Decision Diagram-Based Branch-and-Bound with Caching for Dominance and Suboptimality Detection

Vianney Coppé, Xavier Gillard, Pierre Schaus

UCLouvain, Louvain-la-Neuve, Belgium

{vianney.coppe,xavier.gillard,pierre.schaus}@uclouvain.be

The branch-and-bound algorithm based on decision diagrams introduced by Bergman et al. in 2016 is a framework for solving discrete optimization problems with a dynamic programming formulation. It works by compiling a series of bounded-width decision diagrams that can provide lower and upper bounds for any given subproblem. Eventually, every part of the search space will be either explored or pruned by the algorithm, thus proving optimality. This paper presents new ingredients to speed up the search by exploiting the structure of dynamic programming models. The key idea is to prevent the repeated expansion of nodes corresponding to the same dynamic programming states by querying expansion thresholds cached throughout the search. These thresholds are based on dominance relations between partial solutions previously found and on the pruning inequalities of the filtering techniques introduced by Gillard et al. in 2021. Computational experiments show that the pruning brought by this caching mechanism allows significantly reducing the number of nodes expanded by the algorithm. This results in more benchmark instances of difficult optimization problems being solved in less time while using narrower decision diagrams.

Key words: discrete optimization; decision diagrams; branch-and-bound; caching

1. Introduction

Many NP-hard combinatorial optimization problems can be modeled and solved with *dynamic programming* (DP). Although this resolution technique is appealing, the process of memorizing all solved subproblems rapidly becomes infeasible in terms of memory, as the required memory size can grow exponentially with the input size, rendering the computation intractable. Discrete optimization with *decision diagrams* (DDs) is a recent framework for addressing the memory issue for solving constraint optimization problems using their DP formulation. Apart from offering new modeling perspectives, this technique can exploit

the compactness of DP models within an adapted *branch-and-bound* (B&B) algorithm introduced by Bergman et al. (2016). In addition to conducting the search within the DP state space, the strength of DD-based B&B lies in its dedicated approach to deriving lower and upper bounds. Both are obtained by compiling bounded-width DDs, respectively called *restricted* and *relaxed* DDs – assuming a maximization problem. Those approximate DDs each have their own strategy for limiting their size, which works either by removing or merging excess nodes.

The DDs used today in the field of discrete optimization originate from compact encodings of Boolean functions, first known as binary decision programs (Lee 1959) and later as *binary decision diagrams* (BDDs) (Akers 1978, Bryant 1986). *Multi-valued decision diagrams* (MDDs) were then suggested by Kam and Brayton (1990) as an extension of BDDs to variables and functions taking values from discrete sets. These different variants of DDs were successfully applied in different domains such as formal verification (Hu 1995), model checking (Clarke et al. 1994), computer-aided design (Minato 1995) and optimization (Lai et al. 1994, Hachtel and Somenzi 1997, Becker et al. 2005, Hadžić and Hooker 2006, 2007). More recently, Andersen et al. (2007) introduced relaxed DDs that act as a constraint store for constraint programming solvers. These DDs can represent a superset of the feasible variable assignments with a bounded width that balances the computational costs and the filtering strength. Relaxed DDs were then adapted by Bergman et al. (2014a) as a mean of deriving upper bounds for discrete optimization problems that can be modeled with DP (Hooker 2013). They were followed closely by their lower bounding counterparts called restricted DDs (Bergman et al. 2014b), which represent a subset of the solution space.

Several improvements to the B&B algorithm based on these two ingredients have been suggested since its introduction: Gillard et al. (2021a) proposed additional bounding procedures to enhance the pruning of the B&B and thus speed up the search. Another promising research direction is to discover variable orderings that yield approximate DDs with better bounds. Cappart et al. (2022) designed a reinforcement learning approach to perform this task while Karahalios and van Hoeve (2022) proposed different portfolio mechanisms to dynamically select the best ordering among a predefined set of alternatives. For a complete overview of the latest theoretical contributions and applications of DDs in the field of discrete optimization, we refer the reader to the survey by Castro et al. (2022).

The contributions of this paper stem from a simple observation: as opposed to classical B&B for *mixed-integer programming* (MIP), the DD-based B&B does not split the search space into disjoint parts. The reason is that in this framework, DDs are based on a DP formulation of the discrete optimization problem at hand, and such kind of model typically contains many overlapping subproblems. As a result, the B&B may explore some subproblems multiple times. In addition, the approximate DDs compiled during the algorithm can repeat a lot of the work done previously because they cover overlapping parts of the search space. This inability to build on previous computational efforts is unfortunate, for that is the very strength of the DP paradigm. A first attempt to address this problem was made in (Coppé et al. 2022) by changing the ordering of the nodes in the B&B. By employing a breadth-first ordering, it is guaranteed that at most one B&B node corresponding to each given subproblem will be explored. However, this approach sacrifices the benefits of best-first search – always exploring the most promising node to try to improve the incumbent solution and tighten the bounds at the same time – and leaves the issue of overlapping approximate DDs unsolved.

In this paper, the same problem is tackled while allowing a best-first ordering of the B&B nodes. In the same fashion that a closed list prevents the re-expansion of nodes in shortest-path algorithms, we propose to maintain a *Cache* that stores an *expansion threshold* for each DP state reached by an exact node of any relaxed DD compiled during the B&B. These expansion thresholds combine *dominance* and *pruning thresholds* that exploit the information contained in relaxed DDs to the fullest extent. The former detect dominance relations between partial solutions while the latter extrapolate the pruning decisions related to nodes filtered by the *rough upper bounds* and *local bounds* introduced by Gillard et al. (2021a). By consulting the expansion thresholds stored in the *Cache*, many partial solutions are guaranteed to be either dominated by other partial solutions previously found, or suboptimal with respect to bounds already obtained, and can therefore be pruned during the compilation of subsequent approximate DDs.

In the same line of research, Rudich et al. (2022) suggested inserting open relaxed DDs directly in the B&B queue instead of open nodes, and described a *peeling* operation that splits a relaxed DD into two parts: the first containing all the paths traversing a chosen node and the second containing the rest. The peeling operation permits two things when used as a replacement for classical branching procedures: it allows warm-starting the compilation

of subsequent relaxed DDs, and it strengthens the bounds of the relaxed DD on which the peeling was conducted. This mechanism helps in reducing the overlap between the approximate DDs compiled during the search but does not fully address the issue. These improvements could thus be combined with the techniques hereby introduced.

The paper begins with a summary of DD-based B&B in Section 2, starting from the DD representation of a discrete optimization problem to the B&B algorithm and its latest improvements. This general introduction is followed by a discussion of the caveats that this paper tries to address in Section 3. Next, Section 4 presents the expansion thresholds through its two components – the dominance and pruning thresholds – as well as their computation and integration into both the B&B algorithm and the approximate DD compilation procedure. Section 5 then discusses the limitations of the techniques introduced in the paper. Finally, experimental results on three different discrete optimization problems are reported and discussed in Section 6 before concluding.

2. Preliminaries

This section provides an overview of DD-based discrete optimization. It starts with a reminder on how discrete optimization problems are modeled and solved with DP. Next, the translation of a DP formulation to DDs is explained. In particular, *restricted* and *relaxed* DDs are defined as well as the top-down compilation algorithm used to obtain them. The section ends with a description of the DD-based B&B algorithm exploiting the lower and upper bounds derived from restricted and relaxed DDs, respectively. Throughout the section, a simple *Bounded Knapsack Problem* (BKP) scenario is developed to help the reader visualize the concepts presented.

2.1. Dynamic Programming Formulation

A discrete optimization problem $\mathcal{P}$ is defined by $\max \{f(x) \mid x \in Sol(\mathcal{P}) = D \cap C\}$, where $x = (x_0, \ldots, x_{n-1})$ is a vector of variables taking values in their respective domains $x_j \in D_j$ with $D = D_0 \times \cdots \times D_{n-1}$, C is a set modeling all the constraints that solutions need to satisfy and f is the objective function to be maximized. To formulate $\mathcal{P}$ with DP, the following elements are needed:

- the *control variables* $x_j \in D_j$ with $j \in \{0, \ldots, n-1\}$.
- a *state space* S partitioned into $n+1$ sets $S_0, \ldots, S_n$ that correspond to distinct stages of the DP model. In particular, S_j contains all states having j variables assigned. Several special states are also defined: the *root* $\hat{r}$, the *terminal* $\hat{t}$ and the *infeasible* state $\hat{0}$.

- the *transition functions* $t_j : S_j \times D_j \to S_{j+1}$ for $j = 0, \dots, n-1$ that specify how states of consecutive stages are connected.

- the *transition value functions* $h_j : S_j \times D_j \to \mathbb{R}$ for $j = 0, \dots, n-1$ that compute the contribution of a transition to the objective.

- a *root value* v_r to account for constant terms in the objective.

If those elements can be defined for $\mathcal{P}$, then finding the optimal solution is equivalent to solving:

$$
\begin{aligned}
&\text{maximize } f(x) = v_r + \sum_{j=0}^{n-1} h_j(s^j, x_j) \\
&\text{subject to } s^{j+1} = t_j(s^j, x_j), \text{ for all } j = 0, \dots, n-1, \text{ with } x_j \in D_j \\
&\qquad s^j \in S_j, j = 0, \dots, n \text{ and } x \in C.
\end{aligned}
\tag{1}
$$

EXAMPLE 1. Given a set of items $N = \{0, \dots, n-1\}$ with weights $W = (w_0, \dots, w_{n-1})$, values $V = (v_0, \dots, v_{n-1})$ and quantities $Q = (q_0, \dots, q_{n-1})$, the goal of the BKP is to choose the number of copies of each item to include in the knapsack so that the total value is maximized, and the total weight is kept under a given capacity C. In the DP formulation of this problem, we associate a control variable $x_j \in \{0, \dots, q_j\}$ to each item j that decides the number of copies of it to include in the knapsack. States are uniquely identified by the remaining capacity of the knapsack. The state space is thus $S = \{0, \dots, C\}$, with in particular the root state $\hat{r} = C$ and its value $v_r = 0$. The transition functions are given by:

$$
t_j(s^j, x_j) = \begin{cases} s^j - x_j w_j, & \text{if } s^j \geq x_j w_j, \\ \hat{0}, & \text{otherwise.} \end{cases}
$$

The weight of item j is subtracted from the capacity as many times as the number of copies selected. If the remaining capacity does not allow including x_j copies of item j, the transition is redirected to the infeasible state $\hat{0}$. Similarly, the value of item j is added to the objective for each copy with the transition value functions $h_j(s^j, x_j) = x_j v_j$.

2.2. Decision Diagrams

In the context of discrete optimization, a DD is a graphical representation of the set of solutions to a given problem $\mathcal{P}$. DDs are well suited to encode and manipulate compact formulations such as DP models, due to their ability to preserve the uniqueness of overlapping subproblems. In mathematical terms, a decision diagram $\mathcal{B} = (U, A, \sigma, l, v)$ is a layered
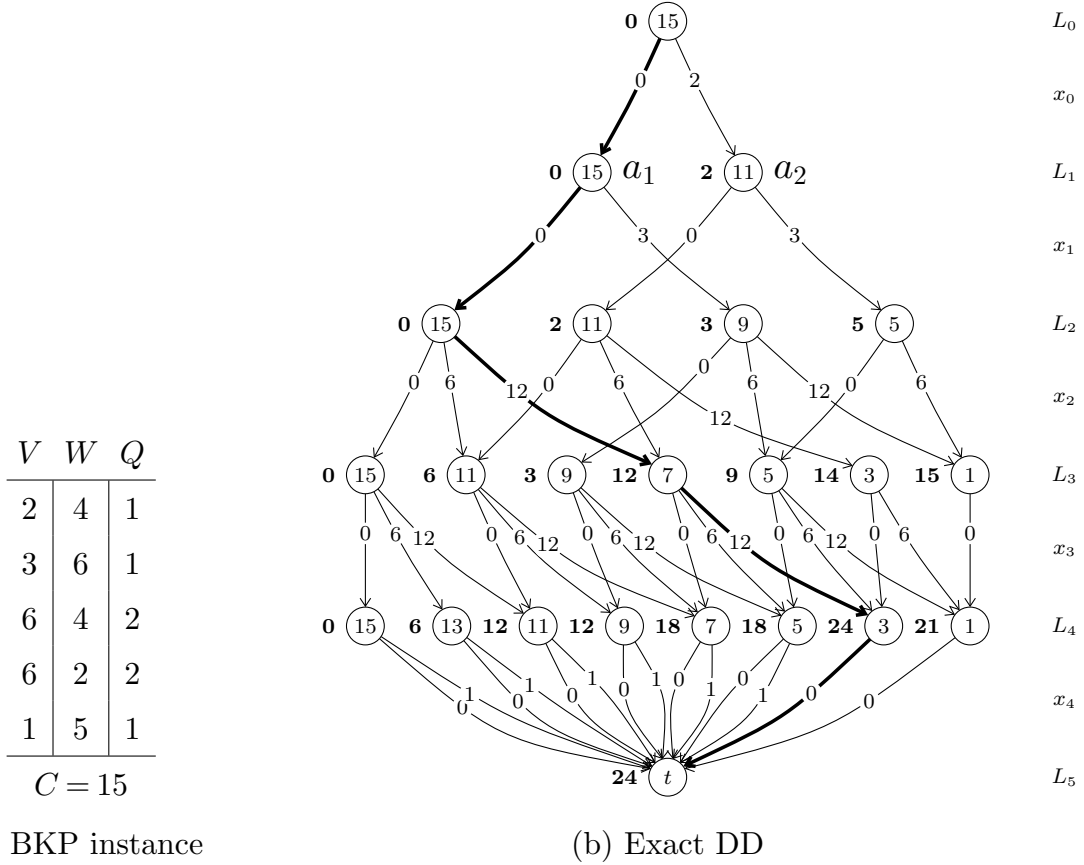
Figure 1   **(a) A BKP instance and (b) the corresponding exact DD. The value inside each node corresponds to its state – the remaining capacity – and the annotation on the left gives the value of the longest path that reaches it. For clarity, only arc values are present. The longest path is highlighted in bold.**

directed acyclic graph consisting of a set of nodes U that are connected by a set of arcs A. Each node is mapped to a DP state by the function σ. The set of nodes U can be partitioned into *layers* $L_0, \ldots, L_n$ corresponding to the successive stages of the DP model, each containing one node for each *distinct* state of the corresponding stage. Therefore, arcs $a = (u_j \xrightarrow{d} u_{j+1})$ connect nodes of consecutive layers $u_j \in L_j, u_{j+1} \in L_{j+1}$ and represent the transition between states $\sigma(u_j)$ and $\sigma(u_{j+1})$. The *label* $l(a) = d$ of an arc encodes the decision that assigns the value $d \in D_j$ to variable x_j. The value $v(a)$ of the arc stores the transition value. Both the first and last layer – L_0 and L_n – contain a single node, respectively the *root* r and the *terminal* node t. Each $r \rightsquigarrow t$ path $p = (a_0, \ldots, a_{n-1})$ that traverses the DD from top to bottom through arcs $a_0, \ldots, a_{n-1}$ represents a solution $x(p) = (l(a_0), \ldots, l(a_{n-1}))$ to $\mathcal{P}$. Its objective value is given by the accumulation of the arc values along the path: $v(p) = v_r + \sum_{j=0}^{n-1} v(a_j)$. Finally, the set of all solutions appearing in the DD is defined as $Sol(\mathcal{B}) = \{x(p) \mid \exists p : r \rightsquigarrow t, p \in \mathcal{B}\}$ and $\mathcal{B}$ is said *exact* if $Sol(\mathcal{B}) = Sol(\mathcal{P})$

Algorithm 1 Compilation of DD $\mathcal{B}$ rooted at node u_r with maximum width W.

1: $i \leftarrow$ index of the layer containing u_r

2: $L_i \leftarrow \{u_r\}$

3: **for** $j = i$ **to** $n - 1$ **do**

4:      $pruned \leftarrow \emptyset$

5:      **if** $j > i$ **then**   // pruning for the root done in Algorithm 3

6:          **for all** $u \in L_j$ **do**

7:              **if** $Cache.contains(\sigma(u))$ **and** $v^*(u \mid \mathcal{B}) \leq \theta(Cache.get(\sigma(u)))$ **then**

8:                  $pruned \leftarrow pruned \cup \{u\}$

9:      $L'_j \leftarrow L_j \setminus pruned$

10:      **if** $|L'_j| > W$ **then**

11:          restrict or relax the layer to get W nodes with Algorithm 2

12:      $L_{j+1} \leftarrow \emptyset$

13:      **for all** $u \in L'_j$ **do**

14:          **if** $v^*(u \mid \mathcal{B}) + \overline{v}_{rub}(\sigma(u)) \leq \underline{v}$ **then**   // rough upper bound pruning

15:              **continue**

16:          **for all** $d \in D_j$ **do**

17:              create node u' with state $\sigma(u') = t_j(\sigma(u), d)$ or retrieve it from L_{j+1}

18:              create arc $a = (u \xrightarrow{d} u')$ with $v(a) = h_j(\sigma(u), d)$ and $l(a) = d$

19:              add u' to L_{j+1} and add a to A

20: merge nodes in L_n into terminal node t

---

and $v(p) = f(x(p)), \forall p \in \mathcal{B}$. For convenience, all nodes u and paths p appearing in a DD $\mathcal{B}$ can be accessed respectively with the notation $u \in \mathcal{B}$ and $p \in \mathcal{B}$.

