Clustering for Relaxed and Restricted Decision Diagram Bounds: When It Works and Why

Alice Burlats, Roger Kameugne, Cristel Pelsser, and Pierre Schaus

UCLouvain/ICTEAM, Louvain-la-Neuve Belgium

Abstract. Decision-Diagram-based Branch-and-Bound solves discrete optimization problems by exploiting bounds provided by two types of bounded-width decision diagram. The first is the *restricted decision diagram* obtained by discarding less promising states that provide primal bounds. The second is the *relaxed decision diagram* obtained by state merging that yields a dual bound. Their performance depends heavily on the heuristic used to discard or merge nodes. While traditional methods discard or merge nodes based on the cost, recent research suggests that clustering nodes based on state similarity (e.g., via k-means) can help produce tighter bounds. However, current clustering methods are difficult to apply to complex, non-vector states. We propose to use a more general clustering framework that accepts user-defined distance metrics, allowing it to be applied to any state definition and scales to very large state spaces. We test this approach against standard cost-based strategies on three distinct problems exhibiting different merge-function properties. We additionally introduce a definition framework that characterizes these properties. Our results show that, counter-intuitively, sophisticated clustering does not always pay off, especially when the merge operator produces states that differ greatly from the originals. We provide a detailed analysis explaining when clustering is beneficial versus when simple cost-based strategies suffice, offering some guidelines for solver configuration.

Keywords: Decision Diagrams · Clustering · Branch-and-Bound · Generalized Hyperplan Partitioning · Discrete Optimization

1 Introduction

An elegant optimization framework called DDO, introduced in [4], solves discrete optimization problems formulated as dynamic programs (DPs) through a branch-and-bound (B&B) search enhanced with decision diagrams (DDs). At every explored node, DDO compiles a DD to obtain both primal and dual bounds. To prevent the combinatorial explosion inherent to exact DDs, their width is bounded, producing limited DDs created by dropping or merging nodes according to heuristics. Primal bounds come from restricted DDs [5], where nodes in oversized layers are dropped, yielding a diagram that contains only a subset of the feasible solutions of the exact DD. Dual bounds are obtained by merging nodes via a problem-specific operator, which may introduce infeasible paths [1, 7, 3, 27, 9]. The effectiveness of DDO strongly depends on the quality of these bounds and the efficiency with which they are computed.

The node selection heuristic for merging or discarding is typically based on cost[1] as originally proposed in [4] and reused in [13, 9, 18, 19]. This *Cost* based strategy is both cheap to compute and good at preserving promising paths from being discarded or merged. Recently, Nafar et al. [24] reconsidered this default choice and showed that clustering the nodes with k-means generally helps to produce tighter primal (resp. dual) bounds for restricted (resp. relaxed) DDs than *Cost* when applied at the root state. This paper is a follow-up work building on the *Clustering* strategy and aims to answer two questions:

– The approach in [24] assumes that states admit a Euclidean embedding (as in the multi-knapsack problem where the states are fixed-size vectors of remaining capacities). For some problems the state has a more complex structure (such as for the Maximum Coverage problem, where the state is naturally defined as a set of varying cardinality). The first question is thus whether the clustering approach of [24] can be generalized to richer state representations than Euclidean spaces?
– The analysis in [24] only focuses on root DDs. But DDs compiled deeper in the search space may exhibit different structures. A second question is whether the *Clustering* strategy consistently outperform the *Cost* strategy throughout the entire B&B search?

We answer the first question positively by introducing a new clustering method for DD compilation, based on Generalized Hyperplane Partitioning [29] (GHP). It supports clustering using a problem-specific state distance without requiring a Euclidean vector representation and also greatly reduces the computation time over the k-means clustering. If GHP is not a novel algorithm, to our knowledge it is the first time that it is applied to DDs.

To address the second question, we evaluate the method on three problems with distinct state structures: the knapsack, the multidimensional knapsack, and the maximum coverage problems. Results show that the benefit of *Clustering* over *Cost* strategy is highly problem-dependent; we provide guidelines indicating when it is likely to help based on the state and merge operator structure.

The paper is structured as follows. Section 2 introduces the optimization problems and their DD formulations. Section 3 reviews the related work. Section 4 presents the generalized hyperplane partitioning approach for DD layer clustering. Section 5 reports experimental results and evaluates the different strategies. We finally conclude in Section 6.

2 Background

A discrete optimization problem $\mathcal{P}$ is defined by a vector of variables $x = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ where each variable x_j can take a value in its domain D_j, a collection of constraints $\mathcal{C}$, and an objective function $f : \mathcal{D} = D_0 \times \ldots \times D_{n-1} \to \mathbb{R}$.

[1] The total cost of the lightest path from the root to the node.

A feasible solution of $\mathcal{P}$ is any complete assignment $x \in \mathcal{D}$ that satisfies all constraints of $\mathcal{C}$. Let $Sol(\mathcal{P}) \subseteq \mathcal{D}$ denote the feasible solution set of $\mathcal{P}$. The goal of the optimization problem is to find a feasible assignment $x \in Sol(\mathcal{P})$ that optimizes the objective function. Throughout the paper, we adopt minimization as the default objective orientation.

Based on the divide-and-conquer strategy, dynamic programming (DP) was introduced in [2] and used to solve discrete optimization problems. The problem is decomposed into small and overlapping subproblems solved recursively. To avoid multiple resolutions of the same subproblem, the intermediate results are stored in a cache. A discrete optimization problem $\mathcal{P}$ can be formulated in the DP formalism as a labeled transition system. It consists of:

– The control variables $x_j \in D_j$ used to specify the decision taken in the domain D_j with $j = 0, \dots, n-1$ and n the number of variables of the problem.
– The state space $\mathcal{S}$ contains partial assignments of the variables x. This space is divided in $n+1$ subset $S_0, S_1, \dots, S_n$, where S_j contains the states where exactly j variables among x are assigned. It also contains three special states: the *root* state $\hat{r}$, the *terminal* state $\hat{t}$ and the *infeasible* state $\hat{0}$.
– The *transition function* $t : S_j \times D_j \to S_{j+1}$ that maps each state of S_j and a decision in D_j to the corresponding state in S_{j+1}.
– The *transition value function* $h : S_j \times D_j \to \mathbb{R}$ that affects a value to each transition.

