Strengthening Relaxed Decision Diagrams for Maximum Independent Set Problem: Novel Variable Ordering and Merge Heuristics

Mohsen Nafar[1]
Bielefeld University, Germany

Michael Römer
Bielefeld University, Germany

Abstract

Finding high-quality bounds is key to devising efficient exact solution approaches for Discrete Optimization (DO) problems. To this end, Decision Diagrams (DDs) provide strong and generic bounding mechanisms. This paper focuses on so-called relaxed DDs which, by merging nodes, over-approximate the solution space of DO problems and provide dual bounds the quality of which hinges upon the ordering of the variables in the DD compilation and on the selection of the nodes to merge. Addressing the Maximum Independent Set Problem, we present a novel dynamic variable ordering strategy relying on induced subgraphs of the original graph, and a new tie-based merge heuristic. In a set of computational experiments, we show that our strategies yield much stronger bounds than the standard state-of-the-art approaches. Furthermore, implementing our heuristics in a DD-based branch-and-bound, we reduce the solution times by around 33 % on average and by more than 50 % on hard instances.

2012 ACM Subject Classification Theory of computation → Discrete optimization

Keywords and phrases Decision Diagram, Dynamic Programming, Maximum Independent Set Problem, Dual Bound

Digital Object Identifier 10.4230/LIPIcs.CP.2024.21

Funding This research was funded by the Return Programme of the Federal State of North Rhine Westphalia (NRW Rückkehrprogramm).

1 Introduction

As becomes clear from the recent survey [6], Decision Diagrams (DDs) form a versatile tool for discrete optimization (DO), as they allow a compact representation of the solution space of DO problems in the form of a layered graph and provide generic mechanisms to obtain primal and dual bounds. Specifically, given a Dynamic Programming (DP) formulation of a DO problem, one can create a so-called *exact DD* such that the set of all paths in the DD corresponds to the set of feasible solutions to the DO problem. While the size of such an exact DD grows exponentially in the number of decision variables, there are two types of approximate DDs for which the number of nodes only grows linearly as the width of each DD layer is not allowed to exceed a threshold: *Restricted DDs*, which are obtained by removing feasible nodes (which are associated with states defined by the DP formulation), provide an under-approximation of the solution space, and *relaxed DDs*, which are obtained by merging nodes associated with non-equivalent states, provide an over-approximation of the solution space. As proposed in [3], relaxed and restricted DDs can be used within an exact branch-and-bound algorithm entirely based on DDs; the authors show that this method

[1] Corresponding author

achieves an excellent performance on DO problems such as the Maximum Independent Set Problem (MISP), the Maximum Cut Problem and the 2-Satisfiability Problem. For an in-depth discussion of DD-based solution approaches illustrated using a wide variety of DO problems, we refer to the monograph [1]. For an efficient open source implementation of DD-based branch-and-bound algorithm in Rust, we refer to the solver DDO presented in [10]. An alternative to the classical DD-based branch-and-bound with an open source implementation in the Julia language is *Peel and Bound* [16, 17].

The performance of DD-based branch-and-bound algorithms is highly dependent on the quality of the bounds of the approximate DDs involved. These approximate DDs are typically compiled using the so-called top-down approach, in which the DD is constructed layer by layer. For a given DO problem and a given maximum DD width, the strength of the approximate DD bounds depends on two key heuristic decisions within the compilation process: (i) the *variable ordering*, that is, the order in which the variables are considered in the top-down compilation, and (ii) the *node selection*, that is, the selection of the nodes in a DD layer to be removed (for restricted DDs) or to be merged (for relaxed DDs) in case the maximum width of a layer is exceeded.

The first decision, that is, devising a good variable ordering, is relatively straightforward for certain problems such as the 0/1-Knapsack Problem. For other problems such as the MISP, however, finding a good variable ordering turns out to be more intricate and has been considered by various authors, see e.g. the review in [6]. In particular, it has been shown that for the MISP it is useful to determine the variable ordering *dynamically*, that is, to decide upon the next variable to consider based on information becoming available during the DD compilation, e.g. in order by minimizing the number of nodes appearing in the next layer, or by performing some lookahead steps, see e.g. [1] for a comparison of different generic strategies. The MISP was also considered in several papers proposing to use Machine Learning to support the dynamic ordering of variables: As an example, in [5] the authors use Deep Reinforcement Learning to determine the variable ordering. They show that for a given maximum width of each layer, the ML-supported approach can substantially improve the bounds compared to the standard variable ordering heuristics considered in the literature. In two follow-up works [15, 4], the authors show that despite the fact that the ML-based compilation of approximate DDs is slower than the standard approaches, this bound improvement leads to a significant overall speed-up of an exact DD-based branch-and-bound solver.

The second decision, that is, the node selection decision, was also investigated by many authors. Following the monograph [1], a generic and often highly efficient strategy is to sort the states according to some criterion (e.g. the length of the partial path ending at each node), to keep the "best" nodes until the maximum width is reached and to merge the remaining nodes with the last node in the list to form one large "tail node". In [8], the authors propose a classification-based mechanism to predict the most promising node selection heuristic for each layer of a relaxed DD. The paper [9] proposes a tie-breaking strategy to deal with the problem of identical criterion values in sorting-based approaches. Other node selection approaches do not rely on sorting nodes (and creating a single large node) but aim at grouping nodes to merge according to some similarity measure. As an example, [12] proposes to use so-called collector nodes that aim at merging states that have the same value with respect to a labeling function. A similar approach was recently used in [7] who merge nodes based on partitioning the state space for a single machine scheduling problem with release times, deadlines setup times and rejection. Another technique for top-down compilation of relaxed DDs which is based on DD reduction is proposed in [13]. In

that paper, the authors partition the nodes in a lookahead layer, which is then used to reduce the target layer, i.e. the layer whose width exceeds the given maximum width. Finally, the paper [14] proposes using clustering approaches to group the nodes according to the state attributes and to form a single merged node for each group.