EXAMPLE 2. Figure 1(a) defines a small BKP instance with 5 items and a knapsack capacity of 15. The exact DD obtained by developing the DP model introduced in Example 1 for this instance is represented on Figure 1(b). The bold path is the longest path of the DD and corresponds to the optimal solution $x^* = (0, 0, 2, 2, 0)$ with value $f(x^*) = 24$.

Given the subproblem $\mathcal{P}|_{u_r}$ that restricts $\mathcal{P}$ to paths traversing some node u_r, Algorithm 1 details the top-down compilation of a DD rooted at u_r, given that $\sigma(u_r)$ is a state at the i-th stage of the DP model. The notations utilized in the algorithms are deliberately less formal to make the explanations easier to understand. Additionally, the highlighted

---

Algorithm 2 Restriction or relaxation of layer L'_j with maximum width W.

1: **while** $|L'_j| > W$ **do**

2:     $\mathcal{M} \leftarrow$ select nodes from L'_j

3:     $L'_j \leftarrow L'_j \setminus \mathcal{M}$

4:     create node μ with state $\sigma(\mu) = \oplus(\sigma(\mathcal{M}))$ and add it to L'_j // for relaxation only

5:     **for all** $u \in \mathcal{M}$ **and** arc $a = (u' \xrightarrow{d} u)$ incident to u **do**

6:        replace a by $a' = (u' \xrightarrow{d} \mu)$ and set $v(a') = \Gamma_{\mathcal{M}}(v(a), u)$

---

portions in Algorithms 1 and 3 pertain to the enhancements introduced in Section 4, which will be explained in further detail later. After initializing the first layer at line 2 with the root node only, the DD is developed layer by layer by applying every possible transition to each node of the last completed layer at lines 13 to 19. Once the DD is fully unrolled, line 20 merges all nodes of the terminal layer into a single terminal node t. At line 10, the *width* of the current layer is computed and compared against a parameter W called the *maximum width*. When the layer width exceeds W, the algorithm yields approximate DDs by either restricting or relaxing the layer at line 11. In order to obtain an exact DD, one can simply set $W = \infty$. For most combinatorial optimization problems, the size of the corresponding exact DD can grow exponentially, making it intractable to compute. Restricted and relaxed DDs are both capable of maintaining the width of all layers under a given maximum width but rely on different strategies to reduce the width of a layer. The former one yields lower bounds and feasible solutions while the latter can be used to derive upper bounds.

Restricted DDs At line 11, restriction is synonymous with removing surplus nodes. As detailed by Algorithm 2, the least promising nodes of the layer are selected according to a heuristic and simply dropped before resuming the compilation as normal. As a result, some solutions will not appear in restricted DDs. Nonetheless, all remaining solutions are feasible. For a restricted DD $\underline{\mathcal{B}}$, we thus have $Sol(\underline{\mathcal{B}}) \subseteq Sol(\mathcal{P})$ and $v(p) = f(x(p)), \forall p \in \underline{\mathcal{B}}$.

Relaxed DDs As opposed to restricted DDs that delete a part of the solutions, relaxed DDs will never remove feasible solutions but they might introduce infeasible ones. This is achieved by *merging* nodes together and thus redirecting arcs pointing to different nodes to a single *meta*-node, as performed by lines 4 to 6 of Algorithm 2. It requires defining problem-specific merging operators to merge the corresponding DP states. If $\mathcal{M}$ is the set
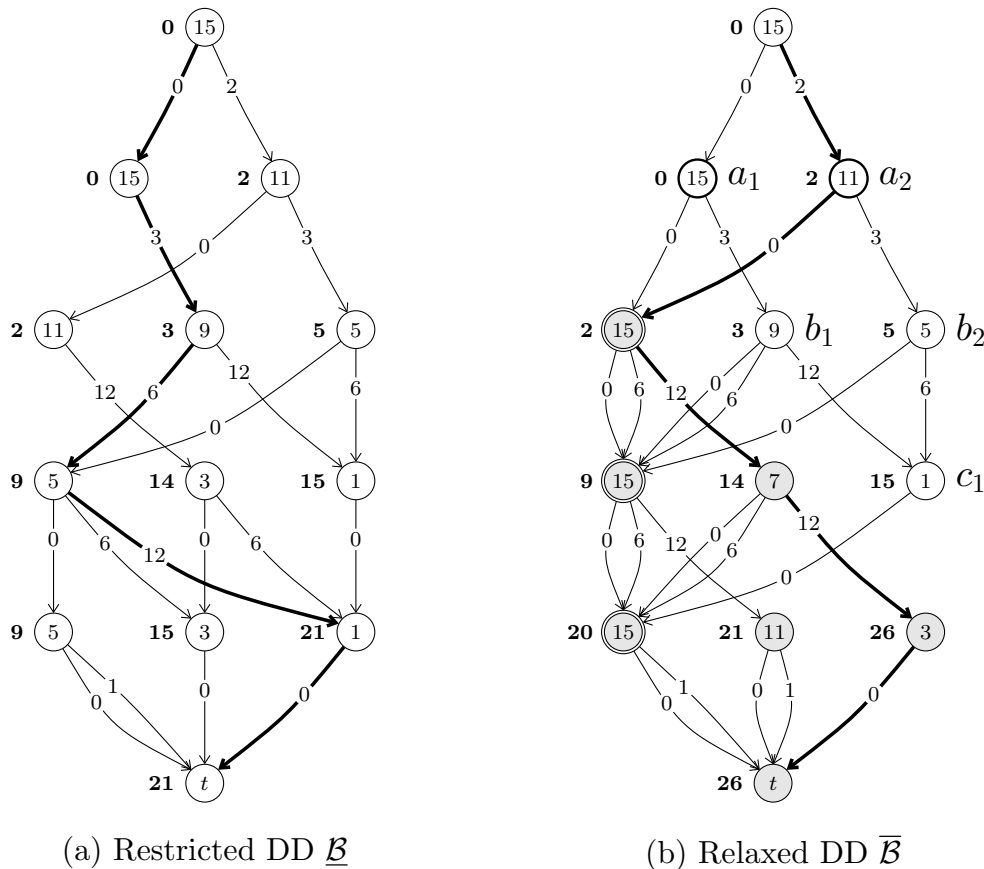
(a) Restricted DD $\underline{\mathcal{B}}$ (b) Relaxed DD $\overline{\mathcal{B}}$

Figure 2   **(a) A restricted and (b) a relaxed DD for the BKP instance of Figure 1(a) with $W = 3$. Merged nodes are circled twice and relaxed nodes are colored in gray.**

of nodes to merge and $\sigma(\mathcal{M}) = \{\sigma(u) \mid u \in \mathcal{M}\}$ the corresponding set of states, the operator $\oplus(\sigma(\mathcal{M}))$ gives the state of the merged node. The resulting state should encompass all merged states and preserve all their outgoing transitions. A second operator denoted $\Gamma_{\mathcal{M}}$ can be specified to adjust the value of the arcs incident to the merged node at line 6. As the merging operator gives an approximate representation of all merged states, it can introduce infeasible outgoing transitions. In addition, merging nodes u_1, u_2 allows combining any $r \rightsquigarrow u_1$ path with any $u_2 \rightsquigarrow t$ path and vice-versa. Given a valid relaxation operator, we can write for any relaxed DD $\overline{\mathcal{B}}$ that $Sol(\overline{\mathcal{B}}) \supseteq Sol(\mathcal{P})$ and $v(p) \geq f(x(p)), \forall p \in \overline{\mathcal{B}}$. To correctly explore the search space, we distinguish *exact nodes* from *relaxed nodes*. A node u in a DD $\mathcal{B}$ rooted at node u_r is said *exact* if, for any $r \rightsquigarrow u_r \rightsquigarrow u$ path in $\mathcal{B}$, applying all transitions specified by its arcs recursively from the root leads to the state $\sigma(u)$. All nodes that do not meet this criterion are called *relaxed*.

EXAMPLE 3. A valid merging operator for the BKP can simply keep the maximum remaining capacity among the states to merge: $\oplus(\sigma(\mathcal{M})) = \max_{u \in \mathcal{M}} \sigma(u)$. This ensures

that all solutions are preserved since it relaxes the capacity constraint. The operator $\Gamma_{\mathcal{M}}$ can be defined as the identity function since there is no need to modify the arc values. Figure 2 shows (a) a restricted and (b) a relaxed DD of width 3 for the BKP instance of Figure 1(a). In both cases, the nodes $u \in \mathcal{M}$ with the lowest longest $r \rightsquigarrow u$ path values were selected for removal or merging. The best solution in the restricted DD gives a lower bound of 21 while the relaxed DD gives an upper bound of 26. This upper bound corresponds to the solution $x = (1, 0, 2, 2, 0)$ with total weight 16, which violates the capacity constraint.

Before explaining how approximate DDs are embedded in a B&B scheme, let us provide some additional elements of notation. We denote by $p^*(u_1 \rightsquigarrow u_2 \mid \mathcal{B})$ an optimal path between nodes u_1 and u_2 in a DD $\mathcal{B}$ and by $v^*(u_1 \rightsquigarrow u_2 \mid \mathcal{B})$ its value. For a DD $\mathcal{B}$ rooted at a node u_r, we assume that an exact $r \rightsquigarrow u_r$ path $p(u_r) := p^*(r \rightsquigarrow u_r \mid \mathcal{B}')$ was found and attached to u_r during the prior compilation of another DD $\mathcal{B}'$. For conciseness, we denote by $p^*(u \mid \mathcal{B}) = p(u_r) \cdot p^*(u_r \rightsquigarrow u \mid \mathcal{B})$ an optimal path connecting the root node r of the problem and node u within the DD $\mathcal{B}$, with $\cdot$ the concatenation operator. The value of this path is given by $v^*(u \mid \mathcal{B}) = v(p(u_r) \cdot p^*(u_r \rightsquigarrow u \mid \mathcal{B})) = v(p(u_r)) + v^*(u_r \rightsquigarrow u \mid \mathcal{B})$. Furthermore, we write $p^*(\mathcal{B}) = p^*(t \mid \mathcal{B}), v^*(\mathcal{B}) = v^*(t \mid \mathcal{B})$ and $x^*(\mathcal{B}) = x(p^*(\mathcal{B}))$ to refer to one of the longest $r \rightsquigarrow t$ paths in $\mathcal{B}$, its value and the corresponding variable assignment. Finally, we define the *successors* of a node $u \in \mathcal{B}$ as the set of nodes reachable from u in $\mathcal{B}$, including u itself: $Succ(u \mid \mathcal{B}) = \{u\} \cup \{u' \mid (u \rightsquigarrow u') \in \mathcal{B}\}$.

A note about reduced DDs A DD is *reduced* (Bryant 1986, Wegener 2000) if it does not contain any *equivalent* subgraphs, that is, isomorphic subgraphs where corresponding nodes belong to the same layer and corresponding arcs share the same labels. DD reduction is very important for many applications, since it produces a unique DD of minimal size for a given variable ordering. However, *weighted* DDs used for optimization are less amenable to reduction since arc values further restrict which arcs can be superimposed. Hooker (2013) explained how transition cost functions of DP models can be rearranged to allow compiling reduced weighted DDs. Yet, this transformation is not always possible nor necessary since DDs are already reduced in some sense, indeed, they are derived from DP models that superimpose equivalent subproblems by design.

2.3. Branch-and-Bound

The two ingredients presented in Section 2.2 allow building a DD-only B&B algorithm, as introduced by Bergman et al. (2016). Restricted DDs are used to quickly find quality

---

Algorithm 3 The DD-based branch-and-bound algorithm.

1: $Fringe \leftarrow \{r\}$ // a priority queue of nodes ordered by decreasing $v(u) + \overline{v}(u)$

2: $Cache \leftarrow \emptyset$ // a hash table of states to threshold $\sigma(u) \rightarrow \langle \theta, expanded \rangle$

3: $\underline{x} \leftarrow \bot, \underline{v} \leftarrow -\infty$ // incumbent solution and its value

4: **while** $Fringe$ is not empty **do**

5:      $u \leftarrow$ best node from $Fringe$, remove it from $Fringe$

6:      **if** $v(u) + \overline{v}(u) \leq \underline{v}$ **then**

7:          **continue**

8:      **if** $Cache.contains(\sigma(u))$ **then**

9:          $\langle \theta, expanded \rangle \leftarrow Cache.get(\sigma(u))$

10:          **if** $v(u) < \theta$ **or** $(v(u) = \theta \wedge expanded)$ **then**

11:              **continue**

12:      $\underline{\mathcal{B}} \leftarrow Restricted(u)$ // compile restricted DD with Algorithm 1

13:      **if** $v^*(\underline{\mathcal{B}}) > \underline{v}$ **then**   // update incumbent

14:          $\underline{x} \leftarrow x^*(\underline{\mathcal{B}}), \underline{v} \leftarrow v^*(\underline{\mathcal{B}})$

15:      **if** $\underline{\mathcal{B}}$ is not exact **then**

16:          $\overline{\mathcal{B}} \leftarrow Relaxed(u)$ // compile relaxed DD with Algorithm 1

17:          compute LocBs with Algorithm 4 applied to $\overline{\mathcal{B}}$

18:          update $Cache$ with Algorithm 5 applied to $\overline{\mathcal{B}}$

19:          **for all** $u' \in EC(\overline{\mathcal{B}})$ **do**

20:              $v(u') \leftarrow v^*(u' \mid \overline{\mathcal{B}}), \overline{v}(u') \leftarrow \overline{v}(u' \mid \overline{\mathcal{B}}), p(u') \leftarrow p^*(u' \mid \overline{\mathcal{B}})$

21:              **if** $v(u') + \overline{v}(u') > \underline{v}$ **then**

22:                  add u' to $Fringe$

23: **return** $(\underline{x}, \underline{v})$

---

solutions from any starting node while relaxed DDs recursively decompose the problem at hand and provide bounds for the subproblems thus generated. The B&B algorithm is formalized by Algorithm 3 and summarized by Figure 3. Starting with the root node, line 4 loops over the set of open nodes, contained in a queue referred to as the *Fringe*. For each of them, a restricted DD $\underline{\mathcal{B}}$ is compiled at line 12. It may provide a solution with value $v^*(\underline{\mathcal{B}})$ improving the current best $\underline{v}$, in which case the incumbent is updated at lines 13 to 14. If $\underline{\mathcal{B}}$ is exact, then this subproblem can be considered as fully explored. Otherwise, a relaxed

DD $\overline{\mathcal{B}}$ must be developed at line 16. The role of this relaxed DD is twofold. First, it allows identifying a set of subproblems that need to be further explored so that all parts of the search space are covered. This set of subproblems is called an *exact cutset* (EC) of $\overline{\mathcal{B}}$.

DEFINITION 1 (EXACT CUTSET). An *exact cutset* $EC(\overline{\mathcal{B}})$ of a relaxed DD $\overline{\mathcal{B}}$ rooted at node u_r is a set of exact nodes such that any $u_r \rightsquigarrow t$ path crosses at least one of them.

Thanks to the divide-and-conquer nature of the DP model underlying the whole DD-based B&B approach, the node u_r corresponds to a subproblem $\mathcal{P}|_{u_r}$ that can be decomposed in a set of subproblems given by its direct successors in $\overline{\mathcal{B}}$. By transitivity, it follows that an EC of $\overline{\mathcal{B}}$ contains an exact decomposition of u_r, i.e., solving $\mathcal{P}|_{u_r}$ is equivalent to solving $\mathcal{P}|_u$ for all $u \in EC(\overline{\mathcal{B}})$. Therefore, nodes of the EC of $\overline{\mathcal{B}}$ are added to the *Fringe* in the B&B at line 22 in order to pursue the exhaustive exploration of the search space. The two most commonly used ECs are the *last exact layer* (LEL), which is the deepest layer in $\overline{\mathcal{B}}$ that contains only exact nodes, and the *frontier cutset* (FC) defined as $FC(\overline{\mathcal{B}}) = \left\{ u \in \overline{\mathcal{B}} \mid u \text{ is exact } \wedge \exists a = (u \xrightarrow{d} u') \text{ such that } u' \text{ is relaxed} \right\}$.

The second role of relaxed DDs is to compute an upper bound $\overline{v}(u)$ for each node $u \in EC(\overline{\mathcal{B}})$, which is used to discard nodes that are guaranteed to lead to solutions of value worse or equal to $\underline{v}$. This check is performed at line 21 before adding a node to the *Fringe* and is repeated at line 6 when a node is selected for exploration. While the upper bound $\overline{v}(u)$ for each node $u \in EC(\overline{\mathcal{B}})$ was originally based solely on $v^*(\overline{\mathcal{B}})$, stronger pruning can be obtained by using *local bounds* and *rough upper bounds* instead, as detailed in Section 2.4. In addition to the upper bound $\overline{v}(u)$, line 20 also attaches to each cutset node u the longest $r \rightsquigarrow u$ path $p(u) = p^*(u \mid \overline{\mathcal{B}})$ and its value $v(u) = v^*(u \mid \overline{\mathcal{B}})$. The cutset nodes are added to the *Fringe* unless their upper bound allows pruning them, and this whole process is repeated until the *Fringe* is emptied.