The optimization problem $\mathcal{P}$ can then be defined by:

$$\text{minimize } f(x) = \sum_{j=0}^{n-1} h(s^j, x_j) \tag{1}$$

$$\text{subject to } s^{j+1} = t(s^j, x_j), \forall j = 0, \dots, n-1, x_j \in D_j \tag{2}$$

$$x \in \mathcal{C}, s^j \in S_j, j = 0, \dots, n \tag{3}$$

2.1 Decision Diagrams-Based Optimization

Solving an NP-hard optimization problem formulated as a labeled transition system using only caching is likely to fail on difficult instances, because the cache may grow exponentially and no bounding mechanism is available to prune states. The B&B framework based on decision diagrams proposed in [4] introduces precisely these two missing ingredients. In this framework, the search space of the labeled transition system is explored using a classical B&B scheme in which the transition function serves as the node generator. At each node, both primal and dual bounds are computed in a generic way by exploiting the structure of the labeled transition system. These bounds are obtained by compiling it into two memory-bounded Decision Diagrams (DD): the restricted decision diagram and the relaxed decision diagram. The first one aims at finding primal bounds, while the second aims at finding a dual bound for the node. A search node is then fathomed whenever the dual bound is worse than the best so far incumbent. For the sake of completeness, we briefly recall the main ingredients on DD compilation from [4], adopting the same notation. A DD is a layered directed acyclic graph $\mathcal{B} = (N, A, \sigma, l, v)$ where N is the set of nodes interconnected by the set of arcs A. Each node u is mapped to a state $\sigma(u)$ by the function σ. The set of nodes N are partitioned as a collection of layers $L = \{L_0, L_1, \ldots, L_n\}$, where each layer usually corresponds to a decision on a variable (n is the number of decision variables in the DP model). The set of arcs A represents the transitions between consecutive states and an arc a is labeled by the function l which maps each arc a to a decision d i.e., $l(a) = d$ and a value function $v(a)$ assigns a cost to the corresponding transition. The layer L_0 contains only the node corresponding to the root state of the DP model, where L_n contains only the terminal nodes. The DD is compiled top-down, layer by layer, starting with L_0.

Algorithm 1: Compilation of a DD rooted at $u_{\hat{r}}$ with a W width limit

```
1  $i \leftarrow 0$
2  $L_i \leftarrow \{u_{\hat{r}}\}$
3  for $j = i$ to $n-1$ do
4      if $|L_j| > W$ then
5          Restriction or relaxation of layer $L_j$ with $W$
6      $L_{j+1} \leftarrow \emptyset$
7      forall $u \in L_i$ do
8          forall $d \in D_j$ do
9              create node $u'$ with $\sigma(u') = t_j(\sigma(u), d)$ or retrieve it from
                $L_{j+1}$
10             create arc $a = (u \overset{d}{\rightarrow} u')$ with $v(a) = h_j(\sigma(u), d)$ and $l(a) = d$
11             add $u'$ to $L_{j+1}$ and add $a$ to $A$
12 merge nodes in $L_n$ into terminal node $u_{\hat{t}}$
```

Because the width limit W restricts the layer sizes, the minimum-cost path in the resulting layered acyclic graph does not necessarily correspond to the exact optimal solution to the problem. The restricted strategy is used to obtain a primal bound, while the relaxed strategy is used to obtain a dual bound.

The Restricted DD compilation simply removes nodes from the current layer together with their incoming arcs. If a root-to-terminal path remains in the restricted DD, it can be used to compute a primal bound.

The Relaxed DD compilation limits the width of a layer L_j by merging nodes to create a *relaxed state* using a binary *merge operator* $\oplus : S \times S \rightarrow S$ that is problem specific. The application of the operator also redirects all the arcs to the new node. This operator should be such that no solution is removed (relaxation), although some infeasible solution might be introduced. Therefore, the shortest path on the relaxed decision diagram allows one to obtain a dual bound on the optimal cost. The infeasible state is absorbing for the merge operator, i.e., $s \oplus \hat{0} = \hat{0}\ \forall s \in \mathcal{S}$.

Cost-Based Heuristic The standard way to select the nodes of a layer that should be merged/dropped (introduced in [4], then also used in [19, 18, 13, 9]) is based on their costs from the root to the node. The $|L_j|-W$ nodes with the worst costs are dropped or merged. The two main advantages of this heuristic are that it is quick to compute and that it tends to preserve the best paths. The drawback is that merging so many nodes as a single state can end up with the introduction of a state that is quite different and much more relaxed from the others of the layer. Another disadvantage is that, being a very greedy strategy, it does not promote diversity among the states kept which could be detrimental for the subsequent decisions at deeper layers.

Clustering-Based Heuristic To address the limitations of the cost-based heuristic, [24] experimented with clustering the nodes based on their state similarity. In a relaxed DD, for each cluster, all its nodes are merged together, while for restricted DD, the best node of each cluster is selected based on its cost and the other ones are dropped. The clustering algorithm used in [24] is *k-means*. The drawback of this approach is that *k-means* requires the state to be of fixed numerical dimensions in a Euclidean space. Clustering-based strategies for compiling relaxed diagram compilation were also used in [11, 12].

In the rest of this paper, we denote by *DDO Model* the pair composed of the DP formulation of a problem and a merge operator.

2.2 DDO Models Characterization

We attempt to classify the DDO model of optimization problems. This characterization depends on the *nontrivial states* and the merge operator of the model.