Contribution. Dealing with relaxed DDs for the MISP, this paper proposes both a new dynamic variable ordering strategy and a new heuristic to select which nodes to merge. The variable ordering strategy exploits the problem structure of the MISP by relying on graph-theoretical properties that can be inferred from the states of the partially compiled DDs. The new merge heuristic aims at reducing the approximation error in sorting-based merge strategies resulting from merging the whole tail into a single large node by introducing an additional merged node from nodes around the maximum width border having the same value of the sorting criterion. In a set of computational experiments with randomly generated graph instances with 100 vertices and different densities, we observe that each of the proposed approaches independently provides significantly stronger DD relaxations for the MISP than DDs compiled with standard approaches. When combined, i.e. using our proposed variable ordering and merge heuristic at the same time, the bounds become much stronger. Furthermore, implementing these heuristics in a DD-based branch-and-bound algorithm, the solution time reduces by 33% on average compared to a branch-and-bound using standard variable ordering and merge strategies. The solution time reduction grows with the hardness of the instances; for the hardest instances, the solution time reduction amounts to more than 50%.

2 Exact and Approximate Decision Diagrams

A decision diagram $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is a layered directed acyclic graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. The paths in $\mathcal{D}$ represent solutions to a discrete optimization problem $\mathscr{P}$ with a maximization objective function f and an n-dimensional vector of decision variables $x_1, \cdots, x_n \in \{0, 1\}$. $\mathcal{N}$ is partitioned into $n+1$ layers $\mathcal{N}_1, \ldots, \mathcal{N}_{n+1}$, where $\mathcal{N}_1 = \{\mathbf{r}\}$ and $\mathcal{N}_{n+1} = \{\mathbf{t}\}$ for a *root* $\mathbf{r}$ and a *terminal* $\mathbf{t}$. Each arc $a = (u, u')$ connects nodes of two consecutive layers $\ell(u), \ell(u') = \ell(u) + 1$ and is associated with a decision $d(a)$ representing the assignment $x_{\ell(u)} = d(a)$. This means that a path $p = (a_1, \ldots, a_n)$ starting from $\mathbf{r}$ and ending at $\mathbf{t}$ represents the solution $x(p) = (d(a_1), \ldots, d(a_n))$. We denote the set of all $\mathbf{r}$-$\mathbf{t}$ paths with $\mathcal{P}$, and we refer to the solutions to $\mathscr{P}$ represented by $\mathcal{P}$ with $\mathrm{Sol}(\mathcal{D})$. Moreover, each arc a has length $w(a)$ and $\sum_{i=1}^{n} w(a_i)$ provides the length $w(p)$ of path p. We refer to $\mathcal{D}$ as exact if $\mathrm{Sol}(\mathcal{D}) = \mathrm{Sol}(\mathscr{P})$ if for each path $p \in \mathcal{P}$ we have $w(p) = f(x(p))$; then a longest path in $\mathcal{D}$ forms an optimal solution to $\mathscr{P}$.

To deal with the exponential growth of the DD size, DD-based solution approaches such as DD-based branch-and-bound [2] rely on so-called approximate DDs that can be used to obtain upper or lower bounds for the solutions of $\mathscr{P}$. There are two types of approximate DDs: in a *restricted* DD $\mathcal{D}$, one aims at considering only promising nodes and arcs, meaning that $\mathrm{Sol}(\mathcal{D}) \subseteq \mathrm{Sol}(\mathscr{P})$, and thus, the longest path in a restricted DD provides a lower bound to $\mathscr{P}$. The second type of approximate DD, which is the one we focus on in this paper, is the *relaxed* DD providing an upper bound: in a relaxed DD, we have $\mathrm{Sol}(\mathcal{D}) \supseteq \mathrm{Sol}(\mathscr{P})$, that is, the set of paths may contain paths associated with infeasible solutions to $\mathscr{P}$. Regarding the objective function value, every path in a relaxed DD needs to satisfy $w(p) \geq f(x(p))$. In both restricted and relaxed DDs, a common approach to control the size of the DD is to impose a maximum width W for each layer in the DD which is enforced by removing nodes (in a restricted DD) or merging nodes (in a relaxed DD).

A common approach to compile an exact DD is to provide a Dynamic Programming (DP) formulation of $\mathscr{P}$ and to compile the DD in a top-down fashion. To do so, every node u is associated with a state S_u and every arc a is associated with a state transition induced by the decision $d(a)$ associated with a. S_u is an element of the state space $\mathcal{S}$; the state $S_{\mathbf{r}}$ associated with the $\mathbf{r}$ is the so-called *initial state*. The state S_v of the target node v of the arc depends on the state S_u of the arc's source node as well as on d and is computed by the state-transition function $f(S_u, d)$. The objective function contribution of a decision are computed by a reward function $g(S_u, d)$. Finally, the set of out-arcs of a node u is determined by the set of feasible decisions $X(S_u)$ given state S_u. The top-down compilation then proceeds layer-by-layer until reaching layer $\mathcal{N}_n$; all arcs emanating from that layer point to the terminal node $\mathbf{t}$. In a DD compiled in the sketched top-down fashion, any pair of nodes in a layer has different states, that is, partial paths ending in the same state point to the same node.

In case of approximate DDs, after having created all nodes of a given layer, one reduces its size to W by removing or merging nodes. Nodes are merged by redirecting the incoming arcs of the nodes to be merged to a single merged node. In order to ensure that no feasible completions of any of the merged nodes is lost, one requires a problem-specific merge operator $\oplus$ for the states associated with the two nodes, see [11] for a discussion of the conditions a valid merge operator needs to satisfy.

Algorithm 1 displays the pseudocode for a top-down compilation procedure for DD construction. The procedure takes a DP formulation DP (comprising the definition of the state space $\mathcal{S}$ including the initial state $S_{\mathbf{r}}$, the functions X, f and g), a DD $\mathcal{D}$ containing only the root node and the maximum width W. Calling the algorithm with an unlimited width W will yield an exact DD and depending on the operation performed in line 11, it will result in a restricted or relaxed DD. In addition to these, the algorithm requires node selection (for restricting and/or relaxing the layers) and variable ordering heuristics (an order for considering the decision variables for layer by layer construction, since every layer corresponds to one decision variable). In order to allow for a dynamic variable ordering, Algorithm 1 introduces the set *unfixed* of variables that have not been considered so far in the compilation as well as the the generic procedure *NextVariable* which chooses the next variable according to a given variable ordering strategy. Note that in case of a static variable ordering strategy, *NextVariable* simply returns the next variable according to a pre-specified order.