EXAMPLE 4. Back to our example problem defined by Figure 1(a), the DDs represented in Figure 2(a) and (b) could be respectively the first restricted and relaxed DDs compiled during the B&B. The best solution of the restricted DD would thus become the incumbent $\underline{v} = v^*(\underline{\mathcal{B}}) = 21$ and an EC should be extracted from the relaxed DD with upper bound $v^*(\overline{\mathcal{B}}) = 26$. The LEL is the second layer of the relaxed DD, containing only node a_1 and a_2, while the FC would encompass nodes a_1, a_2, b_1, b_2 and c_1 since they all have an arc pointing to a relaxed node. Assuming the LEL is used, Figure 3 shows how the B&B would pursue the search after having added nodes a_1 and a_2 to the *Fringe* and selecting
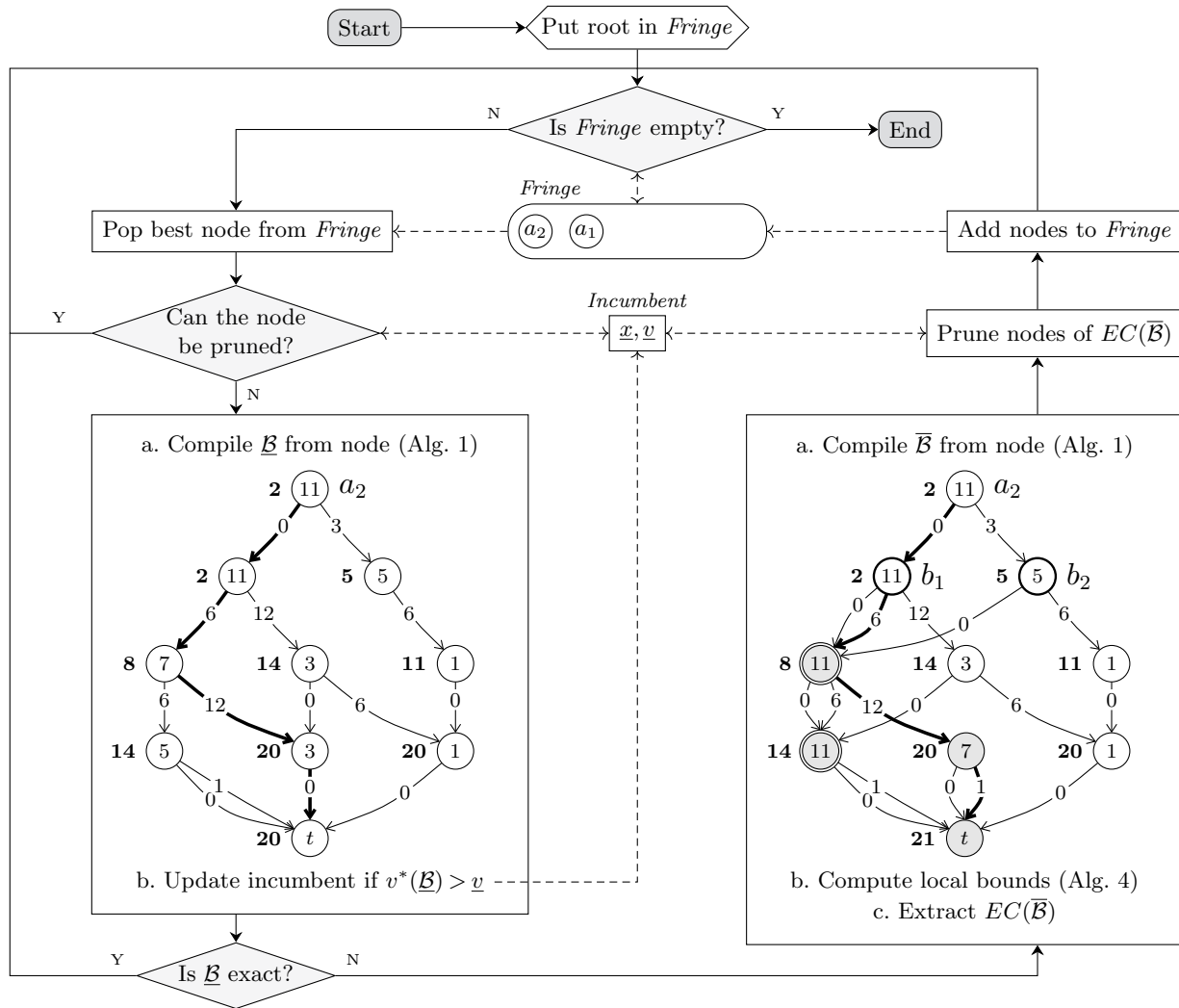
Figure 3   **Flowchart of the DD-based B&B algorithm.**

a_2 for exploration. The restricted DD compiled from a_2 fails to improve the incumbent. Furthermore, the upper bound provided by the relaxed DD is equal to the incumbent value, it is thus unnecessary to enqueue any node of the EC.

2.4.   Additional Filtering Techniques

Gillard et al. (2021a) introduced two additional bounding mechanisms. The first one is the *rough upper bound* (RUB) that aims at identifying suboptimal nodes as early as during the compilation of approximate DDs. It does so by computing a cheap problem-specific upper bound $\overline{v}_{rub}(\sigma(u))$ for every node at line 14 of Algorithm 1, which can help pruning many nodes before even generating their successors. The RUB is an additional modeling component that can be specified for the states of a given DP model, such that $\overline{v}_{rub}(\sigma(u)) \geq v^*(u \rightsquigarrow t \mid \mathcal{B})$ for any node u in $\mathcal{B}$, with $\mathcal{B}$ the exact DD for problem $\mathcal{P}$.

Algorithm 4 Computation of the local bound $\overline{v}_{locb}(u \,|\, \overline{\mathcal{B}})$ of every node u in $\overline{\mathcal{B}}$.

1: $i \leftarrow$ index of the root layer of $\overline{\mathcal{B}}$

2: $(L_i, \ldots, L_n) \leftarrow Layers(\overline{\mathcal{B}})$

3: $\overline{v}_{locb}(u \,|\, \overline{\mathcal{B}}) \leftarrow -\infty, mark(u) \leftarrow$ false **for each node** $u \in \overline{\mathcal{B}}$

4: $\overline{v}_{locb}(t \,|\, \overline{\mathcal{B}}) \leftarrow 0, mark(t) \leftarrow$ true

5: **for** $j = n$ **down to** i **do**

6: 　　**for all** $u \in L_j$ **do**

7: 　　　　**if** $mark(u)$ **then**

8: 　　　　　　**for all** arc $a = (u' \to u)$ incident to u **do**

9: 　　　　　　　$\overline{v}_{locb}(u' \,|\, \overline{\mathcal{B}}) \leftarrow \max\left\{\overline{v}_{locb}(u' \,|\, \overline{\mathcal{B}}), \overline{v}_{locb}(u \,|\, \overline{\mathcal{B}}) + v(a)\right\}$

10: 　　　　　　　$mark(u') \leftarrow$ true

In the original DD-based B&B algorithm, all nodes of the EC retrieved from the relaxed DD developed at line 16 of Algorithm 3 are added to the *Fringe* if the single upper bound given by $v^*(\overline{\mathcal{B}})$ is greater than $\underline{v}$. The *local bounds* (LocBs) presented by Gillard et al. (2021a) refine this reasoning by computing a distinct upper bound for each node u in $\overline{\mathcal{B}}$, given by the value of the longest $u \rightsquigarrow t$ path in $\overline{\mathcal{B}}$.

DEFINITION 2 (LOCAL BOUND). Given $\overline{\mathcal{B}}$ a relaxed DD for problem $\mathcal{P}$ and a node $u \in \overline{\mathcal{B}}$, the *local bound* $\overline{v}_{locb}(u \,|\, \overline{\mathcal{B}})$ of u within $\overline{\mathcal{B}}$ is given by:

$$\overline{v}_{locb}(u \,|\, \overline{\mathcal{B}}) = \begin{cases} v^*(u \rightsquigarrow t \,|\, \overline{\mathcal{B}}), & \text{if } (u \rightsquigarrow t) \in \overline{\mathcal{B}}, \\ -\infty, & \text{otherwise.} \end{cases} \quad (2)$$

LocBs can be computed efficiently by performing a bottom-up traversal of $\overline{\mathcal{B}}$, as formalized by Algorithm 4. In the algorithm, the *mark* flags are propagated to all nodes that have at least one path reaching the terminal node t. In parallel, the value $v^*(u \rightsquigarrow t \,|\, \overline{\mathcal{B}})$ of each marked node u is computed by accumulating the arc values traversed upwards. At the terminal node, $\overline{v}_{locb}(t \,|\, \overline{\mathcal{B}})$ is set to zero at line 4. For all other nodes, the value $\overline{v}_{locb}$ is updated by their direct successors at line 9.

The upper bound $\overline{v}(u \,|\, \overline{\mathcal{B}})$ used in Algorithm 3 is obtained by combining the RUB and the LocB derived from the compilation of a relaxed DD $\overline{\mathcal{B}}$:

$$\overline{v}(u \,|\, \overline{\mathcal{B}}) = \begin{cases} \overline{v}_{rub}(\sigma(u)), & \text{if } v^*(u \,|\, \overline{\mathcal{B}}) + \overline{v}_{rub}(\sigma(u)) \leq \underline{v}, \\ \min\left\{\overline{v}_{rub}(\sigma(u)), \overline{v}_{locb}(u \,|\, \overline{\mathcal{B}})\right\}, & \text{otherwise.} \end{cases} \quad (3)$$

(a) Restricted DD $\underline{\mathcal{B}}$ (b) Relaxed DD $\overline{\mathcal{B}}$
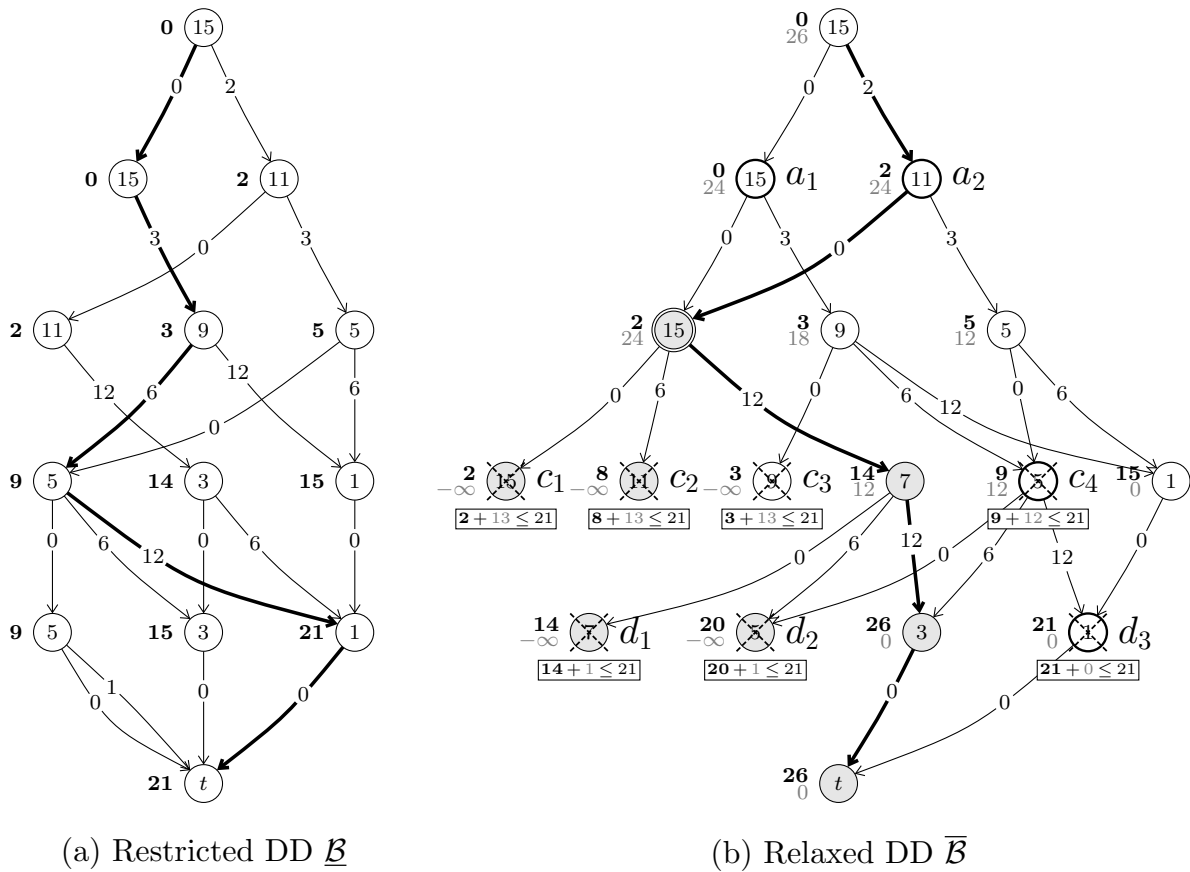
Figure 4   (a) A restricted and (b) a relaxed DD for the BKP instance of Figure 1(a) with $W = 3$. The LocBs are annotated in gray on the left of each node. Pruning decisions are detailed below the filtered nodes.

In the case where the RUB caused the node to be pruned at line 14 of Algorithm 1, the value of the RUB is kept. Otherwise, the minimum of both upper bounds is retained to maximize the pruning potential. Always keeping the minimum upper bound is also correct, but the definition given simplifies the exposition of the concepts introduced in Section 4.

EXAMPLE 5. A naive RUB for the BKP can simply add the maximum quantity of all remaining items to the knapsack, disregarding the capacity constraint. This can be written as $\overline{v}_{rub}(s^j) = \sum_{k=j}^{n-1} q_k v_k$. Figure 4 shows approximate DDs for the BKP instance of Figure 1(a) compiled from the root with $W = 3$, using RUB and LocB pruning this time. Figure 4(a) is the same restricted DD as in Figure 2(a) since RUB pruning requires an initial solution to compare with – except when infeasibility can be proved. However, once the lower bound of 21 is obtained, the relaxed DD of Figure 4(b) can successfully discard nodes c_1, c_2, c_3, d_1 and d_2 using RUB pruning. For instance, the RUB for all nodes of the fourth layer is given by $\overline{v}_{rub}(\sigma(u)) = q_3 v_3 + q_4 v_4 = 2 \times 6 + 1 \times 1 = 13$. This pruning improves

the quality of the relaxed DD as only one layer resorted to node merging, compared to three for the relaxed DD of Figure 2(b). On the other hand, the LocBs allow filtering nodes c_4 and d_3 if the FC is used. Indeed, for each of them we have $v^*(u \mid \overline{\mathcal{B}}) + \overline{v}_{locb}(u \mid \overline{\mathcal{B}}) = 21 \leq \underline{v} = 21$.

3.   Caveats of DD-based Branch-and-Bound

As explained in Section 2, DD-based B&B enables solving discrete optimization problems by taking advantage of the compactness of DP models. Nevertheless, a few observations suggest that all the properties of this type of model have not yet been exploited. First, we point out that branching does not split the search space into disjoint parts. Indeed, the very nature of DP models is the ability to solve a large problem by recursively dividing it into smaller *overlapping* subproblems. Yet, vanilla DD-based B&B processes subproblems as if they were completely independent. To give a better intuition of why this might cause the algorithm to waste computational effort, we propose to look at DD-based B&B as a classical shortest-path algorithm performed on the graph induced by a DP model – in case of directed acyclic graphs, the shortest-path and longest-path problems are equivalent. DD-based B&B can be seen as a combination of *best-first* search at the B&B level with a sort of *breadth-first* search up to the EC of relaxed DDs, coupled with dedicated bounding and pruning procedures. However, while it shares many similarities with shortest-path algorithms like A* (Hart et al. 1968), it lacks one of their important ingredients: the *closed list*. This data structure collects nodes expanded throughout the search and is used to check whether a given node was already expanded, to avoid expanding it again or adding it to the set of open nodes.

Due to the absence of such data structure, both levels of the B&B algorithm end up performing some computations multiple times. At the B&B level, nothing prevents from adding multiple nodes with the same DP state to the *Fringe* of Algorithm 3 nor triggering the compilation of approximate DDs for several of them at different stages of the search. Moreover, at the lower level, the approximate DDs can significantly overlap, even when rooted at nodes associated with different DP states. For instance, suppose we would compute exact DDs for the two subproblems contained in the FC of Figure 4(b), respectively rooted at a_1 and a_2. As one can observe on Figure 1(b), they respectively contain 18 and 15 nodes, among which 11 nodes appear in both DDs. Therefore, if these two DDs are compiled successively within the B&B, most of the work done for the first DD is repeated

to compile the second one while many transitions could actually yield a value worse or equal than the one obtained before. In general, the structure of DP state transition systems causes such scenarios to occur very frequently.

The aim of this paper is thus to integrate some form of closed list into DD-based B&B in order to mitigate the amount of duplicate computations at both levels of the algorithm. Unfortunately, a simple closed list cannot be used since DP states are not expanded purely in best-first order. In other words, given a relaxed DD $\overline{\mathcal{B}}$ and an exact node $u \in \overline{\mathcal{B}}$, nothing guarantees that $p^*(u \mid \overline{\mathcal{B}})$ – the longest path to reach u within $\overline{\mathcal{B}}$ – is the longest path to reach a node with state $\sigma(u)$ within the complete exact DD.