Definition 1. *A* nontrivial state *s is a state different from the root state, the terminal state, and the infeasible state, i.e., $s \notin \{\hat{r}, \hat{t}, \hat{0}\}$.*

Definition 2. *Let $\mathcal{M}$ be a DDO model, $\mathcal{S}$ and $\oplus$ be the state space and the merge operator associated to $\mathcal{M}$. A nontrivial state $s \in \mathcal{S}$ is said to be strongly preserved by $\oplus$ if there is no nontrivial state $s' \in \mathcal{S}$ such that $s \oplus s' \in \{\hat{r}, \hat{t}\}$. Conversely, a state s is said to be weakly preserved by $\oplus$ if it is not strong, i.e., there exist a state $s' \in \mathcal{S}$ such as $s \oplus s' \in \{\hat{r}, \hat{t}\}$*

In this definition, the frequency with which the merge operator will bring the merged states back to trivial states can be estimated for each model. These frequencies can be grouped into three broad categories: never, often, and always.

Definition 3. *A DDO model of an optimization problem is said to be*

- state-preserving *if all its nontrivial states are strongly preserved by its merge operator, or*
- slightly state-preserving *if some of its nontrivial states are strongly preserved by its merge operator while others are weakly preserved, or*
- state-altering *if all its nontrivial states are weakly preserved by its merge operator.*

2.3 Problems of interest

We formalize DP models for three different discrete optimization problems selected for covering the three situations of Definition 3.

The Knapsack Problem (KP) is defined by the given of n items, each item j has a value v_j and a weight w_j. The goal is to select a subset of these to maximize the total value without exceeding a capacity C. In DP formulation of KP, the state is a positive integer that represents the remaining capacity:

– $x_j \in \{0,1\}$ with $j \in \{0,\dots,n-1\}$ is set to 1 when the j^{th} item is taken ($s \geq w_j$) and 0 otherwise.
– $S_j = \{s \mid 0 \leq s \leq C\}$ where s is a positive integer representing the remaining capacity. The root state $\hat{r} = C$ is the state of initial capacity and the terminal state is any state $\hat{t} = c$ with $c < \min\{w_i \mid 0 \leq i < n\}$.
– If the remaining capacity of a state s does not allow including the j^{th} item, the transition is redirected to infeasible state $\hat{0}$.

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

– $h(s,x_j) = -v_j \cdot x_j$ is the value added to the objective for each transition.

The operator used to merge states of this model is the maximum remaining capacity of states, i.e., $s \oplus s' = \max(s,s')$. This model of KP is *state-preserving* because $s \oplus s' = \hat{r} \iff s = \hat{r}$ or $s' = \hat{r}$.

The Multidimensional Knapsack Problem (MKP) is a generalization of the KP to multiple capacity constraints: n items and m dimensions of the knapsack are given, each dimension with capacity bound $(C_1,\dots,C_m)$. An item j occupies a weight dimension w_j^i in the knapsack for $i \in \{1,\dots,m\}$ for a value of v_j. The goal is to select a subset of items whose sum of profits is maximized so that all the capacity bound constraints hold simultaneously. The state in the MKP is an m-tuple $(c_1,\dots,c_m)$ of positive integers that represents the remaining capacity of each dimension of the knapsack.

– $x_j \in \{0,1\}$ with $j \in \{0,\dots,n-1\}$ is set to 1 when the j^{th} item is taken ($s_i \geq w_j^i$ for all $i \in \{1,\dots,m\}$) and 0 otherwise.
– $S_j = \{(s_1,\dots,s_m) \mid 0 \leq s_i \leq C_i$ with $i = 1,\dots,m\}$ where s_i is a positive integer representing the remaining capacity of the i^{th} dimension of the knapsack where $i \in \{1,\dots,m\}$. The root state is $\hat{r} = (C_1,\dots,C_m)$ and the terminal state is any state $\hat{t} = (c_1,\dots,c_m)$ with $c_i < \min\{w_j^i \mid 0 \leq j < n\}$ and $i \in \{1,\dots,m\}$.
– If the remaining capacity of at least one dimension s_i with $i \in \{1,\dots,m\}$ does not allow inclusion of the j^{th} item, the transition is redirected to infeasible state $\hat{0}$.

$$t(s,x_j) = \begin{cases} (s_1 - w_j^1 \cdot x_j,\dots,s_m - w_j^m \cdot x_j) & \text{if } s_i \geq w_j^i \cdot x_j \forall i \in \{1,\dots,m\} \\ \hat{0} & \text{otherwise.} \end{cases}$$

– $h(s, x_j) = -v_j \cdot x_j$ is the value added to the objective for each transition.

In this model, the operator used to merge states is the maximum remaining capacity for each dimension, i.e., $s \oplus s' = (\max(s_1, s'_1), \dots, \max(s_m, s'_m))$. Without loss of generality, we consider only two dimensions of capacity C_1 and C_2. A nontrivial state (c_1, c_2) with $c_1 < C_1$ and $c_2 < C_2$ is *strongly preserved* by $\oplus$. However, any nontrivial state of the form (c_1, C_2) or (C_1, c_2) where $c_1 < C_1$ or $c_2 < C_2$, is *weakly preserved* since $(c_1, C_2) \oplus (C_1, c') = (C_1, C_2)$ or $(C_1, c_2) \oplus (c', C_2) = (C_1, C_2)$ for all $c' < C_2$ or $c' < C_1$. Thus, the DDO model of MKP proposed in the paper is *slightly state-preserving.*

The Maximum Coverage Problem (MCP) is defined by giving a finite set U of n elements, a finite collection of m subsets $V = \{V_1, \dots, V_m\}$ of U, and an integer $1 \leq k \leq m$. The goal is to find a sub-collection $C \subset V$ such that $|C| = k$ that maximizes the total number of elements covered by this subset. In DP formulation, the states are simply the set of covered elements and the DD has exactly $k+1$ layers:

– $x_j \in \{1, \dots, m\}$ where x_j is the index of a subset that covers at least one new element not present in the current state s, i.e., $V_{x_j} \setminus s \neq \emptyset$.
– $S_j = 2^U$ for $j = 0, \dots, k$, $\hat{r} = \emptyset$ and $\hat{t}$ is any state union of k subsets of V.
– $t(s, x_j) = s \cup V_{x_j}$.
– $h(s, x_j) = -|V_{x_j} \setminus s|$, that is the new added elements to the state.