Algorithm 1 Top-Down DD Compilation.

```
1:  CompileTopDown (DP, 𝒟, W)
2:  unfixed = set of all decision variables
3:  for k = 1 to n do
4:     x_k = NextVariable (𝒩_k, unfixed)
5:     unfixed = unfixed/{x_k}
6:     for all u ∈ 𝒩_k do
7:        for all d ∈ X(S_u) do
8:           v = GetOrAddNode (𝒩_k, f(S_u, d))
9:           AddArc (u,v,d)
10:    if |𝒩_{k+1}| > W then
11:       RelaxLayer/RestrictLayer (𝒩_{k+1})
12: return 𝒟
```

As mentioned in the introduction, the strength of the bounds obtained with approximate DDs compiled using a top-down approach crucially depends on the strategies used to determine the variable ordering (determined in line 4 of Algorithm 1), and on the strategies

for selecting the nodes to remove or merge (line 11 of Algorithm 1). In this paper, we devise new heuristics for both of these decisions for the compilation of relaxed DDs for the Maximum Independent Set Problem which will be discussed next.

3 Decision Diagrams for the Maximum Independent Set Problem

In this section, we briefly introduce the Maximum Independent Set Problem (MISP) and its DP formulation. We then illustrate how this DP formulation can be used to construct exact and relaxed DDs.

3.1 The Maximum Independent Set Problem

Given a graph $G = (V, E)$ with n vertices, where $V = \{v_1, v_2, \cdots, v_n\}$ is the set vertices and E is the set of edges, the Maximum Independent Set Problem (MISP) asks for the largest subset $I \subseteq V$ such that no two vertices in I are connected via an edge, i.e. $I = \{v \in V | (u, v) \notin E, \forall u \in I\}$.

Example. Fig. 1 shows an example that will serve for illustration purposes in the remainder of this paper. It shows a graph G with five vertices. As can be easily verified, there are multiple optimal solutions, each of which contains two vertices, e.g. $I = \{v_1, v_4\}$ or $I = \{v_3, v_5\}$ or $I = \{v_2, v_3\}$.

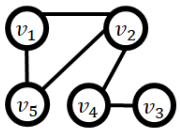

Figure 1 Example Graph G for a Maximum Independent Set Problem.

In order to be able to compile a decision diagram for MISP we need to formulate the MISP in terms of a DP. To begin with, a state S_u associated with a node u in the DD corresponds to a subset of the vertices V of the original graph G, namely with the vertices that are still available to be part of an independent set. The initial state S_r associated with the root node r thus corresponds to V, the terminal state corresponds to the empty set. Each layer j in the DD is associated with the decision variable x_j which consists in adding the j-th vertex $v(j)$ (according to the chosen variable order) in the original graph G to the independent set or not. Given a state S_u and a decision $d(a)$ ($d = 1$ means adding the vertex to the solution, $d = 0$ not adding it) associated with arc $a = (u, u')$ emanating from node u, the state transition function $f_j(S_u, d(a))$ determines the state of node u' in the next layer $j + 1$ of the DD. Specifically, $f_j(S_u, 0) = S_u / \{v(j)\}$, and $f_j(S_u, 1) = S_u / \{\Gamma(v(j))\}$ where $\Gamma(v(j))$ is the set of vertices adjacent to $v(j)$ in V. Note that if $v(j) \notin S_u$, the decision $d = 1$ is not feasible, and thus the DD will not contain an arc a emanating from u with $d(a) = 1$. The reward function $g_j(S_u, d)$ is $g_j(S_u, 0) = 0$ and $g_j(S_u, 1) = 1$.

Example (continued). Fig. 2 shows an exact DD for the MISP on graph G from Fig. 1 with the variable order according to the indexes of the vertices in V, that is, $v(j) = v_j \forall v \in V$. Every node of the DD is framed using blue color, where its corresponding state (i.e. a subset of the vertices in V) is placed inside its frame. Small orange labels next to each

node are their partial objective value. Each dashed red and solid green arc show the assignment of value 0 and 1 to the corresponding decision variable, respectively. The exact DD in this example has a width of 4, and one of the longest r-t-paths with length 2 is $[x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 1]$, giving us an optimal solution, namely $\{v_4, v_5\}$ with the value 2.

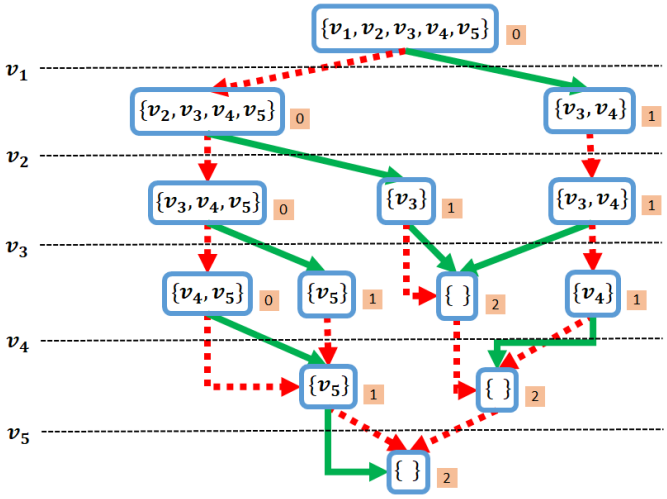

Figure 2 Exact DD for the MISP example graph G.

3.2 Relaxed Decision Diagrams for the MISP

As previously explained, a relaxed DD over-approximates the solution space of the problem under consideration, resulting in a dual bound. This relaxation is obtained by merging nodes with non-equivalent states. In order to obtain a valid relaxation, this merge process must ensure that no feasible solution is lost, that is, that the whole feasible solution space of the problem is included in the space represented by the relaxed DD. Merging two nodes u and u' into a node M involves two steps: First, all incoming arcs to nodes u and u' must be redirected to merged node M. Second, the state S_M of the new node must be determined in a way that the solution space of the tail problem starting from M contains the tail problem solutions of both nodes u and u'. As explained above, the state $S_M = S_u \oplus S_{u'}$, where $\oplus$ is a so-called merge operator. As discussed e.g. in [1], a valid merge operator for MISP is $\cup$, that is, S_M is given by the union of the vertex sets (subsets of V in the problem graph G) forming the states S_u and $S_{u'}$.