4. Branch-and-Bound with Caching

In this section, we explain how the caveats mentioned in Section 3 can be addressed. Our idea is to augment DD-based B&B with a caching mechanism that associates each DP state to an *expansion threshold* that conditions the need for future expansion of nodes with the same DP state. Formally, after compiling a relaxed DD $\overline{\mathcal{B}}$, an expansion threshold $\theta(u \mid \overline{\mathcal{B}})$ can be derived for each exact node $u \in \overline{\mathcal{B}}$ by exploiting the information contained in $\overline{\mathcal{B}}$. It is then stored as a *key-value* pair $\langle \sigma(u), \theta(u \mid \overline{\mathcal{B}}) \rangle$ in an associative array referred to as the *Cache*. When compiling a subsequent approximate DD $\mathcal{B}$, if a node u' happens to be generated such that $\sigma(u')$ corresponds to an existing key in the *Cache*, the expansion of node u' is skipped if its longest path value $v^*(u' \mid \mathcal{B})$ is not strictly greater than the expansion threshold stored. This filtering is carried out at lines 6 to 8 of Algorithm 1.

Concerning the value of these expansion thresholds, a classical DP caching strategy would keep track of the longest path value obtained for each DP state, i.e., by having $\theta(u \mid \overline{\mathcal{B}}) = v^*(u \mid \overline{\mathcal{B}})$ for each exact node $u \in \overline{\mathcal{B}}$. In this section, we show how stronger expansion thresholds can be obtained by exploiting all the information contained in relaxed DDs. To give a better intuition on the origin of the expansion thresholds, we split their computation into two distinct thresholds that each cover a specific scenario requiring the expansion node associated with an already visited DP state. The *dominance* thresholds detect dominance relations between partial solutions found throughout the search, while the *pruning* thresholds extrapolate successful past pruning decisions. Dominance thresholds can work alone if the RUB and LocB pruning rules are not used, but must be complemented by pruning thresholds whenever these extensions are involved. In the general case, they are thus combined into a single expansion threshold that covers both scenarios concomitantly.

After giving a formal definition of the aforementioned thresholds, we explain how they can be computed efficiently for all exact nodes visited during the compilation of relaxed DDs before being stored in the *Cache*. Finally, we describe how the B&B and the compilation of approximate DDs can be modified to benefit from the expansion thresholds.

4.1. Dominance Thresholds

As explained in Section 2.3, the compilation of a relaxed DD $\overline{\mathcal{B}}$ rooted at node u_r allows decomposing the corresponding subproblem into a set of subproblems given by nodes in $EC(\overline{\mathcal{B}})$. By extension, any exact node $u_1 \in \overline{\mathcal{B}}$ is also decomposed into a set of subproblems, given by nodes in $EC(\overline{\mathcal{B}})$ that belong to its successors $Succ(u_1 \,|\, \overline{\mathcal{B}})$. For example, an exact decomposition of node c_2 of Figure 5(a) is given by $Succ(c_2 \,|\, \overline{\mathcal{B}}) \cap EC(\overline{\mathcal{B}}) = \{d_1\}$. Among these nodes, only those that pass the pruning test at line 21 of Algorithm 3 are then added to the *Fringe* of the B&B. For simplicity, let us denote this set of nodes by $Leaves_d(u_1 \,|\, \overline{\mathcal{B}}) = \left\{ u \in Succ(u_1 \,|\, \overline{\mathcal{B}}) \cap EC(\overline{\mathcal{B}}) \,|\, v^*(u \,|\, \overline{\mathcal{B}}) + \overline{v}(u \,|\, \overline{\mathcal{B}}) > \underline{v} \right\}$ where $\underline{v}$ is the value of the incumbent solution. Because we are solving optimization problems, we are only interested in one of the longest $r \rightsquigarrow u_2$ paths for each node $u_2 \in Leaves_d(u_r \,|\, \overline{\mathcal{B}})$, and other paths with lower or equal value can safely be ignored. As a result, we can infer *dominance* relations between the paths obtained within relaxed DDs, and use them later to detect non-dominated paths that are thus still relevant to find the optimal solution of the problem.

DEFINITION 3 (DOMINANCE). Given two $r \rightsquigarrow u$ paths p_1 and p_2 in a relaxed DD $\overline{\mathcal{B}}$, p_1 is said to dominate p_2 – formally denoted $p_1 \succ p_2$ – if and only if $v(p_1) > v(p_2)$. If $v(p_1) \geq v(p_2)$, then p_1 weakly dominates p_2, expressed as $p_1 \succeq p_2$.

Given a relaxed DD $\overline{\mathcal{B}}$ and three exact nodes $u_1 \in \overline{\mathcal{B}}$, $u_2 \in Leaves_d(u_1 \,|\, \overline{\mathcal{B}})$ and $u_1' \notin \overline{\mathcal{B}}$ such that $\sigma(u_1') = \sigma(u_1)$. The first scenario that requires the expansion of node u_1' happens when it is possible that the concatenation $p_1 \cdot p_2$ of a prefix path $p_1 : r \rightsquigarrow u_1'$ with a suffix path $p_2 : u_1 \rightsquigarrow u_2$ dominates the longest path $p^*(u_2 \,|\, \overline{\mathcal{B}})$ found for the cutset node u_2 during the compilation of $\overline{\mathcal{B}}$. Formally, the dominance relation imposes that u_1' be expanded if $p_1 \cdot p_2 \succ p^*(u_2 \,|\, \overline{\mathcal{B}})$. This property leads to the definition below.

DEFINITION 4 (INDIVIDUAL DOMINANCE THRESHOLD). Given a relaxed DD $\overline{\mathcal{B}}$ and two exact nodes $u_1 \in \overline{\mathcal{B}}$, $u_2 \in Leaves_d(u_1 \,|\, \overline{\mathcal{B}})$, the *individual dominance threshold* of u_1 with respect to u_2 is defined by:

$$\theta_{id}(u_1 \,|\, u_2, \overline{\mathcal{B}}) = v^*(u_2 \,|\, \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \,|\, \overline{\mathcal{B}}) \tag{4}$$

$$= \theta_{id}(u_2 \,|\, u_2, \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \,|\, \overline{\mathcal{B}}). \tag{5}$$

PROPOSITION 1. *Given a relaxed DD $\overline{\mathcal{B}}$, exact nodes $u_1 \in \overline{\mathcal{B}}$, $u_2 \in Leaves_d(u_1 \mid \overline{\mathcal{B}})$, $u_1' \notin \overline{\mathcal{B}}$ such that $\sigma(u_1') = \sigma(u_1)$, and a path $p_1 : r \rightsquigarrow u_1'$, if $v(p_1) > \theta_{id}(u_1 \mid u_2, \overline{\mathcal{B}})$ then $p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \succ p^*(u_2 \mid \overline{\mathcal{B}})$. Inversely, if $v(p_1) \leq \theta_{id}(u_1 \mid u_2, \overline{\mathcal{B}})$ then $p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \preceq p^*(u_2 \mid \overline{\mathcal{B}})$.*

Proof. *By Equation (4), we have $v(p_1) > \theta_{id}(u_1 \mid u_2, \overline{\mathcal{B}}) = v^*(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$, or equivalently $v(p_1) + v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) > v^*(u_2 \mid \overline{\mathcal{B}})$. Using the path value definition, we obtain $v(p_1) + v(p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})) > v(p^*(u_2 \mid \overline{\mathcal{B}}))$. By concatenating the paths p_1 and $p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$, the inequality becomes $v(p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})) > v(p^*(u_2 \mid \overline{\mathcal{B}}))$, which by Definition 3 is equivalent to $p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \succ p^*(u_2 \mid \overline{\mathcal{B}})$. The proof of the second implication is obtained by replacing each occurrence of $>$ and $\succ$ respectively by $\leq$ and $\preceq$ in the proof above. As $v^*(u_2 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) = 0$, we also trivially have that Equation (4) and Equation (5) are equivalent.* $\square$

Expanding u_1' is required when a $r \rightsquigarrow u_1'$ path satisfies the first condition of Proposition 1, with respect to any node $u_2 \in Leaves_d(u_1 \mid \overline{\mathcal{B}})$. Therefore, we define a *dominance threshold* for each exact node $u_1 \in \overline{\mathcal{B}}$ that simply computes the least individual dominance threshold with respect to nodes in $Leaves_d(u_1 \mid \overline{\mathcal{B}})$.

DEFINITION 5 (DOMINANCE THRESHOLD). Given a relaxed DD $\overline{\mathcal{B}}$, the *dominance threshold* of an exact node $u_1 \in \overline{\mathcal{B}}$ is given by:

$$\theta_d(u_1 \mid \overline{\mathcal{B}}) = \begin{cases} \infty, & \text{if } Leaves_d(u_1 \mid \overline{\mathcal{B}}) = \emptyset, \\ \min_{u_2 \in Leaves_d(u_1 \mid \overline{\mathcal{B}})} \theta_{id}(u_1 \mid u_2, \overline{\mathcal{B}}), & \text{otherwise.} \end{cases} \qquad (6)$$

EXAMPLE 6. Let us illustrate the computation of the dominance thresholds with Figure 5(a), showing the relaxed DD compiled from node a_2 of our running example. Note that the dominance thresholds can be used alone here because RUBs and LocBs are disabled, pruning thresholds would otherwise be needed to obtain correct expansion thresholds. The FC of this relaxed DD consists of the nodes b_1, b_2, c_1 and d_1. By applying Definition 5, we have for each of them that $\theta_d(u \mid \overline{\mathcal{B}}) = \theta_{id}(u \mid u, \overline{\mathcal{B}}) = v^*(u \mid \overline{\mathcal{B}})$. The dominance thresholds of all other exact nodes in the diagram can be obtained by bottom-up propagation. For node c_2, we obtain $\theta_d(c_2 \mid \overline{\mathcal{B}}) = \theta_{id}(c_2 \mid d_1, \overline{\mathcal{B}}) = \theta_{id}(d_1 \mid d_1, \overline{\mathcal{B}}) - v(c_2 \rightarrow d_1) = 20 - 0 = 20$.

Given those dominance thresholds, Figure 5(b) shows the relaxed DD that would be compiled from node a_1. Two pairs of nodes share the same DP state: c_2, c_2' and d_1, d_1'. In (b), node c_2' is pruned since its value is lower than the dominance threshold computed for c_2 in (a), even though c_2' has a greater value than c_2. On the other hand, the dominance threshold computed in (a) for d_1 is not high enough to prevent the expansion of d_1' in (b).
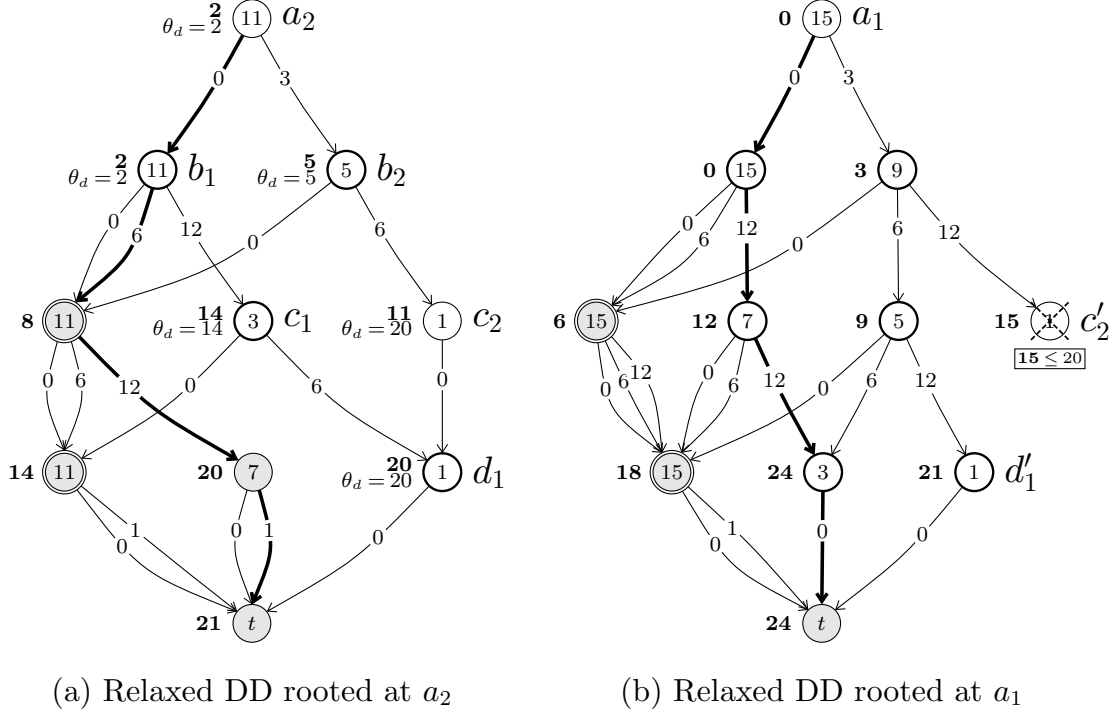
(a) Relaxed DD rooted at a_2 (b) Relaxed DD rooted at a_1

Figure 5   Relaxed DDs with $W = 3$ for the BKP instance of Figure 1 rooted at nodes (a) a_2 and (b) a_1. **Nodes of the relaxed DD (a) are annotated with their dominance threshold, where applicable. The relaxed DD (b) is compiled with respect to the dominance thresholds computed in (a).**

4.2. Pruning Thresholds

The second contribution concerns nodes that were pruned either because of their RUB or their LocB. It aims at identifying cases where a new path could circumvent a pruning decision that was made during the compilation of a relaxed DD $\overline{\mathcal{B}}$. For that purpose, we define a *pruning threshold* for each node $u_1 \in \overline{\mathcal{B}}$. Similarly to what was done for the dominance thresholds, we denote by $Leaves_p(u_1 \mid \overline{\mathcal{B}})$ the set of successor nodes of u_1 that have been pruned because of their RUB or their LocB, i.e., $Leaves_p(u_1 \mid \overline{\mathcal{B}}) = \{u_2 \in Succ(u_1 \mid \overline{\mathcal{B}}) \mid v^*(u_2 \mid \overline{\mathcal{B}}) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) \leq \underline{v}\}$ with $\underline{v}$ the value of the incumbent solution at that point. Formally, given a relaxed DD $\overline{\mathcal{B}}$ and three exact nodes $u_1 \in \overline{\mathcal{B}}$, $u_2 \in Leaves_d(u_1 \mid \overline{\mathcal{B}})$ and $u_1' \notin \overline{\mathcal{B}}$ with $\sigma(u_1') = \sigma(u_1)$, the expansion of node u_1' is required when it is possible that the concatenation $p_1 \cdot p_2$ of a new prefix path $p_1 : r \rightsquigarrow u_1'$ with a suffix path $p_2 : u_1 \rightsquigarrow u_2$ obtains a value sufficient to overcome the past pruning decision about node u_2. In other words, the pruning rule imposes that u_1' be expanded if $v(p_1 \cdot p_2) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$. In line with the methodology used for the dominance thresholds, we thus define the *individual pruning threshold* of an exact node $u_1 \in \overline{\mathcal{B}}$ with respect to a node $u_2 \in Leaves_p(u_1 \mid \overline{\mathcal{B}})$.

DEFINITION 6 (INDIVIDUAL PRUNING THRESHOLD). Given a lower bound $\underline{v}$, a relaxed DD $\overline{\mathcal{B}}$, an exact node $u_1 \in \overline{\mathcal{B}}$ and a pruned node $u_2 \in Leaves_p(u_1 \mid \overline{\mathcal{B}})$, the *individual pruning threshold* of u_1 with respect to u_2 is defined by:

$$\theta_{ip}(u_1 \mid u_2, \overline{\mathcal{B}}) = \underline{v} - \overline{v}(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) \tag{7}$$

$$= \theta_{ip}(u_2 \mid u_2, \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}). \tag{8}$$

PROPOSITION 2. *Given a lower bound $\underline{v}$, a relaxed DD $\overline{\mathcal{B}}$, an exact node $u_1 \in \overline{\mathcal{B}}$, a pruned node $u_2 \in Leaves_p(u_1 \mid \overline{\mathcal{B}})$, an exact node $u_1' \notin \overline{\mathcal{B}}$ such that $\sigma(u_1') = \sigma(u_1)$, and a path $p_1 : r \rightsquigarrow u_1'$. If $v(p_1) > \theta_{ip}(u_1 \mid u_2, \overline{\mathcal{B}})$ then $v(p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$. Inversely, if $v(p_1) \leq \theta_{ip}(u_1 \mid u_2, \overline{\mathcal{B}})$ then $v(p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) \leq \underline{v}$.*

Proof. By Equation (7), we have $v(p_1) > \theta_{ip}(u_1 \mid u_2, \overline{\mathcal{B}}) = \underline{v} - \overline{v}(u_2 \mid \overline{\mathcal{B}}) - v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$, *or equivalently $v(p_1) + v^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$. Using the path value definition, we obtain $v(p_1) + v(p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$. By concatenating the paths p_1 and $p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})$, the inequality becomes $v(p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \mid \overline{\mathcal{B}})) + \overline{v}(u_2 \mid \overline{\mathcal{B}}) > \underline{v}$. The proof of the second implication is obtained by replacing each occurrence of $>$ by $\leq$ in the proof above. As $v^*(u_2 \rightsquigarrow u_2 \mid \overline{\mathcal{B}}) = 0$, we also trivially have that Equation (7) and Equation (8) are equivalent.* $\square$

As for the dominance threshold, finding a $r \rightsquigarrow u_1'$ path satisfying the first condition of Proposition 2 with respect to any node $u_2 \in Leaves_p(u_1 \mid \overline{\mathcal{B}})$ is a sufficient condition to require the expansion of node u_1'. The *pruning threshold* of each exact node $u_1 \in \overline{\mathcal{B}}$ is thus computed as the least individual pruning threshold with respect to any node in $Leaves_p(u_1 \mid \overline{\mathcal{B}})$.