In this model states are merged by computing the intersection of their sets of covered elements, i.e., $s \oplus s' = s \cap s'$. For every nontrivial state s there exists a state s' such as $s \cap s' = \emptyset = \hat{r}$, hence this model is *state-altering.*

This problem could also be represented with a BDD, where each layer represents the decision of selecting or discarding a particular set; the DD would thus contain exactly $m+1$ layers. But early results showed that the MDD model was able to produce tighter bounds despite its high branching factor: usually $k << m$, hence the BDD model is significantly deeper than the MDD, making its paths more susceptible to deterioration.

Example 1. Consider an instance of the KP with $n = 5$, $C = 12$, $v = (5, 4, 3, 6, 8)$ and $w = (3, 5, 6, 4, 5)$. The first four layers of exact DD (Figure 1a) and relaxed cost DD (Figure 1b) for a maximum width of 3 are shown in Figure 1.

Each node represents a state, with transition costs assigned to the arcs. The bold number near a node indicates the path length from the root to that node, while the shortest path is highlighted by bold arrows. The value at a node and the transition cost are negative, since the KP is a maximization problem and we model it as a minimization one.

Layer 2 of the exact DD ($L_2 = \{4, 9, 7, 12\}$) of Figure 1a is sorted according to the node cost. For a maximum width of 3, nodes 7 and 12 are merged, and the resulting upper bound is 21.

The clusters obtained in layer 2 ($L_2 = \{4, 9, 7, 12\}$) of the exact DD of Figure 1a based on the *k-means* are $\{\{4\}, \{9, 7\}, \{12\}\}$ resulting in the relaxed clustering DD (Figure 1c) of upper bound 19 after compilation. In this relaxed DD,

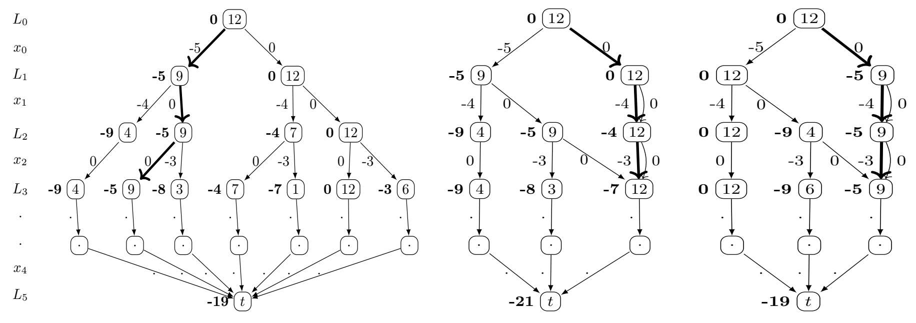

(a) Exact (b) Relaxed Cost (c) Relaxed Clust.

Fig. 1: Decision diagrams of the KP described in the Example 1. For the relaxed DDs $W = 3$.

the path $\hat{r} \rightsquigarrow \hat{t}$ of the minimal length is a feasible solution, since the merged states are exact, which proves the optimality of the solution.

3 Related Work

Alternative node selection heuristics have been explored in several previous works: in [17], the authors break ties in cost-based merging using a distance metric, and test it on the Minimum Independent Set Problem and the Set Covering Problem, two state-altering models. It outperforms the basic cost strategy but still relies on it, contrary to the clustering-based approach evaluated in our paper. In [23] nodes are merged according to their shared children in the next layer. It outperforms the cost strategy on the Knapsack problem, but is not compared to clustering. In [15] the Quadratic Knapsack is solved with a DD using Kruskal-based clustering with a distance metric, but unlike the approaches evaluated in our paper, the produced layers may exceed the maximal width. Their results don't isolate the merging strategy's impact. In [16] the authors select the merging heuristic to use on a layer based on a lookahead of the next layers. Some other works also focused on alternatives to compute dual bounds: in [20, 21], the authors proposed the so-called *Domain-Independent Dynamic Programming* paradigm to solve combinatorial optimization problems by avoiding the need for domain-specific state merging but relying on domain-specific lower-bound heuristics to guide the search using A* like algorithms [20, 21]. In contrast, traditional Decision Diagram Optimization (DDO) methods [4] require explicit specification of state merging operators to construct relaxations.In [18], the authors also include the possibility of domain-specific lower-bounds. Some other related works have focused on enhancing dual bounds derived from relaxed decision diagrams. In [28], the diagram is not compiled from scratch at every node but rather refined incrementally to infer the dual bound and save computation efforts. In [27] a local search framework is proposed to strengthen the dual bound of the relaxed DD. In [6] a MIP model is proposed to discover the provably tightest dual-bound from a relaxed DD.

4 Using Generalized Hyperplane Partitioning for DD

The *k-means* clustering algorithm used by [24] requires mapping each state to a fixed-size coordinate vector so that it can compute iteratively new centroids with coordinate averaging and also use the default Euclidian distance metric to allocate points to the centroids. We propose to replace *k-means* to address two primary limitations: (i) Computational Cost: The overhead of *k-means* becomes significant as the number of state increases. For combinatorial problems like the Maximum Coverage Problem (MCP), this cost is far from negligible, as reported in the experimental section. (ii) Lack of Flexibility: The strict requirement for vector representations and Euclidean distances is a limitation in generic solvers such as [19, 22] since this representation is not natural for problems such as the ones with more complex states such as sets for the MCP. Adapting set-based states to *k-means* requires embedding them into fixed-size vectors over the entire universe, leading to high-dimensional encodings that suffer from the curse of dimensionality when computing Euclidean distances. Instead, a more general approach should support arbitrary, problem-specific metrics (we refer to [14] for a set of metrics on sets). To satisfy these requirements for speed and flexibility, we use the *Generalized Hyperplane Partitioning* (GHP) algorithm introduced in [29] and improved in [8]. GHP is presented in Algorithm 2 in the context of DD. The algorithm assumes $|L_i| > W$. GHP starts from a single cluster containing all the states of L_i, and iteratively selects a cluster to split into two until the desired number of clusters W is obtained. The distance metric used is denoted *dist*. The set of clusters C is stored as a priority queue sorted in decreasing order of the cluster diameters (denoted $\delta(c)$), so that clusters whose states are widely separated are split first. When a cluster c is extracted from C (Line 3), the auxiliary function *computePivots* (Line 5) identifies two states p_1 and p_2 whose distance is ideally close to the diameter of c ($\text{dist}(p_1, p_2) \approx \delta(c)$). The algorithm then partitions the states of