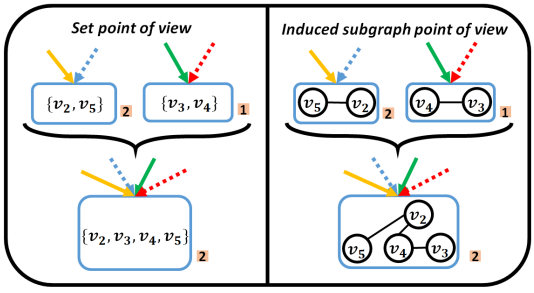

Figure 3 Merging two nodes in a relaxed DD for the MISP.

Fig. 3 illustrates the merge of two nodes into a single node. Observe that the state of the resulting merged node is the union of states of the two nodes on top. Moreover, recall that while a state in MISP is a subset of the vertices, it can be interpreted to represent an induced subgraph of the original graph on those vertices. Therefore, the edge between v_2 and v_4 which did not exist in any of the two induced subgraphs of the nodes to be merged now will be included in the induced subgraph of the state of the merged node. Actually, this augmented interpretation of the DD states for the MISP carries more information about the DD nodes – this fact will later be used to develop the new variable ordering heuristic proposed in this paper.

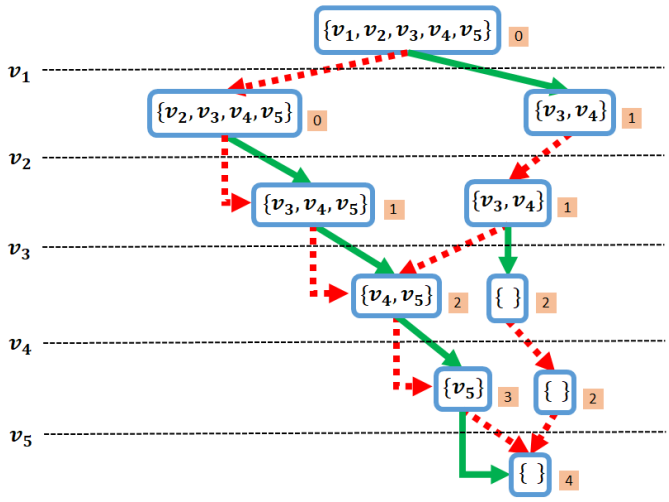

Figure 4 Relaxed DD for graph G, where variables are ordered according to their indices in the original graph and $W = 2$. Its obtained gap is 100%.

Example (continued). Fig. 4 represents the relaxed DD for graph G where $W = 2$, using the variable ordering given by indexes of the vertices, i.e. $v(1) = v_1, v(2) = v_2, v(3) = v_3, v(4) = v_4, v(5) = v_5$, in Fig. 1 that was also used for the exact DD. It turns out that the resulting dual bound has the value of 4 which is two times the optimum value.

4 A New Dynamic Variable Ordering for the MISP

The order of the decision variables according to which the DD is being compiled heavily affects both the size of an exact DD, and quality of the bounds from approximate DDs with a fixed maximum width. In the examples above, we used a static variable ordering, that is, an ordering that is specified before the compilation of the DD and that is independent of the configuration of the layers during the compilation process. However, it turns out that the best variable ordering strategies for the MISP are dynamic, that is, they use information of the partially compiled DD to choose the next decision variable (in case of MISP, the next vertex).

In the following, we will first consider the dynamic variable ordering strategy most commonly used for the MISP in the literature. Then, we propose a novel dynamic variable ordering strategy that exploits the information gained by interpreting the DD states in terms of induced subgraphs.

4.1 Standard Strategy: Minimum Number of States (MIN)

In this ordering, vertices are assigned a value that corresponds to the number of times they appear in the states of the nodes in the current layer $\mathcal{N}_k$. Then, the vertex exhibiting the minimum number appearances is selected as the next vertex (variable) to be considered in the top-down compilation of the DD. Algorithm 2 shows the corresponding heuristic which is called *Minimum Number of States (MIN)*. The worst-case time complexity to perform this selection is $O(W \cdot |V|)$ per layer. In this algorithm, $\mathcal{N}_k$ is the current layer and *unfixed* is the set of the vertices to which no decision has been assigned yet.

Algorithm 2 NextVariableMIN ($\mathcal{N}_k$, unfixed).

```
NS := a dictionary where keys are unfixed variables (associated with vertices of G) and values 0
for x ∈ unfixed do
    for all S ∈ N_k do
        if v(x) ∈ S then
            NS[x]+=1
return the unfixed variable with minimum NS
```

Example (continued). Using the MIN variable ordering in the top-down compilation of a relaxed DD with $W = 2$ for the example graph G from Fig. 1 results in the following variable order: $[v(1) = v_1,\ v(2) = v_2,\ v(3) = v_5,\ v(4) = v_3,\ v(5) = v_4]$. Following this order of variables (vertices), the corresponding relaxed decision diagram which provides a dual bound with value 3 is illustrated in Figure 5.

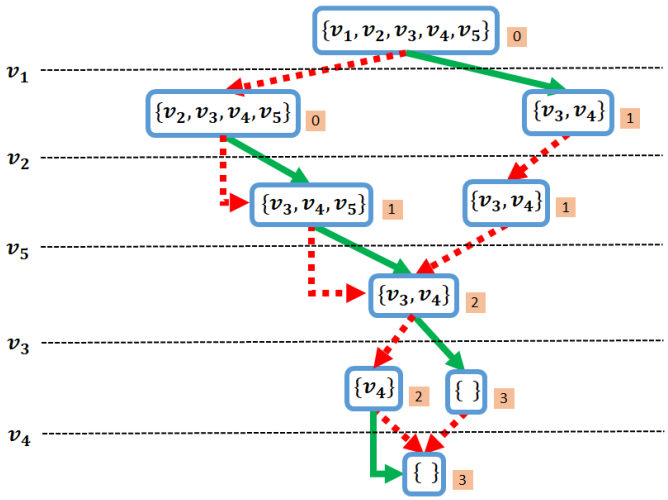

Figure 5 Relaxed DD for graph G, compiled via MIN and $W = 2$, yielding a gap of 50%.

The intuition behind the effectiveness of the MIN variable ordering strategy is that a node u only has an outgoing 1-arc for the variable associated with a vertex v if $v \in S_u$. As a result, a vertex appearing only a few times in the states of the layer under consideration will result in a small number of outgoing arcs of the layer and thus in a small number of nodes in the next layer.

