $1||\sum w_i T_i$. In Table 3, we compare two variants of LNBS, where conflicting transitions in a suffix are not removed (‘No removing conflicts’), and the depth is selected uniformly at random instead of Budgeted-UCB (‘No Budgeted-UCB’). The two variants perform worse in terms of the primal gap and the primal integral. While ‘No Budgeted-UCB’ solves more instances to optimality, it is because the largest depth, which makes LNBS the same as CABS, is more likely to be selected by uniform sampling. Indeed, the primal gap and the primal integral of ‘No Budgeted-UCB’ are close to those of CABS.

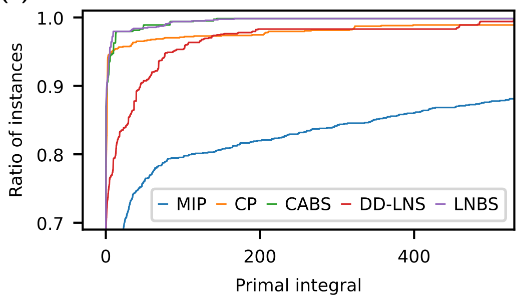
In Figure 4, we present the distribution of the primal gap over instances in m-PDTSP, SALBP-1, BPP, MOSP, Graph-Clear, and large instances in GCP. LNBS is slightly better in m-PDTSP, and CABS is better in SALBP-1, BPP, and large instances in GCP. Figures 5 and 6 show the distribution of the primal integral. The tendency is similar to that of the primal gap.



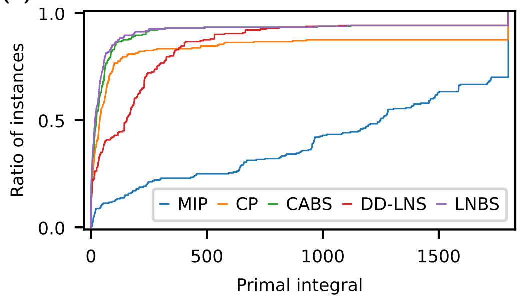
(a) TSPTW.



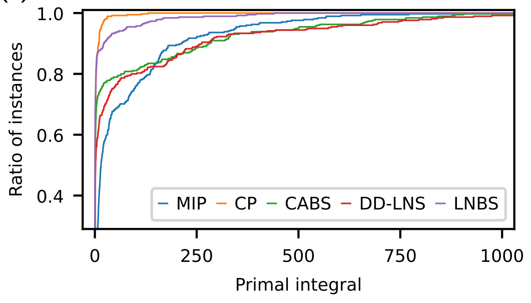
(b) CVRP.



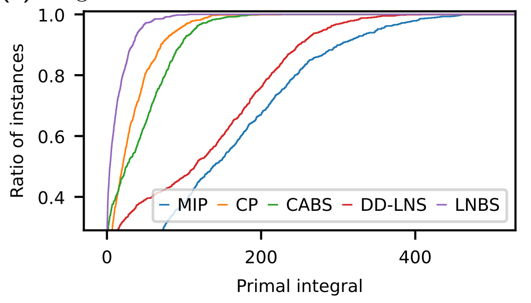
(c) m-PDTSP.



(d) Large instances in m-PDTSP.

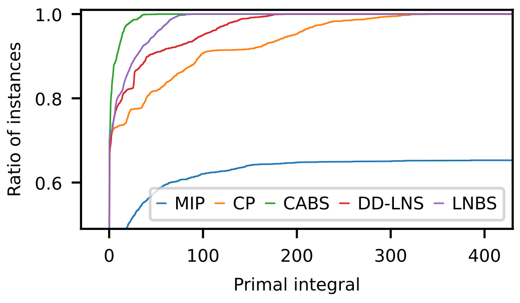


(e) $1||\sum w_i T_i$.

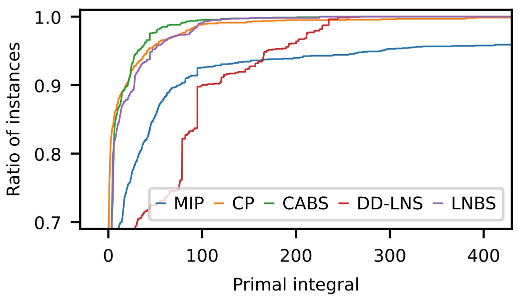


(f) Talent.

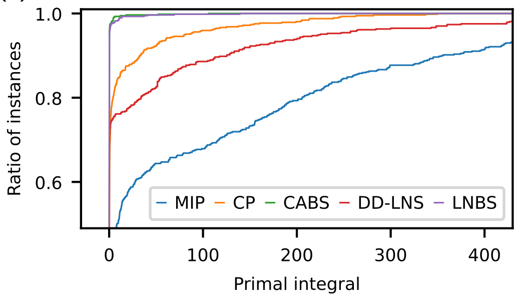
Figure 5 Distribution of the primal integral at the time limit in the problems where LNBS is better. Higher and left is better.



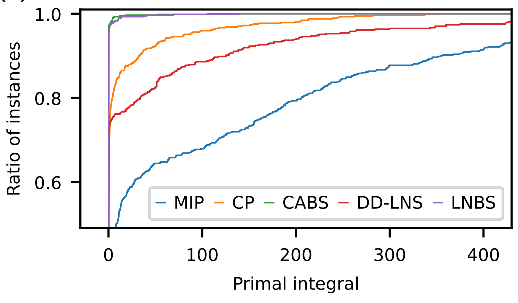
(a) SALBP-1.



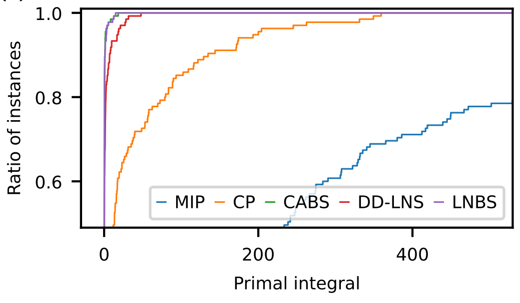
(b) BPP.



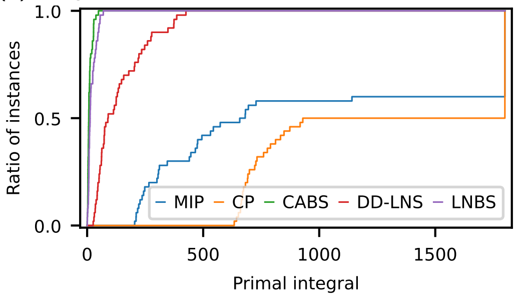
(c) MOSP.



(d) Large instances in MOSP.



(e) GCP.



(f) Large instances in GCP.

Figure 6 Distribution of the primal integral at the time limit in the problems where CABS is better. Higher and left is better.