c into two subclusters according to their proximity to the pivot states (Line 7).

Algorithm 2: Generalized Hyperplane Partitioning

```
1  $C \leftarrow \{L_i\}$
2  while $|C| < W$ do
3      $c \leftarrow extractMax(C)$  // Pull the cluster of maximal diameter
4      $c_1, c_2 \leftarrow \emptyset$
5      $p_1, p_2 \leftarrow computePivots(c)$
6      for $v \in c$ do
7          if $dist(p_1, v) \leq dist(p_2, v)$ then
8              $c_1 \leftarrow c_1 \cup \{v\}$
9          else
10             $c_2 \leftarrow c_2 \cup \{v\}$
11     $C \leftarrow C \cup \{c_1, c_2\}$
12 return $C$
```

The function *computePivots* in Algorithm 3 returns two states p_1, p_2 such that $dist(p_1, p_2) \approx \delta(c)$. To optimize the computation speed, this is done in three consecutive steps. First a state is randomly selected (Line 1). The first pivot p_1 is the one farthest from it (Line 2). Finally, the second pivot p_2 (Line 3) is one farthest from p_1. Again, to speed up the computation, we use an approximation of the diameter to prioritize the clusters in C. It is the largest distance between the pivot and any state in the cluster. This incurs no additional computational overhead, as it is done while executing the inner loop of Algorithm 2.

Algorithm 3: Compute Pivots

```
1  $v \leftarrow RandomSelect(c)$
2  $p_1 \leftarrow \arg\max\{dist(u, v) \mid u \in c\}$
3  $p_2 \leftarrow \arg\max\{dist(u, p_1) \mid u \in c\}$
4  return $p_1$, $p_2$
```

If we denote by $\mathcal{O}(dist)$ the complexity used to compute the distance between two states, then the complexity of Algorithm 3 is $\mathcal{O}(dist \times |L_i|)$. In Algorithm 2, the inner loop calls the metric of $\mathcal{S}$. The outer loop is repeated W times, thus W extractions and $2W$ insertions are performed on the priority queue, each with a $\mathcal{O}(\log(W))$ complexity. Therefore, the overall complexity of the Algorithm 2 is $\mathcal{O}(W(\log(W) + dist \times |L_i| + dist \times |L_i| + 2\log(W))) = \mathcal{O}(W(\log(W) + dist \times |L_i|))$.

Figure 2 illustrates the different steps of the GHP function. The set C currently contains two clusters (blue and orange), and the largest is pulled (Figure 2a). At the second step (Figure 2b), pivot states are computed and the last step (Figure 2c) splits the blue cluster into yellow and green clusters.

Example 2. Let us consider an instance of MCP with $U = \{0, 1, 2, 3, 4\}$, $V = \{\{0, 2\}, \{1, 3\}, \{3\}, \{1, 4\}\}$ and $k = 3$. Figure 3a illustrates the exact DD, the relaxed *Cost* DD (Figures 3b) and the relaxed *clustering* DD (3c) for a maximum width $W = 2$ of the instance. For GHP, the distance between two states is the

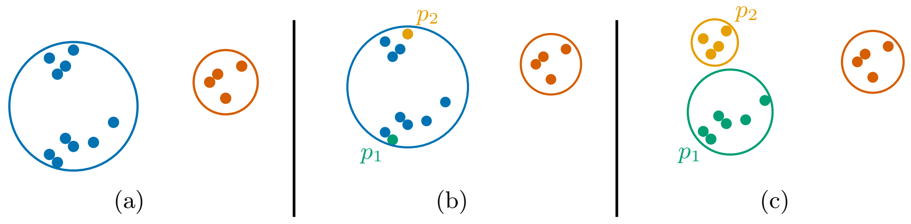
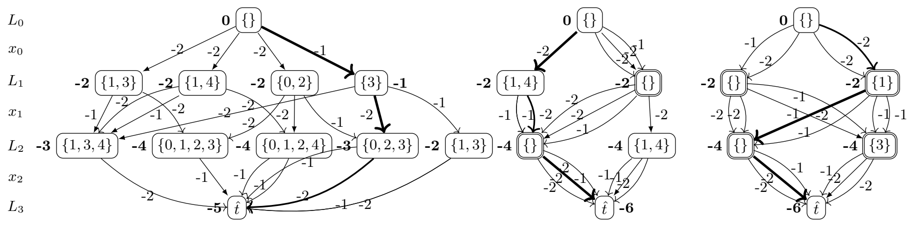


Fig. 2: Example of the GHP procedure.

size of their symmetric difference, normalized by the size of the universe[2]. Merged



(a) Exact (b) Relaxed Cost (c) Relaxed Clust.

Fig. 3: Decision diagrams of the MCP described in Example 2. For the relaxed DDs $W = 2$.

nodes have a double border. In the relaxed diagrams, several relaxed nodes are *trivial nodes* regardless of the heuristic used for relaxation. This reflects the *state-altering* nature of the DDO model for the MCP.