4.2 A New Strategy: Current Degree Sum (CDS)

Now, we present a new dynamic variable ordering heuristic which is based on the interpretation of node states as induced subgraphs of the original graph introduced in Section 3. Previously, we mentioned that the intuition behind the MIN strategy is to improve the quality of the dual bounds in a relaxed DD by controlling the growth of the layers by choosing a variable that will result in less feasible decisions to be taken. Adding to this, the idea behind our new strategy is to reduce the "destructive" effect of the subsequent merge operations: We aim at choosing the variable order in a way that the difference between the states to be merged in a layer is somewhat small such that the resulting merged state is not too different from the states of the nodes to be merged.

Intuitively, a vertex with a lower degree in a given graph is likely to belong to a higher number of independent sets in that graph than a vertex with a larger degree. Therefore, it can be beneficial to decide about such vertices (i.e. vertices with smaller degree) sooner than later, because it can result in exploration of the search space that is closer to the optimum solution. Since the graph-theoretical information of vertices in a MISP evolves a lot during the compilation (every subproblem corresponding to a node/state is associated with an induced subgraph of the original graph on the members of that state), it is crucial to recompute them for each state and then take the best decision.

Current Degree Sum (CDS) is a novel variable ordering that can account for all of these intuitions, i.e. resulting in a lower number of feasible decisions, reducing the destructive effect of subsequent merge operations, and providing a better chance of resulting in partial solutions (tail solutions) closer to optimum. Recall that every state in a DP formulation of MISP is represented via a subset of the vertices of the original graph. Considering the induced subgraph on members of these states, every member (i.e. a vertex) has a degree which might be different from its degree in another state (induced subgraph), we call it the *current degree* of a vertex, i.e. dg_v^S reads degree of vertex v in state S. The strategy is to sum up the current degrees of each vertex, i.e. $\sum_{S \in \mathcal{N}_k} dg_v^S$ in the layer under consideration, and to choose the vertex with minimum sum.

Algorithm 3 shows the process for CDS variable ordering, which has a worst-case time complexity of $O(W \cdot |V|^2)$ per layer.

Algorithm 3 NextVariableCDS procedure ($\mathcal{N}_k$, unfixed).

```
CDS = a dictionary where keys are unfixed variables and values 0
for all S ∈ N_k do
    for x ∈ S do
        for y ∈ S do
            if x is adjacent to y then
                CDS[x] += 1
return the unfixed variable with minimum CDS
```

Example (continued). Using CDS variable ordering in the top-down compilation of a relaxed DD with $W = 2$ for the example graph G from Fig. 1 results in the following order of variables / vertices: $[v(1) = v_3,\ v(2) = v_4,\ v(3) = v_2,\ v(4) = v_1,\ v(5) = v_5]$. The corresponding relaxed decision diagram provides a dual bound with value 2 which is exactly the optimum value of the original problem (see Fig. 6).

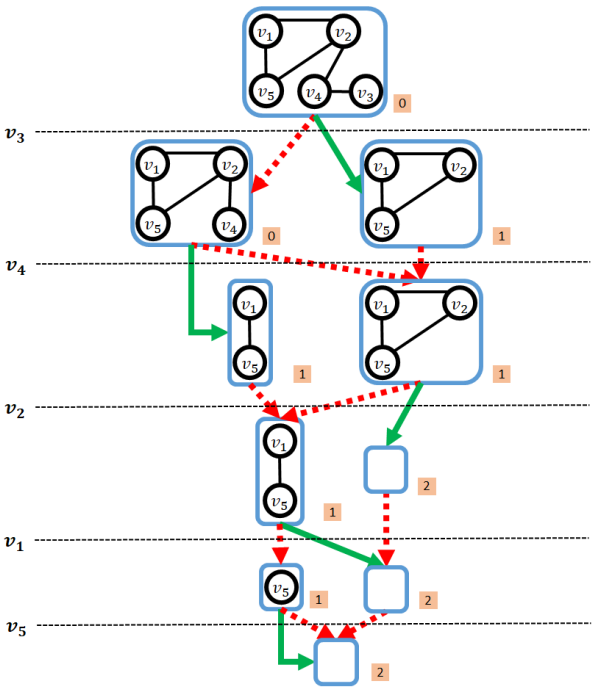

Figure 6 Relaxed DD for graph G compiled using **CDS** and $W = 2$, yielding a gap of 0%.

5 A New Merge Heuristic

The second important decision during the top-down compilation of relaxed DDs that has a huge impact on the quality of the achievable dual bounds is to select the nodes to merge in case the maximum width is exceeded. A (heuristic) strategy for taking this decision is also called a merge heuristic. In this section, we will first describe a problem-agnostic merge heuristic that is commonly used in the literature and then propose a new merge heuristic.

5.1 Standard Strategy: SortObj Merging (SO)

A highly generic merge heuristic for the top-down compilation of relaxed DDs that we will subsequently refer to as *SortObj (SO)* works as follows: First, all nodes in the layer the size of which exceeds the given maximum width W are sorted according to their objective function values. Then, the first $W - 1$ nodes, i.e. the nodes having the highest objective values, will remain in the layer, and the rest of the nodes, which we refer to the tail (of the sorted list) will be merged into a single node called M_{tail}.

Fig. 7 shows an example of SortObj being applied in a layer with 12 nodes, the maximum width is $W = 8$. In the figure, the nodes are sorted according to their objective values (written inside the orange boxes under the nodes). When applying SortObj, the first 7 nodes, i.e. $\{A, B, C, D, E, F, G\}$, will be retained in the layer and the rest of the nodes, i.e. $\{H, I, J, K, L\}$, will be merged and form the merged node M_{tail} that will replace all the nodes in the tail. The vertical dashed line in the figure marks the *border* between the nodes to be retained and those that are merged into M_{tail}.

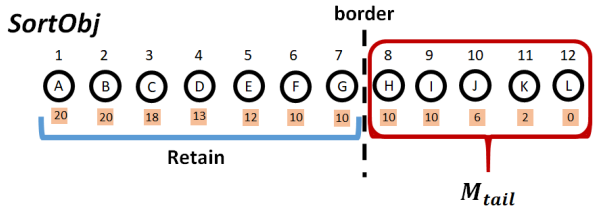

Figure 7 SortObj merging heuristic where $W = 8$.