DEFINITION 7 (PRUNING THRESHOLD). Given a lower bound $\underline{v}$, a relaxed DD $\overline{\mathcal{B}}$ and an exact node $u_1 \in \overline{\mathcal{B}}$, the *pruning threshold* of u_1 within $\overline{\mathcal{B}}$ is given by:

$$\theta_p(u_1 \mid \overline{\mathcal{B}}) = \begin{cases} \infty, & \text{if } Leaves_p(u_1 \mid \overline{\mathcal{B}}) = \emptyset, \\ \min_{u_2 \in Leaves_p(u_1 \mid \overline{\mathcal{B}})} \theta_{ip}(u_1 \mid u_2, \overline{\mathcal{B}}), & \text{otherwise.} \end{cases} \tag{9}$$

EXAMPLE 7. We consider the same scenario as in Example 6 of a relaxed DD rooted at node a_2, but this time with RUBs and LocBs enabled. As shown by Figure 6(a), the RUBs manage to prune every path of the DD before reaching the terminal node. Therefore, we can showcase the pruning thresholds alone, otherwise we should combine them with the dominance thresholds, as will be explained in the next section. For each pruned node,
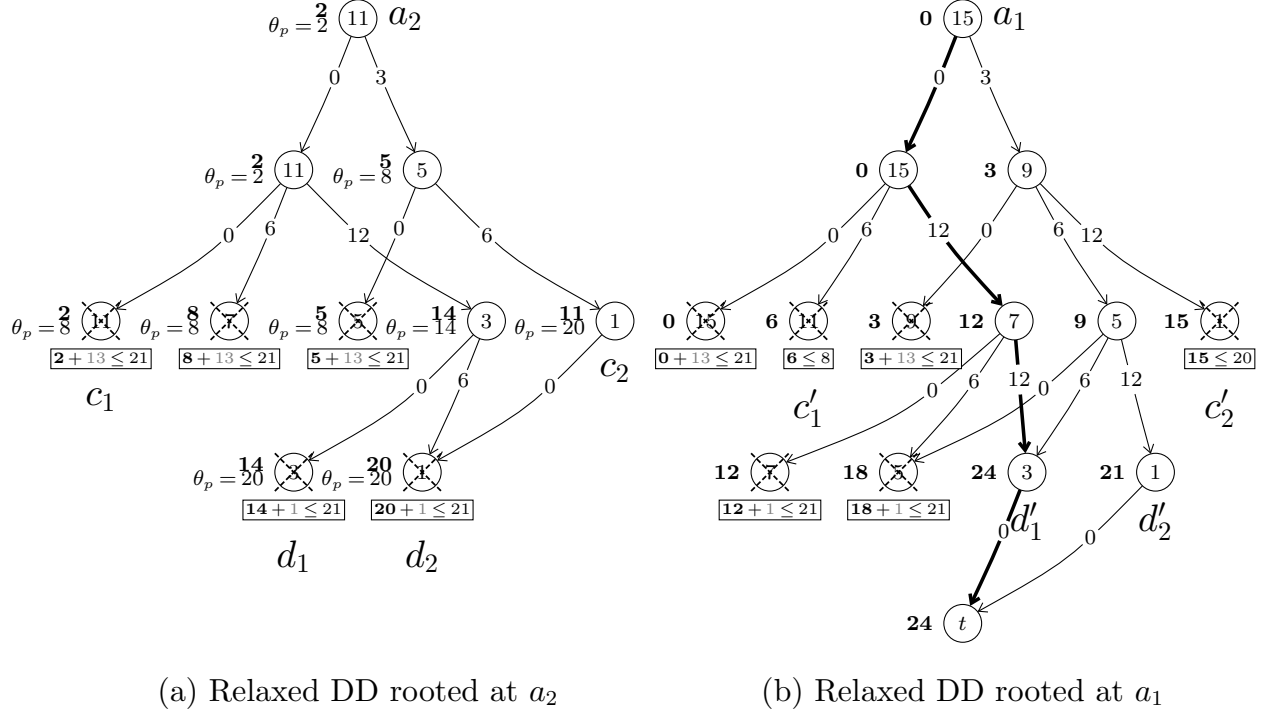
(a) Relaxed DD rooted at a_2 (b) Relaxed DD rooted at a_1

Figure 6   **Relaxed DDs with $W = 3$ for the BKP instance of Figure 1 rooted at nodes (a) a_2 and (b) a_1. Nodes of the relaxed DD (a) are annotated with their pruning threshold. The relaxed DD (b) is compiled with respect to the pruning thresholds computed in (a).**

we have by Definition 7 that $\theta_p(u \mid \overline{\mathcal{B}}) = \theta_{ip}(u \mid u, \overline{\mathcal{B}}) = \underline{v} - \overline{v}(u \mid \overline{\mathcal{B}})$. For instance, we have $\theta_p(c_1 \mid \overline{\mathcal{B}}) = 21 - 13 = 8$ and $\theta_p(d_2 \mid \overline{\mathcal{B}}) = 21 - 1 = 20$. We can then propagate these values to compute pruning thresholds of non-pruned nodes such as c_2: $\theta_p(c_2 \mid \overline{\mathcal{B}}) = \theta_{ip}(c_2 \mid d_2, \overline{\mathcal{B}}) = \theta_{ip}(d_2 \mid d_2, \overline{\mathcal{B}}) - v^*(c_1 \rightsquigarrow d_2 \mid \overline{\mathcal{B}}) = 20 - 0 = 20$.

Once these pruning thresholds are obtained, they can impact the compilation of a relaxed DD from node a_1, as illustrated by Figure 6(b). Among the nodes that share a common DP state across DDs (a) and (b), c_1' and c_2' are successfully filtered by the pruning thresholds computed for c_1 and c_2, although, like in Example 7, the values respectively obtained for c_1' and c_2' in (b) are greater than those of c_1 and c_2 in (a). The expansion of nodes d_1' and d_2' cannot be avoided. However, a new incumbent solution is found after expanding node d_1': $\underline{x} = (0, 0, 2, 2, 0)$ with $\underline{v} = f(\underline{x}) = 24$, which is the optimal solution of the problem. This concludes the resolution of our BKP instance by the B&B algorithm since both diagrams of Figure 6 are exact and a_1 and a_2 were the only two nodes in the EC of Figure 4(b).

4.3. Expansion Thresholds

Sections 4.1 and 4.2 presented the two scenarios that necessitate the expansion of nodes associated with a DP state previously reached by an exact node in a relaxed DD, along with the dominance and pruning thresholds used to detect them. This section explains how these two thresholds are combined to form an *expansion threshold*, and provides a proof of correctness of the accompanying pruning criterion. It also details the algorithm for computing the expansion threshold for all exact nodes in a relaxed DD.

Dominance and pruning thresholds each give a sufficient condition for the expansion of a node with a previously reached DP state. The definition of the expansion threshold given below simply ensures that the expansion is triggered if any of those two conditions are met, and inversely that it is avoided if none of those two conditions are met.

DEFINITION 8 (EXPANSION THRESHOLD). Given a relaxed DD $\overline{\mathcal{B}}$, the *expansion threshold* of an exact node $u_1 \in \overline{\mathcal{B}}$ is defined by $\theta(u_1 \,|\, \overline{\mathcal{B}}) = \min \left\{ \theta_d(u_1 \,|\, \overline{\mathcal{B}}), \theta_p(u_1 \,|\, \overline{\mathcal{B}}) \right\}$.

PROPOSITION 3. *Given a relaxed DD $\overline{\mathcal{B}}$, an exact node $u_1 \in \overline{\mathcal{B}}$, an exact node $u_1' \notin \overline{\mathcal{B}}$ such that $\sigma(u_1') = \sigma(u_1)$, and a path $p_1 : r \rightsquigarrow u_1'$, if $v(p_1) \leq \theta(u_1 \,|\, \overline{\mathcal{B}})$ then p_1 can be pruned.*

Proof. If $v(p_1) \leq \theta(u_1 \,|\, \overline{\mathcal{B}})$ then by definition of the expansion threshold, we have $v(p_1) \leq \theta_d(u_1 \,|\, \overline{\mathcal{B}})$ and $v(p_1) \leq \theta_p(u_1 \,|\, \overline{\mathcal{B}})$. For the dominance threshold inequality, we have by Definition 5 that $\forall u_2 \in Leaves_d(u_1 \,|\, \overline{\mathcal{B}}) : v(p_1) \leq \theta_{id}(u_1 \,|\, u_2, \overline{\mathcal{B}})$. When applying Proposition 1, we obtain $\forall u_2 \in Leaves_d(u_1 \,|\, \overline{\mathcal{B}}) : v(p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \,|\, \overline{\mathcal{B}})) \preceq p^*(u_2 \,|\, \overline{\mathcal{B}})$. The second inequality, concerning the pruning threshold, can be developed similarly: by Definition 7, $\forall u_2 \in Leaves_p(u_1 \,|\, \overline{\mathcal{B}}) : v(p_1) \leq \theta_{ip}(u_1 \,|\, u_2, \overline{\mathcal{B}})$. And by substituting with Proposition 2, $\forall u_2 \in Leaves_p(u_1 \,|\, \overline{\mathcal{B}}) : v(p_1 \cdot p^*(u_1 \rightsquigarrow u_2 \,|\, \overline{\mathcal{B}})) + \overline{v}(u_2 \,|\, \overline{\mathcal{B}}) \leq \underline{v}$. Therefore, if $v(p_1) \leq \theta(u_1 \,|\, \overline{\mathcal{B}})$ then the concatenation of p_1 with any path $u_1 \rightsquigarrow u_2$ with $u_2 \in Leaves_d(u_1 \,|\, \overline{\mathcal{B}}) \cup Leaves_p(u_1 \,|\, \overline{\mathcal{B}})$ is guaranteed to be either weakly dominated or pruned. Furthermore, by definition of $Leaves_d(u_1 \,|\, \overline{\mathcal{B}})$ and $Leaves_p(u_1 \,|\, \overline{\mathcal{B}})$, their union covers all successors of u_1 that are in $EC(\overline{\mathcal{B}})$ or that are pruned. It thus constitutes an exhaustive decomposition of the corresponding subproblem, concluding our argument that p_1 is irrelevant for the search. □

In Definitions 4 and 6, one can notice that the individual dominance and pruning thresholds $\theta_{id}(u_1 \,|\, u_2, \overline{\mathcal{B}})$ and $\theta_{ip}(u_1 \,|\, u_2, \overline{\mathcal{B}})$ only depend on the value $v^*(u_1 \rightsquigarrow u_2 \,|\, \overline{\mathcal{B}})$, apart from the thresholds $\theta_{id}(u_2 \,|\, u_2, \overline{\mathcal{B}})$ and $\theta_{ip}(u_2 \,|\, u_2, \overline{\mathcal{B}})$. For a relaxed DD rooted at node u_r, if those initial individual thresholds are correctly set for nodes in $Leaves_d(u_r \,|\, \overline{\mathcal{B}})$ and $Leaves_p(u_r \,|\,$

Algorithm 5 Computation of the threshold $\theta(u \mid \overline{\mathcal{B}})$ of every exact node u in a relaxed DD $\overline{\mathcal{B}}$ and update of the *Cache*.

1: $i \leftarrow$ index of the root layer of $\overline{\mathcal{B}}$
2: $(L_i, \ldots, L_n) \leftarrow Layers(\overline{\mathcal{B}})$
3: $\theta(u \mid \overline{\mathcal{B}}) \leftarrow \infty$ **for all** $u \in \overline{\mathcal{B}}$
4: **for** $j = n$ **down to** i **do**
5:     **for all** $u \in L_j$ **do**
6:         **if** $Cache.contains(\sigma(u))$ **and** $v^*(u \mid \overline{\mathcal{B}}) \leq \theta(Cache.get(\sigma(u)))$ **then**
7:             $\theta(u \mid \overline{\mathcal{B}}) \leftarrow \theta(Cache.get(\sigma(u)))$
8:         **else**
9:             **if** $v^*(u \mid \overline{\mathcal{B}}) + \overline{v}_{rub}(\sigma(u)) \leq \underline{v}$ **then**   // rough upper bound pruning
10:                 $\theta(u \mid \overline{\mathcal{B}}) \leftarrow \underline{v} - \overline{v}_{rub}(\sigma(u))$
11:             **else if** $u \in EC(\overline{\mathcal{B}})$ **then**
12:                 **if** $v^*(u \mid \overline{\mathcal{B}}) + \overline{v}_{locb}(u \mid \overline{\mathcal{B}}) \leq \underline{v}$ **then**   // local bound pruning
13:                     $\theta(u \mid \overline{\mathcal{B}}) \leftarrow \min\left\{\theta(u \mid \overline{\mathcal{B}}), \underline{v} - \overline{v}_{locb}(u \mid \overline{\mathcal{B}})\right\}$
14:                 **else**
15:                     $\theta(u \mid \overline{\mathcal{B}}) \leftarrow v^*(u \mid \overline{\mathcal{B}})$
16:             **if** u is exact **then**
17:                 $expanded \leftarrow$ false **if** $u \in EC(\overline{\mathcal{B}})$ **else** true
18:                 $Cache.insertOrReplace(\sigma(u), \langle\theta(u \mid \overline{\mathcal{B}}), expanded\rangle)$
19:         **for all** arc $a = (u' \xrightarrow{d} u)$ incident to u **do**
20:             $\theta(u' \mid \overline{\mathcal{B}}) \leftarrow \min\left\{\theta(u' \mid \overline{\mathcal{B}}), \theta(u \mid \overline{\mathcal{B}}) - v(a)\right\}$

$\overline{\mathcal{B}}$), the expansion thresholds can thus be computed by performing a single bottom-up pass on the relaxed DD $\overline{\mathcal{B}}$, as described by Algorithm 5.

The thresholds are first initialized at line 3 with a default value of ∞ for all nodes. Then, the loops at lines 4 and 5 iterate through all nodes of the DD, starting with the nodes of the last layer up to those of the first layer, and propagate threshold values in a bottom-up fashion. Depending on whether they are pruned or belong to the EC, several nodes require a careful initialization of their expansion threshold.

- **Pruned by the *Cache*:** line 6 checks whether the *Cache* already contains an expansion threshold greater than $v^*(u \mid \overline{\mathcal{B}})$. If so, the node was pruned and the previously computed threshold can simply be recycled, as it remains valid throughout the entire algorithm.

- **Pruned by RUB:** when reaching a node u pruned with its RUB during the top-down compilation at line 9, line 10 sets its expansion threshold $\theta_{ip}(u \mid u, \overline{\mathcal{B}})$ the individual pruning threshold of u relative to itself, because it has no other successor.

- **Pruned by LocB:** if a node u of the EC was pruned because of its LocB however, the expansion threshold is computed at line 13 as the minimum between $\theta_{ip}(u \mid u, \overline{\mathcal{B}})$ the individual pruning threshold relative to u and the current value of $\theta(u \mid \overline{\mathcal{B}})$, which accounts for the individual dominance and pruning thresholds relative to successors of u.

- **EC node:** the expansion threshold of each cutset node u that will be added to the *Fringe* is simply given by $\theta_{id}(u \mid u, \overline{\mathcal{B}})$ the individual dominance threshold of u relative to itself, as shown at line 15.

Lines 19 and 20 then take care of the bottom-up propagation of the expansion thresholds as to obtain the correct threshold values for all nodes. The last step at line 18 is to save the expansion threshold computed for each exact node in the *Cache* for later use. The *Cache* is an associative array storing *key-value* pairs $\langle \sigma(u), \theta(u \mid \overline{\mathcal{B}}) \rangle$, implemented as a *hash table* in practice. Note that Algorithm 5 can be applied to any type of EC, but an important detail is that thresholds should not be inserted in the *Cache* for exact successors of nodes in the EC to allow for their later expansion. As the FC contains all the deepest exact nodes, the algorithm can be applied as is. For the LEL however, a check must be added at line 16.

In addition to the threshold value, a flag called *expanded* is stored in the *Cache* to distinguish two types of expansion thresholds. First, the nodes that have been added to the *Fringe* from the EC of a relaxed DD $\overline{\mathcal{B}}$ with $\theta(u \mid \overline{\mathcal{B}}) = v^*(u \mid \overline{\mathcal{B}})$ and that need to be further explored by the B&B, therefore being associated with an expanded flag set to *false* at line 18 of Algorithm 5. And second, all exact nodes above the EC of any relaxed DD $\overline{\mathcal{B}}$ in which the threshold $\theta(u \mid \overline{\mathcal{B}})$ was obtained. In this case, they can be considered as expanded since all their outgoing paths were either pruned or cross a node in the EC of $\overline{\mathcal{B}}$. When those thresholds are inserted in the *Cache* at line 18 of Algorithm 5, their *expanded* flag is thus set to *true*.

4.4. Filtering the Search Using the Cache

Now that we have presented how expansion thresholds can be computed for all exact nodes of relaxed DDs, we can now elaborate on how these values are utilized within the B&B. There are two places where the *Cache* might be beneficial and prune some nodes. As mentioned before, and shown in Examples 6 and 7, the first is in the top-down compilation of approximate DDs, as described by Algorithm 1. At line 7, the *Cache* is queried and whenever the value $v^*(u \mid \mathcal{B})$ of a node u fails to surpass the threshold θ, it is added to the *pruned* set. This set of pruned nodes is used to define L'_j at line 9, a clone of the j-th layer from which the pruned nodes have been removed. In the rest of the algorithm, the pruned layer L'_j is employed instead of L_j to prevent generating any outgoing transition from the pruned nodes. Nodes from the *pruned* set are kept in the original layer L_j so that the threshold computations of Algorithm 5 can be performed correctly.