5 Experimental Evaluation

The goal of this section is to evaluate the behavior of clustering heuristics on the three problems of interests KP, MKP, and MCP. We denote by GHP and KMEANS the clustering approaches based on GHP and on k-means (as in [24]), respectively. Additionally two heuristics are added. One is a generalization of COST and GHP in which the $\alpha \cdot W$ nodes are selected according to their cost and the remaining nodes are grouped into $(1 - \alpha) \cdot W$ clusters using GHP with $\alpha \in [0, 1]$. This heuristic is denoted α-HYBRID. For KP and MCP, it gives bounds between COST and GHP: as α approaches 1, the bounds align with COST's ones; as it decreases, with GHP's ones. But for MKP 0.6-HYBRID shows better performances. For readability, we thus display only the performances of 0.6-HYBRID in the following results. Finally, a baseline strategy, where W nodes

[2] $d(A, B) = \frac{|(A \cup B) - (A \cap B)|}{|U|}$, where A and B are two subsets of U

are randomly selected, denoted RANDOM, is also considered to evaluate if an extreme diversification of the states retained in the DD is useful.

Regarding the metric applied to each problem: for both KP and MKP, we use the Euclidean distance, which is normalized by the maximum distance possible for the instance, i.e., the Euclidean distance between the root state $\hat{r}$ and a state where all the remaining capacities are null. For the MCP, we experimented with several distance metrics: the Jaccard distance, the Dice distance [14] and the size of the symmetric difference, normalized by the size of the universe.Preliminary results showed that the best bounds were obtained with the normalized symmetric difference. Concerning *k-means* on MCP, states are represented with boolean vectors where the value at coordinate i is 1 if i is covered in the state, 0 otherwise. For MKP, as done in [24], we add the objective function associated with a node to the features considered by *k-means*.

Our experiments aim to answer the following questions, for which we already provide synthetic answers based on the detailed results presented after:

Question 1 Does clustering help obtain tighter lower bounds in relaxed DD?
Answer: Yes for *state-preserving* models (e.g., KP), but not for *state-altering* ones (e.g., MCP).

Question 2 Do GHP and KMEANS obtain the same bound quality and do they execute at the same speed?
Answer: In terms of speed, there is no major difference with small branching factors (KP, MKP), but GHP is superior when the branching factor is large (MCP). The bound quality is roughly the same with GHP and KMEANS.

Question 3 Does clustering help obtain tighter upper bounds in restricted DD?
Answer: Yes, for all three problems.

Question 4 Is clustering worthwhile in a complete B&B search despite its heavier computational cost?
Answer: Yes for *state-preserving* models; but the performances observed for the MCP suggest that for *state-altering* models the tradeoff is worthless, as performance gain obtained by the stronger pruning depends on the bound quality. The root bound quality seems to be a good estimator of it.

We implemented the approach using an open-source Java version[3] of [19]. Our source code and instances are available on the same repository. All the experiments were executed on an Intel Skylake 16-cores Xeon 6142 processor at 2.6 GHz with up to 192 GB of RAM[4]. For the *k-means* algorithm, we use the implementation available in the SMILE library[5]. Concerning the maximum number of iterations of the *k-means* algorithm, we experimented with several values, but observed that a high number of iterations increases the time needed to compute the DDs while not having a significant impact on the quality of the

[3] Available at https://github.com/DDOLIB-CETIC-UCL/DDOLib

[4] Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

[5] https://github.com/haifengl/smile

bound. According to these observations, the maximum number of iterations of the *k-means* is set to 5 in our experiments. We use the benchmark of 100 KP instances from [25] with 200 items per instance and the 270 MKP instances from [10][6] with 100 items and 5 dimensions. For MCP benchmark, 300 instances of $|U| \in \{100, 150, 200\}$ elements were generated following [26]. For all problems, variables are sorted in lexicographical order.

5.1 Relaxed DD

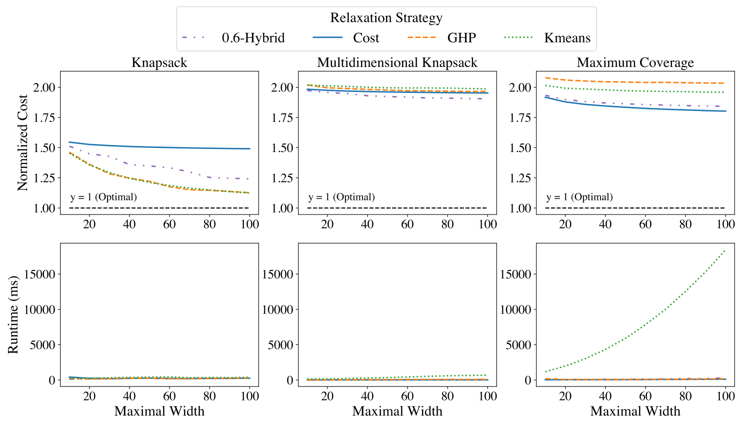


Fig. 4: Average bound quality and runtime versus the maximal width for Relaxed DDs, depending on the heuristic.

Figure 4 compares, for the different heuristics, the average normalized bound (wrt to the optimal solution) at the root node obtained with the relaxed DD and the runtime used to compute it for different values of maximum width W (x-axis). The observations for the three problems are:

KP Both Kmeans and GHP allow obtaining significantly better bounds than Cost. The quality of the bounds is similar for Kmeans and GHP. Execution times are all similar.

MKP The three strategies give very similar bounds[7], but the ones offered by the Cost are slightly better. Interestingly, the 0.6-Hybrid obtains the best bounds among all strategies. In terms of runtime Kmeans is slightly slower.

[6] Available at https://people.brunel.ac.uk/ mastjjb/jeb/orlib/mdmkpinfo.html

[7] Those results are a bit different from those reported in [24], where Kmeans achieves a tighter bound than Cost. When run on the same set of instances, our implementation produces comparable outcomes. However, our instance pool is larger and includes all of theirs. Thus, we believe that our results provide a more comprehensive view of the heuristics' performance.