5.2 A new Strategy: Border Tie Merging (BT)

Here, we present a new merge heuristic which is based on identifying certain ties that may arise in the layers of relaxed DDs. This merging heuristic that we call *Border Tie* merging and denote by **BT** is described after the following definition:

▶ **Definition 1** (Border Tie). *Let the nodes in a layer of a DD be sorted according to some criterion C and SortC be its corresponding sorted list, and let W be the given maximum width. A subsequence of SortC in which all nodes have the same criterion value and which includes SortC $[W-1]$ and SortC $[W]$ is called a Border Tie.*

Note that the smallest border tie includes at least 2 nodes, i.e. SortC $[W-1]$ and SortC $[W]$. Furthermore, there can exist at most one border tie in a layer. The criterion we use in border tie merging for the MISP is the same as in SortObj, that is, objective function value associated with each node.

A merged node in MISP is a "two-sided" over-approximation of all the nodes in the merge: From one side it over-approximates the states of the merged nodes by forming a super-set of their states. From the other side, redirecting the in-arcs causes an over-approximation of the longest path from root to the merged node. The intuition behind this new merge heuristic is to control the approximation error caused by the merge operation from one side, i.e. the over-approximation of the length of the partial solution caused by redirecting the in-arcs. Therefore, if we merge the nodes that have the same objective value (the lengths of their partial solutions are the same), then we can control the over-approximation error from one side. Moreover, we reckon that this merging heuristic will be a perfect fit for **CDS**, since one of the intuitions behind the design of **CDS** was to keep the diversity of the states in layers in a small range so as to control the destructive effect of the merge, that is the other side of the over approximation in MISP. Therefore, it is expected that coupling these two heuristics, i.e. **BT** and **CDS**, will strengthen the dual bounds obtained via relaxed DDs for MISP as they decrease the over-approximation error from both sides.

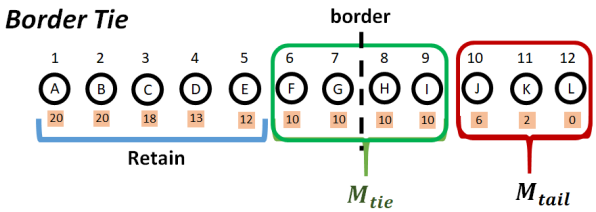

Figure 8 Border Tie merging heuristic where $W = 8$.

Fig. 8 shows the border tie merge heuristic applied to a layer with 12 nodes where $W = 8$. Note that the first two nodes in the list have the same objective value and therefore form a tie; however, their tie does not play a role in border tie merging heuristic. The **border tie** heuristic first identifies the nodes in the border tie (i.e., $\{F, G, H, I\}$) and merges them into a single merged node called M_{tie}. The nodes on the left side of the border tie are unchanged and remain in the layer. Finally, the nodes in the tail of the list, if there exist any, will be merged into M_{tail}. Note the following special cases regarding the tail of the sorted list may arise:

1. the tail is empty: in this case, M_{tie} is the only one merged node in the relaxed layer,
2. the tail comprises a single node: M_{tie} is the only one merged node in the relaxed layer, and the single tail node is unchanged and remains in the relaxed layer.

6 Computational Results

In this section, we present the results of computational experiments with two different dynamic variable ordering approaches (i.e. MIN, CDS) and two merge heuristics (i.e. SO, BT) and their combinations for the MISP. First, we assess the performance of the different strategy combinations with respect to the time-bound trade-off when compiling relaxed DDs. Second, we investigate the effects of these strategy choices for compiling relaxed DDs on the performance of an otherwise standard DD-based branch-and-bound algorithm [3]. For performing the experiments, we created nine instance sets, each of which contains 20 randomly generated graphs with 100 vertices. The instances set differ with respect to the graph density which ranges from 0.9 to 0.1. We implemented all approaches in the Julia programming language and ran the experiments on a Windows machine with 16GB RAM and an 11th Gen Intel(R) Core(TM) i7-11800H processor with 2.30 GHz.

6.1 Results on Dual Bounds

Fig. 9 shows the average dual gaps provided by using the strategy combinations (MIN,SO), (**CDS**, SO), (MIN,**BT**), and (**CDS**, **BT**) in relaxed DDs for the MISP on graph instances with different densities. The gaps reported in this figure are computed as $\frac{\text{dual bound}-\text{optimum}}{\text{optimum}} \times 100$ and then averaged over the 20 instances. Moreover, all relaxed DDs are compiled using given maximum widths $W \in \{50, 100, 200, 500, 1000\}$; the X-axis displays the time needed for compiling the relaxed DDs in ms. Every sub-plot in this figure corresponds to one density. In the sub-plots red-dashed, blue-solid, green-solid, and black-solid curves show (MIN,SO), (**CDS**, SO), (MIN,**BT**), and (**CDS**, **BT**) performances, respectively.

From the figure it is clear that all combinations involving our new heuristics, i.e. (**CDS**, SO), (MIN,**BT**), and (**CDS**, **BT**), provide stronger dual bounds than the standard strategy, i.e. (MIN,SO). Looking into the plots reveals that as the density decreases, the instances become harder and the performance difference between standard method and our proposed methods increases, meaning that for harder instances our strategies provide significantly stronger bounds. Moreover, (**CDS**, **BT**) which is the combination of our proposed dynamic variable ordering with our proposed merge heuristic clearly outperforms the other approaches.