Furthermore, the *Cache* can be used when selecting a node for exploration in the B&B loop. At line 8 of Algorithm 3, the *Cache* is queried to potentially retrieve an entry with a threshold value as well as an *expanded* flag. If a threshold is indeed present, two cases arise depending on the value of the *expanded* flag. If it is *True*, the node is ignored if its value is less or equal to the threshold. When *False* however, the node is only ignored if its value is strictly less than the threshold, because in case of equality, this node will be the first to be expanded with that value. As this filtering happens at the B&B level, it is not repeated for the root node of approximate DDs, see line 5 of Algorithm 1.

An interesting implication of the use of expansion thresholds is that during the bottom-up traversal of relaxed DDs done to compute LocBs, it is possible to detect *dangling nodes*, i.e., nodes with no feasible outgoing transition. As they have no path to the terminal node, their LocB is $-\infty$ and the expansion threshold stored is ∞. As a result, the bottom-up propagation of the thresholds permits to detect both suboptimality and infeasibility earlier in the future compilations of approximate DDs, in addition to dominance relations discussed before. In terms of complexity, computing the thresholds requires a single traversal of the relaxed DDs compiled, so it has the same complexity as Algorithm 1 and Algorithm 4 that respectively handle the top-down compilation and the computation of the LocBs.

5. Limitations

The thresholds presented in Section 4 offer new pruning perspectives that will be shown to be very impactful in Section 6. Yet, they exhibit two main limitations that are discussed in this section.

5.1. Memory Consumption

In order to apply B&B with caching, extra information must be stored in memory. Some effort is done to reduce memory consumption by deleting thresholds as soon as they are no longer required by the algorithm. A sufficient condition to remove a threshold is when it concerns a node located in a layer above the *first active layer*.

DEFINITION 9 (FIRST ACTIVE LAYER). Given $\mathcal{B}$ the exact DD for problem $\mathcal{P}$ with layers $L_0, \ldots, L_n$. The *first active layer* of the B&B specified by Algorithm 3 is defined as the least index j such that a node of layer L_j is in the *Fringe* or is currently selected for exploration at line 5.

Thresholds related to nodes above the first active layer can be safely removed from the *Cache* as there is no way of reaching the associated DP states again. If memory consumption was nevertheless an issue, a simple solution would be to delete an arbitrary subset of the thresholds stored in the *Cache*. This does not compromise the optimality guarantees of the algorithm since the only effect of thresholds is to prevent the expansion of nodes associated with already visited DP states. However, removing some thresholds decreases the pruning perspectives of the algorithm. Some memory will therefore be saved at the cost of speed. One could even imagine using eviction rules based on an activity measure of the thresholds in the *Cache*. Note that all the algorithms presented in the paper are written as if the *Cache* could remove some thresholds along the way.

An argument that could be held against the use of the *Cache* in the context of DD-based B&B is that in the worst case, it still might need to accommodate as many nodes as there are nodes in the exact DD encoding the problem being solved. While this argument is true, we would like to point out that the aforementioned explosive worst-case memory requirement already plagued the original B&B algorithm as it does not implement any measures to limit the size of the *Fringe*. Furthermore, we would like to emphasize that maintaining the *Cache* is no more costly than maintaining the *Fringe* and significantly less expensive than memorizing an actual instantiation of the exact DD. Indeed, the maintenance of both the *Cache* and the *Fringe* is $O(|U|)$ – where U is the set of nodes in the state space – whereas the requirement to store an actual instantiation of the exact DD in memory is $O(|U|+|A|)$ where A is the set of arcs connecting the nodes in U. It is also worth mentioning that in that case, $|A|$ dominates $|U|$ since by definition of the domains and transition relations, the size of A is bounded by the product of the domain sizes $\Pi_{j=0,\ldots,n-1}|D_j|$.

5.2. Variables Orderings

Many discrete optimization problems have an imposed variable ordering in their natural DP model. For instance, DP models for sequencing problems usually make decisions about the j-th position of the sequence at the j-th stage of the DP model (Cire and van Hoeve 2013). For DP models that allow it, however, it has been shown that variable orderings can yield exact DDs of reduced size as well as approximate DDs with tighter bounds (Bergman et al. 2012, Cappart et al. 2022, Karahalios and van Hoeve 2022). When DDs are used within a B&B algorithm, variable orderings thus constitute an additional heuristic that can speed up the search. They can be separated in two categories: *static* and *dynamic* variable orderings. The former refers to a single variable ordering used for all DD compilations during the B&B. The latter denotes a heuristic that dynamically decides which variable to branch on when generating the next layer of a DD, based on the states contained in the current layer. Since the techniques introduced in this paper are solely based on the overlapping structure of the DP models, a dynamic variable ordering would most likely compromise much of the expected pruning. Indeed, it seems unlikely that many states would overlap in DDs compiled with different variable orderings, although it ultimately depends on the modeling of each specific problem.

6. Computational Experiments

This section presents the results of the computational experiments that were conducted to evaluate the impact of the additional pruning techniques presented in this paper. We performed experiments on three different well-known discrete optimization problems: the asymmetric variant of the Traveling Salesman Problem with Time Windows (TSPTW), the Pigment Sequencing Problem (PSP) – also known as the Discrete Lot Sizing and Scheduling Problem (DLSP) in the literature – and the Single-Row Facility Layout Problem (SRFLP). We describe below the experimental setting used for each problem.

TSPTW The solvers were tested on a classical set of benchmark instances introduced in the following papers (Ascheuer 1996, Dumas et al. 1995, Gendreau et al. 1998, Langevin et al. 1993, Ohlmann and Thomas 2007, Pesant et al. 1998, Potvin and Bengio 1996). A dynamic width was used, where the maximum width for layers at depth j is given by $n \times (j+1) \times \alpha$ with n the number of variables in the instance and α a multiplying parameter for which several values were used in the experiments.

PSP A set of randomly generated instances was created with the number of items in $\{5, 7, 10\}$, the number of periods in $\{50, 100, 150, 200\}$ and the density in $\{0.9, 0.95, 1\}$. The density is computed as the number of demands over the number of time periods. An additional parameter $\rho \in \{0.001, 0.01, 0.1\}$ was introduced to control the proportion between the stocking costs and the changeover costs. The costs are then sampled uniformly in $[\rho \cdot 5000, \rho \cdot 15000]$ and $[(1 - \rho) \cdot 5000, (1 - \rho) \cdot 15000]$, respectively. To generate the demands, item type and time period pairs were selected uniformly among all possible values and added to the instance as long as it remained feasible. Five instances were generated for each combination of the parameters mentioned. A fixed width was used but which is also proportional to the number of variables: $n \times \alpha$.

SRFLP The experiments consist of a compilation of benchmark instances from (Amaral 2006, 2008, 2009, Anjos et al. 2005, Anjos and Vannelli 2008, Anjos and Yen 2009, Duff et al. 1989, Heragu and Kusiak 1991, Hungerländer and Rendl 2013, Nugent et al. 1968, Obata 1981, Sarker 1989, Simmons 1969, Yu and Sarker 2003) for the SRFLP as well as the Single-Row Equidistant Facility Layout Problem and the Minimum Linear Arrangement Problem – two of its subproblems. We limited our experiments to instances with fewer than 50 departments. The same width strategy as for the PSP was applied.

The models used to encode these problems within the DD framework are described in the Appendix. For all the experiments, the best results were obtained with an LEL cutset for the classical B&B (B&B) and with an FC for B&B with caching (B&B+C). It is worth mentioning that the baseline approach already uses duplicate state detection in the *Fringe*. Concerning the heuristics of the approach, the node with the highest upper bound is selected first in the B&B loop. Moreover, nodes with the smallest longest path value are respectively deleted or merged in restricted and relaxed DDs compilations. For each problem, we tested multiple maximum widths by varying the parameter $\alpha \in \{1, 10, 100\}$. The DD approach was implemented in Rust, based on DDO (Gillard et al. 2021b) which is a generic library for DD-based optimization. The source code and all benchmark instances are available online (Coppé et al. 2024). In order to showcase the effectiveness of the DD approach, it is compared against a state-of-the-art MIP model for each problem respectively introduced by (Hungerländer and Truden 2018) as Formulation (1) for the TSPTW, by (Pochet and Wolsey 2006) as PIG-A-3 for the PSP and by (Amaral 2009) for the SRFLP. The MIP models were solved with Gurobi 9.5.2 (Gurobi Optimization, LLC 2022) with
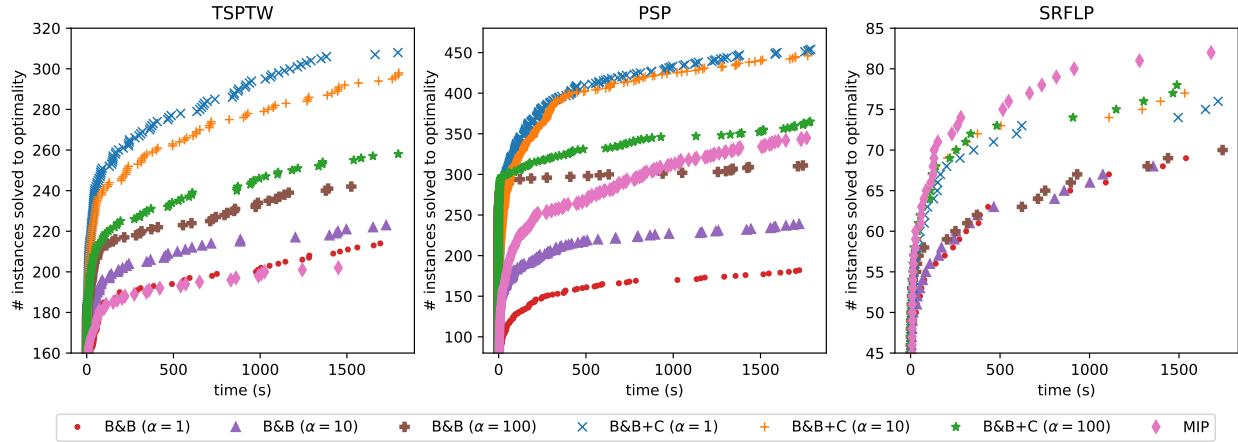
Figure 7   **Number of instances solved over time by B&B, B&B+C and Gurobi for three different problems. The DD approaches use different maximum widths with $\alpha \in \{1, 10, 100\}$.**
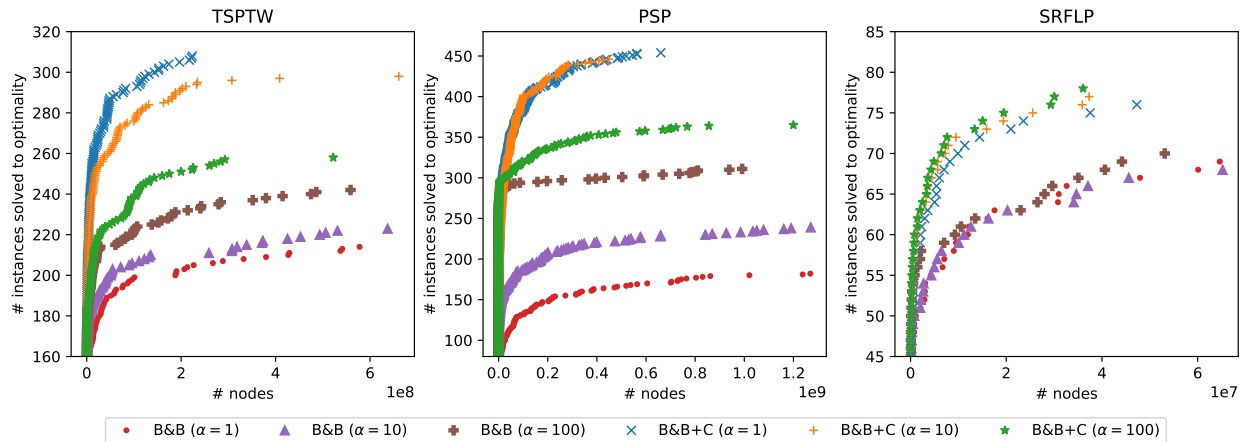

Figure 8   **Number of instances solved by B&B and B&B+C with respect to the number of DD nodes expanded for three different problems. Different maximum widths were used with $\alpha \in \{1, 10, 100\}$.**

the default settings. For all problems, the solvers were given 1800 seconds to solve each instance to optimality, using a single thread.

6.1.   Impact of the Caching Mechanism

Figures 7 and 8 show the results of these experiments, respectively in terms of computation time and number of DD nodes expanded during the search. This measure of nodes expanded accounts for all nodes expanded during top-down compilations of restricted and relaxed DDs. Each graph presents the total number of instances solved under any given time or nodes expanded. For each of the studied problems and whatever the maximum width used, B&B+C is able to solve significantly more instances than B&B within the time limit. This speedup is directly linked to a reduction of the number of DD nodes expanded, as can be seen Figure 8. This confirms our intuition that a lot of work is unnecessarily repeated by

B&B and shows that the pruning techniques introduced in this paper help neutralize much of this problem. Moreover, if we look at the performance achieved using different maximum widths for the TSPTW and the PSP, one can notice that for B&B, increasing the width helps to solve more instances. Indeed, larger DDs allow stronger bounds to be derived and instances to be closed more quickly, as long as they are not too expensive to compile. Yet, the opposite observation can be made about B&B+C: using smaller maximum widths results in solving more instances for the TSPTW and the PSP. This perfectly captures the double benefit of the *Cache*. The strength of dominance and pruning thresholds allow discarding many transitions during the compilation of approximate DDs and avoid repeating previous work. As a result, narrower DDs can explore and prune the search space as fast while being much cheaper to generate, and sometimes identify uninteresting parts earlier in the search. Again, this observation about Figure 7 can be validated on Figure 8 where B&B+C manages to solve significantly more instances within a given maximum number of nodes when using narrower DDs. Another benefit of B&B+C is that it fully exploits the potential of FCs: as all non-improving transitions are blocked by the *Cache*, it is only natural to use the deepest possible exact cutset. The same cannot be said in the case of B&B because frontier cutsets usually contain many nodes, some of which are the parents of others, and this only exacerbates the caveats mentioned in Section 3.

For the SRFLP, the difference of performance obtained by varying the maximum width is not as significant as for the other two problems. This is probably because the problem is unconstrained and the relaxation is not very tight. As a result, the thresholds computed with small width DDs may not be strong enough to avoid exploring nodes that are explored by DDs of larger widths. As a matter of fact, slightly fewer instances are solved using narrow DDs for a same number of DD nodes expanded. In this case, wider DDs may derive better bounds and thresholds and perform slightly better. Still, using a *Cache* with DDs of any width helps to close more instances than without it.

To put these results into perspective, we compare them on Figure 7 to those obtained with MIP models solved with Gurobi. For the TSPTW, Gurobi could only solve 202 instances, which is less than the worst-performing configuration of DDO. For the PSP, B&B+C with $\alpha = 1$ could solve 454 of the generated instances while Gurobi solved 345 of them. On the other hand, B&B with $\alpha = 100$ solved only 311 instances. The addition
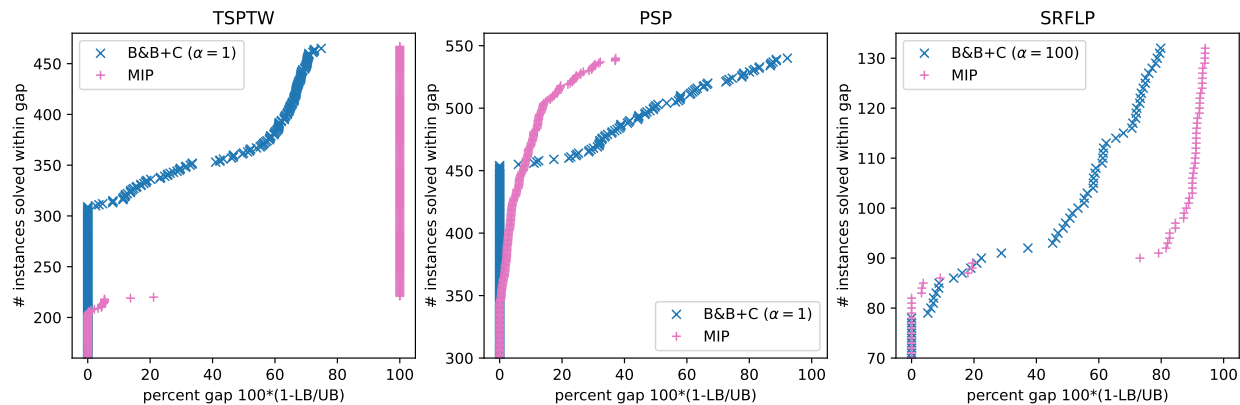
Figure 9   **Number of instances solved by the best performing B&B+C and Gurobi within a given optimality gap for each of the three problems.**

of the *Cache* thus helped improve DD-based B&B by a large margin and allowed outperforming Gurobi on this set of instances. For the SRLFP, B&B and B&B+C with $\alpha = 100$ were able to solve respectively 12 and 4 instances less than Gurobi within the given time budget. Therefore, even if the best results are achieved with the MIP model, using the *Cache* closed much of the gap that separates the two techniques.