MCP The Cost strategy offers a significantly better bound than the clustering approaches. Kmeans is much slower than GHP that keeps a runtime close to Cost. Kmeans has a slightly better bound quality than GHP though.

For a better understanding of the impact of the heuristics on the relaxed DD, we measure for each instance the proportion of exact nodes and the average degradation of each state defined as its distance with respect to the result of the merge. When the node is exact, its degradation is equal to 0. We also measure the average distance between each state of the DD and the root state. By design, the merge operators for the three problems result in states more similar to the root state. Hence, a relaxed DD with highly deteriorated states will contain states closer to the root state. Table 1 shows statistics on the distribution across the different instances of these measures, for relaxed DDs with a 60 maximal width.

Problem	Strategy	Proportion of Exact Nodes			Avg State Degradation			Avg Distance with Root		
		Min	Med	Max	Min	Med	Max	Min	Med	Max
KP	Cost	**0.05**	**0.51**	**0.98**	0.12	0.24	0.30	0.19	0.51	0.57
	GHP	0.01	0.01	0.07	**0.00**	**0.00**	**0.00**	**0.45**	**0.63**	**0.80**
	Kmeans	0.01	0.01	0.07	**0.00**	**0.00**	**0.00**	**0.43**	**0.63**	**0.80**
MKP	Cost	**0.20**	**0.48**	**0.77**	0.19	0.22	0.25	0.44	0.53	0.60
	GHP	0.00	0.01	0.04	**0.00**	**0.01**	**0.03**	0.46	0.77	0.92
	Kmeans	0.00	0.01	0.04	**0.00**	**0.01**	**0.03**	**0.56**	**0.83**	**0.94**
MCP	Cost	**0.13**	**0.31**	**0.98**	0.13	0.29	0.45	**0.11**	**0.24**	**0.39**
	GHP	0.12	0.21	0.37	0.05	0.11	0.18	0.01	0.04	0.09
	Kmeans	0.08	0.13	0.24	**0.03**	**0.07**	**0.12**	0.02	0.06	0.12

Table 1: Statistics on the distribution among the problem instances of the proportion of exact nodes, the average state degradation and the average distance between states and root in the relaxed DDs with a maximal width set to 60.

For KP and MKP, clustering methods (GHP and Kmeans) greatly reduce state deterioration compared to Cost. This is less true for MCP, where the distance of the states to the root is very small when clustering is used, which explains why the lower bounds are weaker on this problem with clustering-based merge strategies.

5.2 Restricted DD

Figure 5 compares the average normalized primal bound obtained with restricted DDs and the runtime, using the different heuristics on the benchmarks.

KP Using Clustering significantly improves (near-optimal solutions found) the primal bounds obtained with restricted DDs. Random selection (Random) performs worse, confirming that the extra cost of Clustering is worthwhile.

- **MKP** The gap between Cost and Clustering is smaller than in KP. However, Kmeans and GHP return significantly better bounds than Cost. In terms of computational cost, Kmeans is more expensive, especially compared to GHP and Cost.
- **MCP** All heuristics return close to optimal solutions. Kmeans is notably more costly, while GHP and Cost have similar performance. Clustering still provides slightly tighter bounds, and Random is the worst heuristic.

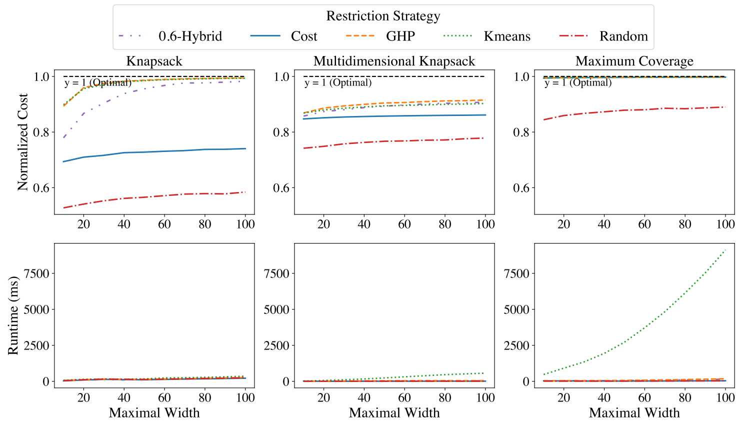


Fig. 5: Average bound quality and runtime versus the maximal width for Restricted DDs, depending on the heuristic.

5.3 Complete Branch-and-Bound

Figure 6 compares the different heuristics on a complete B&B search. A time limit of 5 minutes is set per instance, and the maximum width is fixed at $W = 60$ for both relaxed and restricted DDs. The cumulative number of instances solved over time (x-axis) is reported for the KP, and for MKP and MCP it shows the number of instances solved with an optimality gap below the threshold (x-axis) at the timeout, as these problems are mostly not solved optimally.

- **KP** The heuristic used for the restricted DD has a strong impact on the runtime. Using Clustering significantly reduces runtime compared to Cost. Overall, the best configuration is GHP for both relaxed and restricted DDs, as it improves bounds at the root and reduces solution time.
- **MKP** All configurations show similar performances. Only the heuristic used for the restricted DD affects performance slightly, with Cost being slightly disadvantaged and Clustering slightly reducing the optimality gap. The relaxed DD heuristic has minimal impact.

MCP All heuristics produce near-optimal bounds for the restricted DD, so the choice of restricted DD heuristic has little effect on performance. The relaxed DD heuristic affects performance slightly, with Cost giving the best bound and slightly improving overall performance compared to GHP or Kmeans.

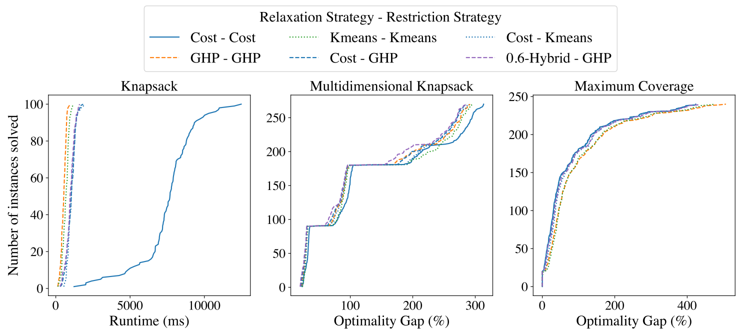


Fig. 6: Number of instances solved by each configuration for the three different problems with respect to the runtime or the optimality gap.