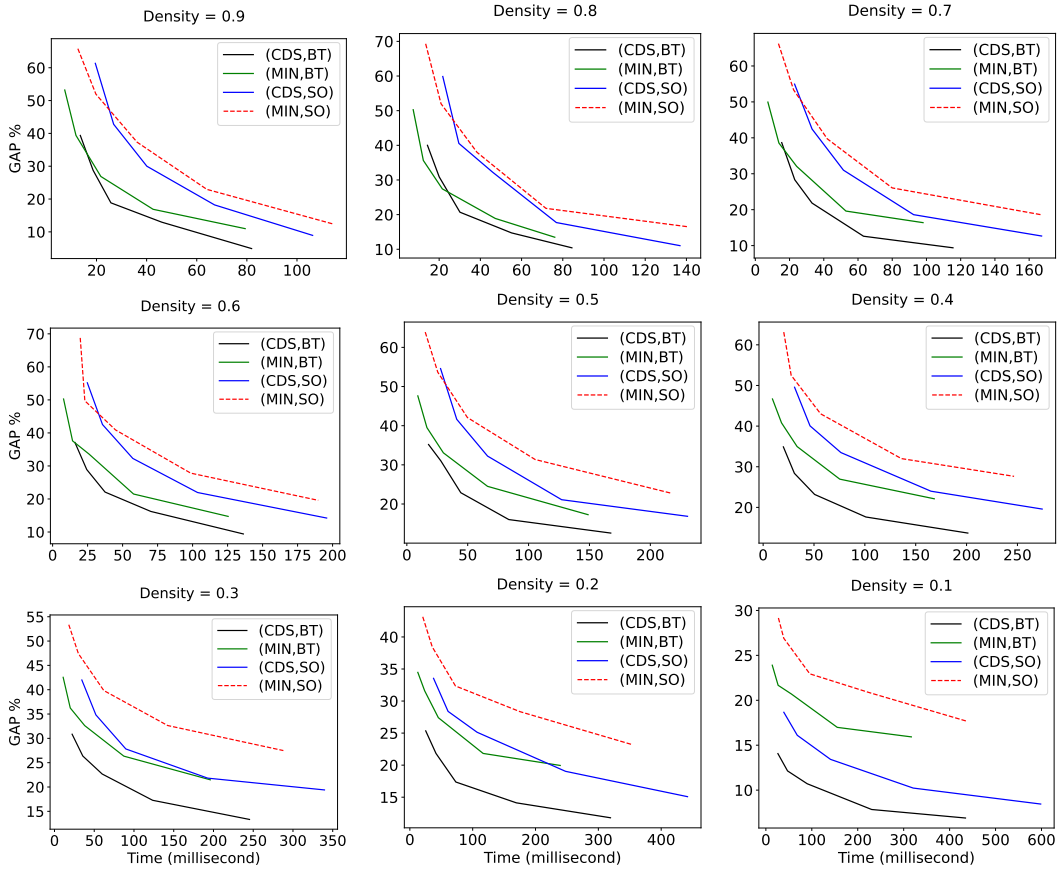

Figure 9 Gap-time performance of different strategies when being used in compiling relaxed DDs.

Table 1 shows the average size of the relaxed DDs built using different strategy combinations for a given maximum width $W = 1000$. A comparison between (MIN,SO) and (**CDS**,SO) supports one of the intuitions behind the design of CDS, i.e. reducing the destructive effect of merge operation by controlling the diversity of the states (subproblems / induced subgraphs). While MIN, which was designed to decrease the width of the layers, provides smaller relaxed DDs than those compiled using CDS, the relaxed DDs compiled via CDS are stronger, hinting at the successful control of the diversity of the states when using CDS. Another interesting observation is that when comparing the combinations that include **BT** to those that do not is the significant drop of the sizes of the DDs compiled having **BT** as their merge heuristic. It shows that although the DDs compiled using this heuristic contain more merged nodes (in some layers they can contain up to two merged nodes whereas in SortObj every layer has at most 1 merged node) and one might expect worse bound quality, it actually results in stronger dual bounds (see Fig. 9).

6.2 Performance within a DD-based Branch-and-Bound

We now present the results of implementing the proposed variable ordering and merge heuristics and their combinations in a DD-based branch-and-bound algorithm proposed in [3]. In the implemented branch-and-bound algorithm, the primal bounds are obtained using restricted DDs. For the compilation of the restricted DDs, no advanced approaches are used: The variables are ordered only once according to their degree in the original graph

Table 1 Average size of relaxed DDs compiled via different methods with $W = 1000$.

Density	Average DD Size (nodes)			
	(MIN,SO)	(**CDS**,SO)	(MIN,**BT**)	(**CDS**,**BT**)
0.1	73513	76190	**58059**	59069
0.2	71324	73190	**56393**	57173
0.3	68484	69470	**54020**	55253
0.4	65088	66580	**51870**	53049
0.5	62053	63355	**50456**	51356
0.6	58946	59661	**47715**	47505
0.7	55058	56040	**43606**	45629
0.8	50808	51466	**35334**	36025
0.9	45770	47272	**37578**	41607

in increasing order; the node selection follows the SortObj heuristic. We are dealing with a relatively pure implementation of a DD-based branch-and-bound that does not make use of advanced techniques but basically follows the description in the monograph [1]. All relaxed and restricted DDs in the branch-and-bound are compiled with a maximum width $W = 100$.

Table 2 Average solution time of DD-based branch-and-bound for strategy combinations.

Density	Time (Seconds)			
	(MIN,SO)	(**CDS**,SO)	(MIN,**BT**)	(**CDS**,**BT**)
0.1	872.3	821.9	445.8	**320.9**
0.2	126.4	94.0	85.5	**56.4**
0.3	31.0	26.8	24.3	**17.0**
0.4	10.1	9.3	8.6	**5.9**
0.5	4.3	3.5	4.0	**2.5**
0.6	1.92	1.70	1.57	**1.4**
0.7	1.11	1.01	0.93	**0.89**
0.8	0.73	0.70	0.81	**0.66**
0.9	0.60	**0.56**	0.65	0.61

Table 2 shows the average solution time of DD-based branch-and-bound using relaxed DDs constructed via different strategies (all instances are solved to optimality). The results reveal that all combinations involving our new strategies are able to reduce the solution time considerably in comparison to the baseline combination (MIN,SO).

Table 3 Solution time reduction in percent of different strategy combination in comparison to the baseline, i.e. (MIN,SO).

Density	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	Average
(**CDS**,SO)	5.7	25.61	13.42	8.4	18.3	11.4	9.0	4.1	**6.6**	11.43
(MIN,**BT**)	48.8	32.3	21.4	15.6	6.2	18.2	16.2	-10.9	-8.3	15.52
(**CDS**,**BT**)	**63.2**	**55.3**	**45.2**	**42.0**	**40.6**	**25.52**	**19.8**	**9.5**	-1.6	**33.30**

Table 3 summarizes the solution time reductions of each combination per density compared to the baseline. As becomes clear from the table, using our proposed heuristics individually, i.e. (**CDS**,SO) and (MIN,**BT**), reduces the solution time by 11% and 15% on

average. However, if we use both heuristics at the same time, i.e. (**CDS**,**BT**), the solution reduction increases to 33% on average. In all cases, as the hardness of the instances increases (that is, the graph density decreases), the superiority of the proposed methods becomes more significant, such that the best combination, i.e. (**CDS**,**BT**), has a solution time reduction of more than 50% compared to the baseline for instances with density 0.2 and 0.1.