Slightly different observations can be made through Figure 9, which compares the number of instances solved within a given optimality gap by B&B+C with the best choice of α for each problem and by Gurobi. For the TSPTW, it appears that Gurobi fails to find an admissible solution for nearly all unsolved instances, which results in an optimality gap of 100%. B&B+C does not suffer from this problem, yet the optimality gap for unsolved instances is quite large in most cases. Even though Gurobi solved fewer instances for the PSP, it manages to solve all instances within a 40% optimality gap and most of them within a 20% optimality gap. The optimality gap obtained by B&B+C for unsolved instances is much poorer. Surprisingly, B&B+C achieves tighter optimality gaps for hard SRFLP instances: all of them are within 80% while for Gurobi, many unsolved instances lie outside this optimality gap. In general, the optimality gaps obtained for unsolved instances with B&B+C seem to deteriorate very rapidly as the instances get harder. One would therefore be tempted to infer that a lot of computational effort is still required to solve the remaining instances – even those with a relatively small gap. However, we believe that, in the case of B&B+C, the optimality gap and the remaining computational effort are not perfectly correlated. Indeed, having filled the *Cache* with thresholds makes it easier to process the remaining nodes.
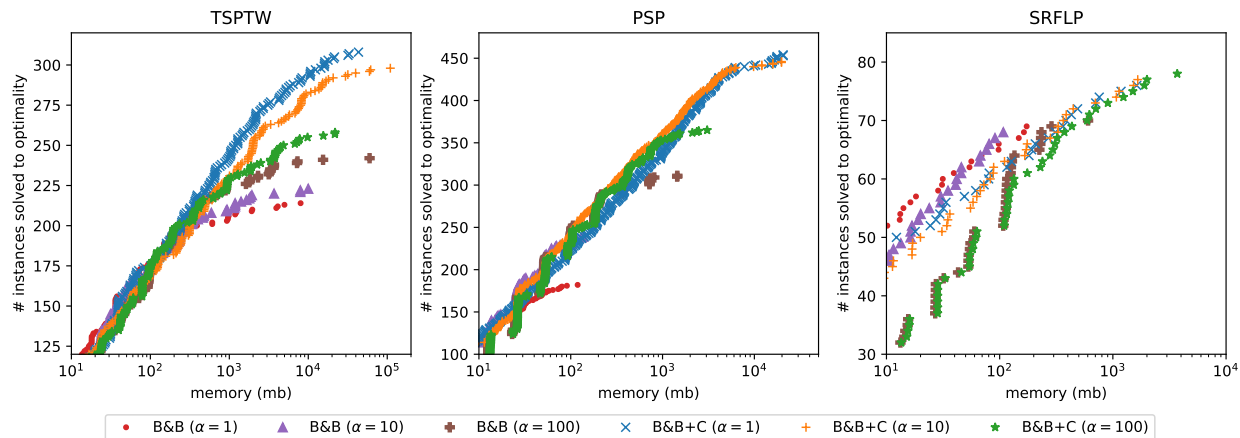
Figure 10 **Number of instances solved by B&B and B&B+C for three different problems and for multiple maximum widths obtained with $\alpha \in \{1, 10, 100\}$, with respect to the peak amount of memory used.**

6.2. Memory Analysis

The performance improvements discussed in Section 6.1 do not come completely for free, as the *Cache* must store all the thresholds computed during the B&B algorithm. It is thus important to study the impact of this technique on the memory consumption of the algorithm. For every instance solved by each algorithm, the peak amount of memory used during the execution was recorded. Figure 10 shows the number of instances solved using a given maximum amount of memory. It appears that the memory consumption of B&B and B&B+C are of the same order of magnitude. Actually, for the TSPTW and the PSP, as the instances get harder, B&B+C even starts solving more instances with the same peak amount of memory. Thus, even in terms of memory consumption, the cost of maintaining the *Cache* seems to be compensated by its pruning effect which causes DDs to be sparser.

7. Conclusion

In this paper, we first discussed how the DD-based B&B algorithm tends to repeatedly explore overlapping parts of the search space. Then, we introduced dominance and pruning thresholds with the intention of overcoming this limitation. The former propagate dominance relations between partial solutions obtained within approximate DDs while the latter allow the approach to be combined and strengthened by the pruning performed by LocBs and RUBs. Both types of thresholds are used in a single pruning mechanism that is able to discard many nodes associated with previously visited DP states during subsequent DD compilations. Finally, we presented experimental results that clearly show the impact of the techniques introduced: B&B with caching vastly outperforms the classical B&B algorithm

in terms of instances solved on the three discrete optimization problems studied in the paper, and compares well to state-of-the-art MIP models. Furthermore, the experiments showed that the memory consumption induced by the *Cache* was either acceptable or even overcompensated by its pruning effect.

In general, we expect that using the *Cache* will be beneficial for solving problems whenever the theoretical search tree of the DP model comprises isomorphic subtrees, which are normally superimposed in the corresponding exact DD. That is, we expect that the *Cache* to be profitable whenever pure DP would be an efficient – albeit impractical because of memory limitations – method for solving the problem at hand.

DD-based B&B is known to parallelize very well and it is one of its main advantages compared to other approaches. The improvements described in this paper are based on a shared data structure that avoids successive DD compilations from being completely blind to previously done work. Therefore, concurrent DD compilations are not fully independent anymore and require some synchronization to access and update the *Cache*. As future work, we thus aspire to optimize the pruning capacities of the *Cache* while minimizing contention in the data structure.

References

Akers SB (1978) Binary decision diagrams. *IEEE Transactions on Computers* 27(06):509–516.

Amaral AR (2006) On the exact solution of a facility layout problem. *European Journal of operational research* 173(2):508–518.

Amaral AR (2008) An exact approach to the one-dimensional facility layout problem. *Operations Research* 56(4):1026–1033.

Amaral AR (2009) A new lower bound for the single row facility layout problem. *Discrete Applied Mathematics* 157(1):183–190.

Andersen HR, Hadzic T, Hooker JN, Tiedemann P (2007) A constraint store based on multivalued decision diagrams. *International Conference on Principles and Practice of Constraint Programming*, 118–132 (Springer).

Anjos MF, Kennings A, Vannelli A (2005) A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization* 2(2):113–122.

Anjos MF, Vannelli A (2008) Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing* 20(4):611–617.

Anjos MF, Yen G (2009) Provably near-optimal solutions for very large single-row facility layout problems. *Optimization Methods & Software* 24(4-5):805–817.

Ascheuer N (1996) *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems.* Ph.D. thesis, University of Technology Berlin.

Becker B, Behle M, Eisenbrand F, Wimmer R (2005) BDDs in a branch and cut framework. *International Workshop on Experimental and Efficient Algorithms*, 452–463 (Springer).

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2012) Variable ordering for the application of bdds to the maximum independent set problem. *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*, 34–49 (Springer).

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2014a) Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing* 26(2):253–268.

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2016) Discrete optimization with decision diagrams. *INFORMS Journal on Computing* 28(1):47–66.

Bergman D, Cire AA, van Hoeve WJ, Yunes T (2014b) BDD-based heuristics for binary optimization. *Journal of Heuristics* 20(2):211–234.

Bryant RE (1986) Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on* 100(8):677–691.

Cappart Q, Bergman D, Rousseau LM, Prémont-Schwarz I, Parjadis A (2022) Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing* 34(5):2552–2570.

Castro MP, Cire AA, Beck JC (2022) Decision diagrams for discrete optimization: A survey of recent advances. *INFORMS Journal on Computing* 34(4):2271–2295.

Cire AA, van Hoeve WJ (2013) Multivalued decision diagrams for sequencing problems. *Operations Research* 61(6):1411–1428.

Clarke EM, Grumberg O, Long DE (1994) Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)* 16(5):1512–1542.

Coppé V, Gillard X, Schaus P (2022) Solving the Constrained Single-Row Facility Layout Problem with Decision Diagrams. Solnon C, ed., *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, 14:1–14:18 (Schloss Dagstuhl – Leibniz-Zentrum für Informatik).

Coppé V, Gillard X, Schaus P (2024) Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection. URL http://dx.doi.org/10.1287/ijoc.2022.0340.cd, available for download at https://github.com/INFORMSJoC/2022.0340.

Duff IS, Grimes RG, Lewis JG (1989) Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS)* 15(1):1–14.

Dumas Y, Desrosiers J, Gelinas E, Solomon MM (1995) An optimal algorithm for the traveling salesman problem with time windows. *Operations research* 43(2):367–371.

Gendreau M, Hertz A, Laporte G, Stan M (1998) A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research* 46(3):330–335.

Gillard X, Coppé V, Schaus P, Cire AA (2021a) Improving the filtering of branch-and-bound mdd solver. *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 231–247 (Springer).

Gillard X, Schaus P (2022) Large neighborhood search with decision diagrams. De Raedt L, ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 4754–4760 (International Joint Conferences on Artificial Intelligence Organization).

Gillard X, Schaus P, Coppé V (2021b) DDO, a generic and efficient framework for mdd-based optimization. *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 5243–5245.

Gurobi Optimization, LLC (2022) Gurobi Optimizer Reference Manual. URL https://www.gurobi.com.

Hachtel GD, Somenzi F (1997) A symbolic algorithms for maximum flow in 0-1 networks. *Formal Methods in System Design* 10(2):207–219.

Hadžić T, Hooker JN (2006) Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University.

Hadžić T, Hooker JN (2007) Cost-bounded binary decision diagrams for 0-1 programming. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 84–98 (Springer).

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Held M, Karp RM (1962) A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics* 10(1):196–210.

Heragu SS, Kusiak A (1991) Efficient models for the facility layout problem. *European Journal of Operational Research* 53(1):1–13.

Hooker JN (2013) Decision diagrams and dynamic programming. *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 94–110 (Springer).

Hu AJ (1995) *Techniques for efficient formal verification using binary decision diagrams*. Ph.D. thesis, Stanford University, Department of Computer Science.

Hungerländer P, Rendl F (2013) A computational study and survey of methods for the single-row facility layout problem. *Computational Optimization and Applications* 55(1):1–20.

Hungerländer P, Truden C (2018) Efficient and easy-to-implement mixed-integer linear programs for the traveling salesperson problem with time windows. *Transportation research procedia* 30:157–166.

Kam TYk, Brayton RK (1990) *Multi-valued decision diagrams* (Electronics Research Laboratory, College of Engineering, University of California).

Karahalios A, van Hoeve WJ (2022) Variable ordering for decision diagrams: A portfolio approach. *Constraints* 27(1):116–133.

Lai YT, Pedram M, Vrudhula SBK (1994) EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(8):959–975.

Langevin A, Desrochers M, Desrosiers J, Gélinas S, Soumis F (1993) A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks* 23(7):631–640.

Lee CY (1959) Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal* 38(4):985–999.

Minato Si (1995) *Binary decision diagrams and applications for VLSI CAD*, volume 342 (Springer Science & Business Media).

Nugent CE, Vollmann TE, Ruml J (1968) An experimental comparison of techniques for the assignment of facilities to locations. *Operations research* 16(1):150–173.

Obata T (1981) The quadratic assignment problem: Evaluation of exact and heuristic algorithms. *Transportation Research Part A: General* 15(4):346.

Ohlmann JW, Thomas BW (2007) A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing* 19(1):80–90.

Pesant G, Gendreau M, Potvin JY, Rousseau JM (1998) An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science* 32(1):12–29.

Pochet Y, Wolsey LA (2006) *Production planning by mixed integer programming*, volume 149 (Springer).

Potvin JY, Bengio S (1996) The vehicle routing problem with time windows part II: genetic search. *INFORMS Journal on Computing* 8(2):165–172.

Rudich I, Cappart Q, Rousseau LM (2022) Peel-And-Bound: Generating Stronger Relaxed Bounds with Multivalued Decision Diagrams. Solnon C, ed., *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, 35:1–35:20 (Schloss Dagstuhl – Leibniz-Zentrum für Informatik).

Sarker BR (1989) *The amoebic matrix and one-dimensional machine location problems* (Texas A&M University).

Simmons DM (1969) One-dimensional space allocation: an ordering algorithm. *Operations Research* 17(5):812–826.

Wegener I (2000) Branching programs and binary decision diagrams: Theory and applications. *Discrete Applied Mathematics* .

Yu J, Sarker BR (2003) Directional decomposition heuristic for a linear machine-cell location problem. *European Journal of Operational Research* 149(1):142–184.

Appendix. Models Used in the Experimental Study

The three discrete optimization problems for which we present models in this appendix are minimization problems. The DD-based optimization framework remains applicable but the terminology must be adapted. For instance, the optimal solution of a minimization problem is given by the *shortest* path in the corresponding exact DD. Moreover, restricted and relaxed DDs respectively provide upper and lower bounds. Similarly, rough upper bounds are replaced with *rough lower bounds* (RLBs).

A. TSPTW

The TSPTW is a variant of the well-known Traveling Salesman Problem (TSP) where the cities are replaced by a set of customers $N = \{0, \ldots, n-1\}$ that must each be visited during a given time window $\mathcal{TW}_i = (e_i, l_i)$. The first customer is a dummy customer that represents the depot where the salesman must begin and end its tour. As in the classical TSP, we are given a symmetrical distance matrix D that contains in each entry (i, j) the distance D_{ij} separating customers i and j. In addition to the time windows controlling the earliest and latest time when the salesman can visit the customers, the horizon H limits the time at which the salesman must return to the depot.

A.1. DP Model

The DP model presented here extends the one introduced by Held and Karp (1962) for the TSP, which successively decides the next customer to visit and defines states with the set of customers that still must be visited along with the current position of the salesman. For the TSPTW, this state representation is extended to a tuple $\langle L, t, M, P \rangle$, where L (locations) and t (time) respectively represent the set of locations where the salesman might be and the minimum time to reach one of these locations. The set M (must) contains the visits that must still be completed, while P (possible) are the visits that can possibly be done in addition to the ones in M. This set of possible visits P is useful for tightening the relaxation of the relaxed MDD, as explained next. M and P are disjoint sets of customers that must or might be visited in order to close the tour even in case of relaxed states. We now describe the full DP model:

- Control variables: $x_j \in N$ with $j \in N$ decides which customer is visited in j-th position.
- State space: $S = \{\langle L, t, M, P \rangle \mid L, M, P \subseteq N, M \cap P = \emptyset, 0 \leq t \leq H\}$. The root state is $\hat{r} = \langle \{0\}, 0, N, \emptyset \rangle$ and the terminal states are all states $\langle \{0\}, t, \emptyset, P \rangle$ with $0 \leq t \leq H$.
- Transition functions:

$$
t_j(s^j, x_j) = \begin{cases} \langle t_j^L(s^j, x_j), t_j^t(s^j, x_j), t_j^M(s^j, x_j), t_j^P(s^j, x_j) \rangle, & \begin{array}{l} \text{if } x_j \in s^j.M \\ \text{and } s^j.t + \min_{i \in s^j.L} D_{ix_j} \leq l_{x_j}, \end{array} \\ \langle t_j^L(s^j, x_j), t_j^t(s^j, x_j), t_j^M(s^j, x_j), t_j^P(s^j, x_j) \rangle, & \begin{array}{l} \text{if } x_j \in s^j.P \\ \text{and } s^j.t + \min_{i \in s^j.P} D_{ix_j} \leq l_{x_j} \\ \text{and } |M| < n - j, \end{array} \\ \hat{0}, & \text{otherwise.} \end{cases}
\tag{10}
$$

where

$$
\begin{aligned}
t_j^L(s^j, x_j) &= \{x_j\} \\
t_j^t(s^j, x_j) &= \max\left\{e_{x_j}, s^j.t + \min_{i \in s^j.P} D_{ix_j}\right\} \\
t_j^M(s^j, x_j) &= s^j.M \setminus \{x_j\} \\
t_j^P(s^j, x_j) &= s^j.P \setminus \{x_j\}
\end{aligned}
$$

In Equation (10), transitions are only allowed if they respect the time window constraint l_{x_j}. Since the salesman can be at multiple different positions in relaxed states, the minimum distance is used to compute the arrival time. The second condition in Equation (10) ensures that no customers are selected from the possible set P when there remains enough stops only for the customers in the must-set M.

- Transition value functions: $h_j(s^j, x_j) = \min_{i \in s^j.L} D_{ix_j}$.

- Root value: $v_r = 0$.

A.2. Relaxation

The merging operator is defined as follows:

$$
\oplus(\mathcal{M}) = \langle \oplus_L(\mathcal{M}), \oplus_t(\mathcal{M}), \oplus_M(\mathcal{M}), \oplus_P(\mathcal{M}) \rangle \tag{11}
$$

where

$$
\begin{aligned}
\oplus_L(\mathcal{M}) &= \bigcup_{s \in \mathcal{M}} s.L \\
\oplus_t(\mathcal{M}) &= \min_{s \in \mathcal{M}} s.t \\
\oplus_M(\mathcal{M}) &= \bigcap_{s \in \mathcal{M}} s.M \\
\oplus_P(\mathcal{M}) &= \left(\bigcup_{s \in \mathcal{M}} s.M \cup s.P\right) \setminus \left(\bigcap_{s \in \mathcal{M}} s.M\right)
\end{aligned}
$$

These operators ensure that all transitions are preserved and that the transition values are not increased. Note that the M and P sets are disjoint so that P contains customers that must or might be visited in some states, but no customers that must be visited in all states. The relaxed transition value operator is simply the identity function $\Gamma_{\mathcal{M}}(v, u) = v$.