6 Conclusion

We proposed using the Generalized Hyperplane Partitioning (GHP) as a faster, more flexible alternative to k-means proposed in [24] for clustering states in bounded-width decision diagrams (DD). It is particularly efficient for problems with high branching factors (typically MDD rather than BDD). We also study the impact of using clustering as a heuristic for node selection when computing bounded-width DDs. We observe that for Restricted DDs (primal bounds), it offers tighter bounds than cost-based methods. However, for Relaxed DDs (dual bounds), its success is tied to the merge operator: clustering outperforms cost-based methods on Knapsack, a *state-preserving* model, but underperforms on Maximum Coverage, a *state-altering* model, where merging strongly dilutes the bounds. Consequently, we recommend using clustering for primal bounds, but selectively for dual bounds only when the merge operator generates states relatively close to the originals; otherwise, the standard cost-based heuristic introduced in [4] seems to remain superior. In future work, we intend to evaluate additional models across each problem class to strengthen our conclusions.

References

1. H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A Constraint Store Based on Multivalued Decision Diagrams. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 118–132, Berlin, Heidelberg, 2007. Springer.

2. Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.
3. David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Optimization Bounds from Binary Decision Diagrams. *INFORMS Journal on Computing*, August 2013. Publisher: INFORMS.
4. David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete Optimization with Decision Diagrams. *INFORMS Journal on Computing*, 28(1):47–66, February 2016. Publisher: INFORMS.
5. David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and Tallys Yunes. BDD-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, April 2014.
6. David Bergman and Andre Augusto Cire. On finding the optimal bdd relaxation. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 41–50. Springer, 2017.
7. David Bergman, Willem-Jan van Hoeve, and John N. Hooker. Manipulating MDD Relaxations for Combinatorial Optimization. In Tobias Achterberg and J. Christopher Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 20–35, Berlin, Heidelberg, 2011. Springer.
8. Edouard Bugnion, T Roos, F Shi, Peter Widmayer, and Felizitas Widmer. A spatial index for approximate multiple string matching. In *Proceedings of the 1st South American Workshop on String Processing (SP 1993), Belo Horizonte, Brazil*, pages 43–53. First South American Workshop on String Processing, 1993.
9. Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. Decision Diagrams for Discrete Optimization: A Survey of Recent Advances. *INFORMS Journal on Computing*, 34(4):2271–2295, July 2022. Publisher: INFORMS.
10. Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.
11. Andre A. Cire and Willem-Jan van Hoeve. Multivalued Decision Diagrams for Sequencing Problems. *Operations Research*, 61(6):1411–1428, December 2013. Publisher: INFORMS.
12. Vianney Coppé, Xavier Gillard, and Pierre Schaus. Boosting Decision Diagram-Based Branch-And-Bound by Pre-Solving with Aggregate Dynamic Programming. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969.
13. Vianney Coppé, Xavier Gillard, and Pierre Schaus. Decision Diagram-Based Branch-and-Bound with Caching for Dominance and Suboptimality Detection. *INFORMS Journal on Computing*, 36(6):1522–1542, November 2024. Publisher: INFORMS.
14. Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009.
15. M. Eliass Fennich, Leandro C. Coelho, and Franklin Djeumou Fomeni. Tight upper and lower bounds for the quadratic knapsack problem through binary decision diagrams. *Computers & Operations Research*, 184:107197, 2025.
16. Nikolaus Frohner and Günther R. Raidl. Merging quality estimation for binary decision diagrams with binary classifiers. In Giuseppe Nicosia, Panos Pardalos, Renato Umeton, Giovanni Giuffrida, and Vincenzo Sciacca, editors, *Machine Learning, Optimization, and Data Science*, pages 445–457, Cham, 2019. Springer International Publishing.

17. Nikolaus Frohner and Günther R. Raidl. Towards improving merging heuristics for binary decision diagrams. In Nikolaos F. Matsatsinis, Yannis Marinakis, and Panos Pardalos, editors, *Learning and Intelligent Optimization*, pages 30–45, Cham, 2020. Springer International Publishing.
18. Xavier Gillard, Vianney Coppé, Pierre Schaus, and André Augusto Cire. Improving the Filtering of Branch-and-Bound MDD Solver. In Peter J. Stuckey, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 231–247, Cham, 2021. Springer International Publishing.
19. Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a Generic and Efficient Framework for MDD-Based Optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 5243–5245, Yokohama, Japan, July 2020. International Joint Conferences on Artificial Intelligence Organization.
20. Ryo Kuroiwa and J Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 236–244, 2023.
21. Ryo Kuroiwa and J Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 245–253, 2023.
22. Laurent Michel and Willem-Jan van_Hoeve. Codd: A decision diagram-based solver for combinatorial optimization. 2024.
23. Mohsen Nafar and Michael Römer. Lookahead, merge and reduce for compiling relaxed decision diagrams for optimization. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 74–82, Cham, 2024. Springer Nature Switzerland.
24. Mohsen Nafar and Michael Römer. Using Clustering to Strengthen Decision Diagram Bounds for Discrete Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8082–8089, March 2024. Number: 8.
25. David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.
26. Mauricio G. C. Resende. Computing approximate solutions of the maximum covering problem with GRASP. *J. Heuristics*, 4(2):161–177, 1998.
27. Michael Römer, André A. Ciré, and Louis-Martin Rousseau. A local search framework for compiling relaxed decision diagrams. In *CPAIOR*, volume 10848 of *Lecture Notes in Computer Science*, pages 512–520. Springer, 2018.
28. 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.
29. Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, November 1991.