Table 4 Average number of sub-problems solved in the DD-based branch-and-bound for different strategy combinations.

Density	**Node Size (Sub-problem Solved in B&B)**			
	(MIN,SO)	(**CDS**,SO)	(MIN,**BT**)	(**CDS**,**BT**)
0.1	63880	39051	34183	**17150**
0.2	15454	6623	9088	**3816**
0.3	5906	3053	3830	**1700**
0.4	2609	1581	1814	**802**
0.5	1452	796	947	**450**
0.6	702	397	440	**266**
0.7	399	256	283	**195**
0.8	291	210	245	**173**
0.9	247	**174**	224	178

Another interesting aspect of a branch-and-bound algorithm is the number of subproblems that are solved until an optimal solution is reached. We report the average number of the subproblems solved in the DD-based branch-and-bound using different methods in Table 4.

Table 5 Reduction of the number of the solved subproblems in percent in DD-based branch-and-bound using different strategy combinations in comparison to the base line, i.e. (MIN,SO).

Density	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	Average
(**CDS**,SO)	38.8	57.1	48.3	39.4	45.2	43.3	35.7	27.7	29.6	40.6
(MIN,**BT**)	46.4	41.1	35.1	30.4	34.7	37.3	29.0	15.8	9.3	31.0
(**CDS**,**BT**)	**73.1**	**75.30**	**71.2**	**69.2**	**68.9**	**62.0**	**51.0**	**40.3**	**27.6**	**59.8**

Table 5 summarizes the reduction of the number of subproblems solved in DD-based branch and bound using different combinations of strategies. A comparison between the methods shows that although (MIN,**BT**) have better dual gaps and better solution times than (**CDS**,SO), it requires more subproblems to be solved. This can be a sign that in a DD-based branch-and-bound, having good quality bounds is not the only factor for having a good solution time: If it was the only factor, we should have seen a smaller number of solved subproblems for (MIN,**BT**) as it gives better bounds than (**CDS**,SO). This suggests that perhaps the reason for this can be one of the intuitions behind the design of CDS, which was to move in the direction of having solutions that potentially have more intersections with optimal solutions. However, when combining CDS and BT, the algorithm has the benefits of the both, i.e. good quality bounds and a reduction of the number of solved subproblems by 50% on average.

To put our results into perspective, let us briefly mention the results reported in [4], where the authors compare the impact of their RL-based dynamic variable ordering strategy to the MIN strategy within a standared DD-based branch-and-bound algorithm for solving randomly generated MISP instances with a density of 0.3 and between 200 and 300 vertices. It turns out that for instances that could be solved to optimality, the reduction in solution

time of their best approach compared to MIN is around 10%, whereas our reductions amount to 13.42%, 21.4%, and 45.2% for (**CDS**,SO), (MIN,**BT**), and (**CDS**,**BT**) combinations, respectively, for instances with density 0.3.

7 Conclusion

In this paper, we propose a novel dynamic variable ordering and a new merge heuristic for the top-down compilation of relaxed DDs for the MISP. The dynamic variable ordering strategy relies on the information obtainable from induced subgraphs of the original graph and the merge heuristic merges the nodes among which there is a tie regarding their partial objective value. Our computational experiments from applying the new strategies to a set of randomly generated graph instances containing 100 vertices with densities ranging from 0.9 to 0.1 (20 instances per density) show that our proposed strategies, individually, are capable of significantly strengthening the dual bounds compared to the standard strategy from the literature where this strengthening becomes more significant when our methods are combined. For the harder instances, i.e. lower densities, the performance gap is higher in our favor. In the end, the implementation of the resulting relaxed DDs into a DD-based branch-and-bound reduces the solution time by 33% on average and more than 50% on harder instances.

References

1 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John Hooker. *Decision Diagrams for Optimization*. Springer Publishing Company, Incorporated, 1st edition, 2016.

2 David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John Hooker. Branch-and-bound based on decision diagrams. In *Decision Diagrams for Optimization*, pages 95–122. Springer, 2016.

3 David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

4 Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5):2552–2570, 2022.

5 Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1443–1451, 2019.

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

7 Mathijs de Weerdt, Robert Baart, and Lei He. Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal of Operational Research*, 291(2):629–639, 2021.

8 Nikolaus Frohner and Günther R Raidl. Merging quality estimation for binary decision diagrams with binary classifiers. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 445–457. Springer, 2019.

9 Nikolaus Frohner and Günther R Raidl. Towards improving merging heuristics for binary decision diagrams. In *Learning and Intelligent Optimization: 13th International Conference, LION 13, Chania, Crete, Greece, May 27–31, 2019, Revised Selected Papers 13*, pages 30–45. Springer, 2020.

10 Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 5243–5245, 2021.

11 John N Hooker. Job sequencing bounds from decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 565–578. Springer, 2017.

12 Matthias Horn, Johannes Maschler, Günther R Raidl, and Elina Rönnberg. A*-based construction of decision diagrams for a prize-collecting scheduling problem. *Computers & Operations Research*, 126:105125, 2021.

13 Mohsen Nafar and Michael Römer. Lookahead, merge and reduce for compiling relaxed decision diagrams for optimization. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 74–82. Springer, 2024.

14 Mohsen Nafar and Michael Römer. Using clustering to strengthen decision diagram bounds for discrete optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 8082–8089, 2024.

15 Augustin Parjadis, Quentin Cappart, Louis-Martin Rousseau, and David Bergman. Improving branch-and-bound using decision diagrams and reinforcement learning. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*, pages 446–455. Springer, 2021.

16 Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Peel-and-bound: Generating stronger relaxed bounds with multivalued decision diagrams. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CP.2022.35.

17 Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Improved peel-and-bound: Methods for generating dual bounds with multivalued decision diagrams. *Journal of Artificial Intelligence Research*, 77:1489–1538, 2023.