A.3. Rough Lower Bound

The RLB shown in Equation (12) computes an estimate of the remaining distance to cover in order for the salesman to finish his tour and return to the depot. For all customers i in M, the distance of the cheapest edge incident to i is added. Then, $n - j - |M|$ customers from P must be visited in order to complete the tour. Since this is a lower bound computation, we create a vector C containing the cheapest edge incident to each customer in P and select the $n - j - |M|$ shortest ones. We denote by $X_{<,i}$ the i-th smallest element from a vector X.

$$
\underline{v}_{rlb}(s^j) = \sum_{i \in s^j.M} cheapest_i + \sum_{i=1}^{n-j-|M|} C_{<,i} \tag{12}
$$

A check is also added to detect states where it is impossible to visit independently all of the customers from M and at least $n - j - |M|$ customers from P, given the current time and the time windows of the remaining customers. Plus, if the estimate provided by Equation (12) prevents the salesman from returning to the depot within the time constraint, a RLB of $-\infty$ is returned, effectively discarding the corresponding states.

B. PSP

The PSP is a single-machine planning problem where one has to find a production schedule that satisfies a set of demands at minimal cost. Formally, there is a set of item types $I = \{0, \ldots, n-1\}$ that are associated with a stocking cost S_i. When the machine switches the production from item type i to j, it incurs a changeover cost C_{ij}. The planning spans for a given time horizon H. For each time period $0 \leq p < H$ of this horizon, $Q_p^i \in \{0, 1\}$ indicates whether an item of type i must be delivered. When an item i is produced at time period p_1 and delivered at period p_2, stocking the item costs $S_i(p_2 - p_1)$.

To give a better understanding of the problem, we hereby recall the MIP model denoted PIG-A-1 in (Pochet and Wolsey 2006). Variables $x_p^i \in \{0, 1\}$ decide whether an item of type i is produced at time period p. On the other hand, variables $y_p^i \in \{0, 1\}$ decide whether the machine is ready to produce an item of type i at time period p. Indeed, the machine can be idle at certain periods. Variables $q_p^i \in \mathbb{N}_0$ accumulate the quantity of items of type i stored at period p. Finally, variables $\chi_p^{i,j} \in \{0, 1\}$ capture a changeover between item types i and j at period p.

$$\min \sum_{i \in I} \sum_{p=0}^{H-1} S_i q_p^i + \sum_{i,j \in I} \sum_{p=0}^{H-1} C_{ij} \chi_p^{i,j} \tag{13}$$

$$q_{p-1}^i + x_p^i = q_p^i + Q_p^i \qquad\qquad \forall i \in I, 0 \leq p < H \tag{14}$$

$$q_{-1}^i = 0 \qquad\qquad \forall i \in I \tag{15}$$

$$x_p^i \leq y_p^i \qquad\qquad \forall i \in I, 0 \leq p < H \tag{16}$$

$$\sum_{i \in I} y_p^i = 1 \qquad\qquad \forall 0 \leq p < H \tag{17}$$

$$\chi_p^{i,j} \geq y_{p-1}^i + y_p^i - 1 \qquad\qquad \forall i, j \in I, 0 < p < H \tag{18}$$

Equation (14) models the stocking of the items and consumes them when needed, with the quantities initialized by Equation (15). Then, Equation (16) ensures that the machine is in the correct mode to produce an item and Equation (17) allows only one mode for each time period. Finally, Equation (18) sets the correct value for the changeover variables.

B.1. DP Model

The DP model introduced in (Gillard and Schaus 2022) is extended to allow relaxation as well as to cover instances where the machine can be idle at some time periods. It proceeds by taking decisions starting from the end of the planning horizon and working backwards to avoid expanding any infeasible states. For clarity, variables x_j decide the type of item produced at period j. This implies that the reverse variable ordering $x_{H-1}, \ldots, x_0$ is used. Before pursuing the modeling of the PSP in the DD framework, let us define P_r^i as the time period at which an item of type i must be delivered when there remains r of them to produce, i.e., $P_r^i = \min \left\{ 0 \leq q < H \mid \sum_{p=0}^q Q_p^i \geq r \right\}$ for all $i \in N, 0 \leq r \leq \sum_{0 \leq p < H} Q_p^i$.

States $\langle i, R \rangle$ where i is the item type produced at the next time period and R is a vector where R_i gives the remaining number of demands to satisfy for item type i. We pose $N' = N \cup \{\bot\}$ where $\bot$ is a dummy item type used to represent periods where the machine is idle.

- Control variables: $x_j \in N'$ with $0 \leq j < H$ decides which item type is produced at time period j.

- State space: $S = \left\{ s \mid s.i \in N', \forall i \in N, 0 \leq s.R_i \leq \sum_{0 \leq p < H} Q_p^i \right\}$. The root state is $\hat{r} = \left\langle \perp, (\sum_{0 \leq p < H} Q_p^0, \ldots, \sum_{0 \leq p < H} Q_p^{n-1}) \right\rangle$ and the terminal states are of the form $\langle i, (0, \ldots, 0) \rangle$ with $i \in N'$.

- Transition functions:

$$
t_j(s^j, x_j) = \begin{cases} \langle t_j^i(s^j, x_j), t_j^R(s^j, x_j) \rangle, & \text{if } x_j \neq \perp \text{ and } s^j.R_{x_j} > 0 \text{ and } j \leq P_{s^j.R_{x_j}}^{x_j}, \\ \langle t_j^i(s^j, x_j), t_j^R(s^j, x_j) \rangle, & \text{if } x_j = \perp \text{ and } \sum_{i \in N} s^j.R_i < j+1, \\ \hat{0}, & \text{otherwise.} \end{cases} \tag{19}
$$

where

$$
\begin{aligned}
t_j^i(s^j, x_j) &= \begin{cases} x_j, & \text{if } x_j \neq \perp \\ s^j.i, & \text{otherwise.} \end{cases} \\
t_j^R(s^j, x_j) &= \begin{cases} (s^j.R_0, \ldots, s^j.R_{x_j} - 1, \ldots, s^j.R_{n-1}), & \text{if } x_j \neq \perp \\ s^j.R, & \text{otherwise.} \end{cases}
\end{aligned}
$$

The first condition in Equation (19) states that in order to schedule a production of item type x_j at state s^j, there must remain at least one demand of that item type to satisfy, and due at period j or after. The second one ensures that idle periods can only be scheduled when the total remaining demand is smaller than the number of remaining time periods.

- Transition value functions:

$$
h_j(s^j, x_j) = \begin{cases} C_{x_j s^j.i}, & \text{if } x_j \neq \perp \text{ and } s^j.i \neq \perp \\ 0, & \text{otherwise.} \end{cases} + \begin{cases} S_{x_j} \cdot (j - P_{s^j.R_{x_j}}^{x_j}), & \text{if } x_j \neq \perp \\ 0, & \text{otherwise.} \end{cases}
$$

- Root value: $v_r = 0$.

B.2. Relaxation

The merging operator is defined as follows:

$$
\oplus(\mathcal{M}) = \left\langle \perp, (\min_{s \in \mathcal{M}} s.R_0, \ldots, \min_{s \in \mathcal{M}} s.R_{n-1}) \right\rangle. \tag{20}
$$

As merged states might disagree on the item type produced before, it is reset to $\perp$. For each item type, the minimum remaining number of items to produce is computed, meaning that all demands satisfied by a least one state are considered satisfied in the merged state. As for the TSPTW, the relaxed transition value operator is the identity function $\Gamma_{\mathcal{M}}(v, u) = v$.

B.3. Rough Lower Bound

When the changeover costs are ignored, the PSP falls under the Wagner-Whitin conditions (Pochet and Wolsey 2006) that allow computing the optimal stocking cost. Conversely, if the stocking costs and the delivery constraints are omitted, the PSP can be reduced to the TSP. Therefore, a valid lower bound on the total changeover cost to produce a remaining set of items is to take the total weight of a Minimum Spanning Tree computed on the graph of changeover costs limited to item types that still need to be produced. The optimal weight for all these spanning trees can be precomputed because the number of items is usually small. As there is no overlap between the two lower bounds described, the RLB for the PSP can sum their individual contributions to obtain a stronger lower bound.

C.   SRFLP

The SRFLP is an ordering problem that aims to place a set of departments $N = \{0, \ldots, n-1\}$ on a line. Each department $i \in N$ has a positive length L_i, and the connection between each pair of departments $i, j \in N$ is described by a positive traffic intensity C_{ij}. The goal is to find a bijection $\pi : N \to N$ that maps each department to a position on the line, while minimizing the total distance covered in the facility:

$$SRFLP(\pi) = \sum_{k=0}^{n-1} L_k \sum_{\substack{i=0 \\ \pi(i)<\pi(k)}}^{n-1} \sum_{\substack{j=0 \\ \pi(k)<\pi(j)}}^{n-1} C_{ij} + \underbrace{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} C_{ij} \frac{L_i + L_j}{2}}_{K}. \tag{21}$$

The second term of Equation (21) is a constant that does not depend on the ordering π, which is usually denoted K. The modeling used in this paper is similar to the one presented in (Coppé et al. 2022) except that merging is included. The following definitions were thus adapted to cover states where each department must, might or might not be added to the ordering, similarly to the TSPTW modeling explained in Section A.

C.1.   DP Model

The DP model works as follows: departments are added one by one on the line from left to right. The states are defined by tuples $\langle M, P, C \rangle$ where M (must) and P (possible) are the sets of departments that respectively must and might be placed on the line. C (cuts) is a vector that accumulates the *cut value* of each department. The cut value of a department is the total traffic intensity incident to this department, coming from placed departments. Formally, the DP model is defined as follows:

- Control variables: $x_j \in N$ with $j \in N$ decides which department is placed at position j on the line.
- State space: $S = \{\langle M, P, C \rangle \mid M, P \subseteq N, M \cap P = \emptyset\}$. The root state is $\hat{r} = \langle N, \emptyset, (0, \ldots, 0)\rangle$ and terminal states $\langle M, P, C \rangle$ must only verify $M = \emptyset$.
- Transition functions:

$$t_j(s^j, x_j) = \begin{cases} \left\langle t_j^M(s^j, x_j), t_j^P(s^j, x_j), t_j^C(s^j, x_j)\right\rangle, & \text{if } x_j \in s^j.M, \\ \left\langle t_j^M(s^j, x_j), t_j^P(s^j, x_j), t_j^C(s^j, x_j)\right\rangle, & \text{if } x_j \in s^j.P \text{ and } |M| < n-j, \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

where

$$t_j^M(s^j, x_j) = s^j.M \setminus \{x_j\}$$
$$t_j^P(s^j, x_j) = s^j.P \setminus \{x_j\}$$
$$t_j^C(s^j, x_j) = (C_0, \ldots, C_{n-1}) \text{ with } C_i = \begin{cases} s^j.C_i + C_{x_j i}, & \text{if } x_j \in (s^j.M \cup s^j.P) \setminus \{x_j\}, \\ 0, & \text{otherwise.} \end{cases}$$

The second condition in Equation (22) ensures that no departments can be added from $s^j.P$ when there remains space only for the departments in $s^j.M$.

- Transition value functions: the contribution of place department x_j at position j is its length L_{x_j} multiplied by the sum of the cut values of departments that will be located to its right. These departments are contained in $s^j.M$ and in case the state is relaxed, in $s^j.P$. However, there might be more departments in $s^j.M \cup s^j.P$ than needed to complete the ordering. Selecting the $n - j - |s^j.M|$ smallest cut values among departments of $s^j.P$ will result in a lower bound on the value obtained by placing any subset of departments in $s^j.P$. For a transition from state s^j with decision x_j, we also define $C^{P,x_j} = (s^j.C_i \mid i \in s^j.P \setminus \{x_j\})$ as the vector of cut values of departments in $s^j.P \setminus \{x_j\}$. Then, the transition value functions are:

$$h_j(s^j, x_j) = L_{x_j} \left(\sum_{i \in s^j.M \setminus \{x_j\}} s^j.C_i + \sum_{i=1}^{n-j-|s^j.M|} C_{<,i}^{P,x_j} \right).$$

- Root value: $v_r = K$ (see Equation (21)).

C.2. Relaxation

The merging operator is defined as follows:

$$\oplus(\mathcal{M}) = \langle \oplus_M(\mathcal{M}), \oplus_P(\mathcal{M}), \oplus_C(\mathcal{M}) \rangle \tag{23}$$

where

$$
\begin{aligned}
\oplus_M(\mathcal{M}) &= \bigcap_{s \in \mathcal{M}} s.M \\
\oplus_P(\mathcal{M}) &= \left(\bigcup_{s \in \mathcal{M}} s.M \cup s.P \right) \setminus \left(\bigcap_{s \in \mathcal{M}} s.M \right) \\
\oplus_C(\mathcal{M}) &= (C_0, \ldots, C_{n-1}) \qquad \text{with } C_i = \min_{s \in \mathcal{M}} \begin{cases} s.C_i, & \text{if } i \in s.M \cup s.P, \\ \infty, & \text{otherwise.} \end{cases}
\end{aligned}
$$

The M and P sets are aggregated just as in the TSPTW relaxation of Section A.2. Concerning the cut values, $\oplus_C$ simply keeps for each department the minimum cut value among the states that still must or might place the department considered. Once again, the relaxed transition value operator is the identity function $\Gamma_{\mathcal{M}}(v, u) = v$.

C.3. Rough Lower Bound

The RLB used for the SRFLP is divided into separate bounds: the first computes the optimal ordering of the remaining departments with respect to their cut values only, while the second estimates the arrangement cost of the remaining departments, ignoring the departments that have already been placed. The *first-generation bound* introduced by Simmons (1969) allows computing the ordering that minimizes the contribution of cut values with respect to departments placed on the left. In that case, the optimal arrangement is given by ordering the departments by decreasing cut-to-length ratios. The only obstacle to using this lower bound is that for a state s^j all departments from $s^j.M$ will be included in the arrangement but only $n^P = n - j - n^M$ from $s^j.P$, with $n^M = |s^j.M|$. As the optimal subset of departments to select from $s^j.P$ is unknown, we create n^P artificial departments with the n^P shortest lengths and smallest cut values among the departments in $s^j.P$. A vector of ratios that contain tuples $\langle r, c, l \rangle$ is created for departments in $s^j.M$ and $s^j.P$, where $r = \frac{c}{l}$, c is a cut value and l is a length. As stated by Equation (26), the k-th shortest length is combined with the $(n^P - k + 1)$-th smallest cut value as to minimize the lower bound formula given by Equation (29).

$$L^P = (L_k \mid k \in s^j.P) \tag{24}$$

$$C^P = (L_k \mid k \in s^j.P) \tag{25}$$

$$R^P = \left(\left\langle \frac{C^P_{<, n^P - k + 1}}{L^P_{<, k}}, C^P_{<, n^P - k + 1}, L^P_{<, k} \right\rangle \mid k = 1, \ldots, n^P \right) \tag{26}$$

$$R^M = \left(\left\langle \frac{s^j.C_k}{L_k}, s^j.C_k, L_k \right\rangle \mid k \in s^j.M \right) \tag{27}$$

$$R = concat(R^M, R^P) \tag{28}$$

The concatenation of both vectors of ratios is then used to compute the lower bound as in (Simmons 1969).

$$LB_{cut}(s^j) = \sum_{k=1}^{n-j} R_{>,k}.c \sum_{l=1}^{k-1} R_{>,l}.l \tag{29}$$

The second part of the RLB is a lower bound on the arrangement cost of $n^M + n^P$ departments from $s^j.M$ and $s^j.P$. First, a subset of traffic intensities is selected: all those connecting two departments in $s^j.M$ and then the $n^M n^P$ lowest of those connecting one department from $s^j.M$ to one in $s^j.P$ and the $\frac{n^P(n^P-1)}{2}$ lowest of those connecting two departments in $s^j.P$. Second, the n^P shortest department lengths from $s^j.P$ are combined with those from $s^j.M$.

$$T^{M,M} = \big(C_{kl} \mid k,l \in s^j.M, k < l\big) \tag{30}$$

$$T^{M,P} = \big(C_{kl} \mid k \in s^j.M, l \in s^j.P\big) \tag{31}$$

$$T^{P,P} = \big(C_{kl} \mid k,l \in s^j.P, k < l\big) \tag{32}$$

$$T = concat \begin{pmatrix} T^{M,L}, \\ T^{M,P}_<[1\ldots n^M n^P], \\ T^{P,P}_<[1\ldots \frac{n^P(n^P-1)}{2}] \end{pmatrix} \tag{33}$$

$$L^M = \big(L_k \mid k \in s^j.M\big) \tag{34}$$

$$L = concat(L^M, L^P_<[1\ldots n^P]) \tag{35}$$

Given the traffic and length vectors T and L, a lower bound on the arrangement cost can then be computed by multiplying the traffic intensities by an optimistic distance. In Equation (36), the lowest traffic intensity is multiplied by the $n - j - 2$ shortest department lengths. Then, the next two lowest traffic intensities are multiplied by the $n - j - 3$ shortest department lengths, and so on. With $\Delta_k = \frac{k(k+1)}{2}$ the k-th triangular number, we write:

$$LB_{edge}(s^j) = \sum_{k=1}^{n-j-1} \sum_{l=\Delta_{k-1}+1}^{\Delta_k} T_{<,l} \sum_{m=1}^{n-j-1-k} L_{<,m}. \tag{36}$$

In the end, the RLB for the SRFLP is given by: $\underline{v}_{rlb}(s^j) = LB_{cut}(s^j) + LB_{edge}(s^j)$.