Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Relaxed BDDs: An Admissible Heuristic for Delete-Free Planning Based on a Discrete Relaxation

Margarita P. Castro,[1] Chiara Piacentini,[1] Andre A. Cire,[2] J. Christopher Beck[1]
[1]Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada, ON M5S 3G8,
[2]Department of Management, University of Toronto Scarborough, Toronto, Canada, ON M1C 1A4
mpcastro@mie.utoronto.ca, chiarap@mie.utoronto.ca, acire@utsc.utoronto.ca, jcb@mie.utoronto.ca

Abstract

We investigate the use of relaxed binary decision diagrams (BDDs) as an alternative to linear programming (LP) for computing an admissible heuristic for the cost-optimal delete-free planning (DFP) problem. Our main contributions are the introduction of a novel BDD encoding, a construction algorithm for the sequential relaxation of a DFP task and a study of the effectiveness of relaxed BDD heuristics, both from a theoretical and practical perspective. We further show that relaxed BDDs can be used beyond heuristic computation to extract delete-free plans, find action landmarks, and identify redundant actions. Our empirical analysis shows that while BDD-based heuristics trail the state of the art, even small relaxed BDDs are competitive with the LP heuristic for the DFP task.

1 Introduction

Cost-optimal delete-free planning (DFP) is a variant of classical planning that omits the delete effects of actions. While NP-hard (Bylander 1994), DFP has been extensively investigated by the planning community as the basis for efficient methodologies to address classical planning problems (Betz and Helmert 2009; Helmert and Domshlak 2009).

Previous work has shown that admissible heuristics for classic planning can be represented as a linear programming (LP) model and used to obtain state-of-the-art performance (Pommerening et al. 2014). In this work, we explore the effectiveness of *relaxed binary decision diagrams* (BDDs) as an alternative to LPs for computing admissible heuristics for DFP tasks. A relaxed BDD is a graphical structure that encodes a superset of the solutions of a discrete problem. They have recently formed the core of state-of-the-art methodologies in a variety of combinatorial optimization problems (Bergman et al. 2016).

Contributions. We present a novel BDD representation that encodes a sequential relaxation of a DFP task, investigating its structural properties and presenting a numerical comparison to the DFP LP relaxation (Imai and Fukunaga 2014). Specifically, we propose a BDD construction that guarantees the admissibility and consistency of our heuristic and explore theoretical properties of the resulting relaxed BDD with respect to the maximum size needed to represent only valid plans. Furthermore, we show how to leverage the graphical structure to identify landmarks and redundant actions, and also to extract delete-free plans.

We include an extensive empirical analysis that highlights the advantages and disadvantages of using an heuristic based on our relaxed BDD as opposed to the LP relaxation. We show that our heuristic has competitive performance solving DFP tasks. Namely, while still not the state of the art, our approach outperforms the LP relaxation in nine delete-free IPC domains and the mixed-integer programming model (Imai and Fukunaga 2014; 2015) in one.

2 Background

This work considers cost-optimal DFP using the STRIPS formalism restricted to tasks with no negative preconditions and no conditional effects.

A DFP task is given by a tuple $\Pi = \langle \mathcal{P}, s_I, \mathcal{G}, \mathcal{A} \rangle$ where $\mathcal{P}$ corresponds to the set of propositions, s_I is the initial state, $\mathcal{G} \subseteq \mathcal{P}$ is the set of goal propositions, and $\mathcal{A}$ is the set of actions. We define a state s by its set of true propositions, i.e., we say that $p \in s$ if p is true in s.

An action $a \in \mathcal{A}$ is a tuple $\langle \texttt{pre}(a), \texttt{add}(a), \texttt{c}(a) \rangle$, where $\texttt{pre}(a) \subseteq \mathcal{P}$ is the set of preconditions, $\texttt{add}(a) \subseteq \mathcal{P}$ is the set of additive effects, and $\texttt{c}(a) \geq 0$ corresponds to the action cost. We assume, without loss of generality, that $\texttt{add}(a) \cap \texttt{pre}(a) = \emptyset$ for all $a \in \mathcal{A}$. We say that an action a is applicable to a state s if its preconditions are true in s, i.e., $\texttt{pre}(a) \subseteq s$. Given a state s and an applicable action a, the successor state s' is given by $s' = \texttt{succ}(a, s) = s \cup \texttt{add}(a)$. In general, we say that a sequence of actions $(a_1, \ldots, a_n)$ is applicable to a state s if a_1 is applicable in s and each a_i is applicable in $\texttt{succ}((a_1, \ldots, a_{i-1}), s)$.

Given a DFP task Π, we define a plan $\pi = (a_1, ..., a_n)$ as a sequence of applicable actions from the initial state s_I such that $s_G = \texttt{succ}(\pi, s_I)$ satisfies all goal propositions, i.e., $\mathcal{G} \subseteq s_G$. The cost of a plan is given by $\texttt{c}(\pi) := \sum_{a \in \pi} \texttt{c}(a)$. In particular, a cost-optimal plan π^* for Π is a plan with minimal cost, i.e, $\texttt{c}(\pi^*) \leq \texttt{c}(\pi)$ for all plans $\pi \in \Pi$.

We say that $p \in \mathcal{P}$ is a *propositional landmark* if $p \in \texttt{succ}(\pi, s_I)$ for all plans $\pi \in \Pi$. Similarly, $a \in \mathcal{A}$ is an *action landmark* if $a \in \pi$ for all $\pi \in \Pi$.

2.1 Sequential Relaxation

The *sequential relaxation* of a DFP task, also known as a temporal relaxation, ignores the order in which the actions are applied (Imai and Fukunaga 2014).

Definition 2.1. Given a DFP task Π, a valid sequence-relaxed plan for Π, sr-plan, is a set of actions $\pi_{sr} = \{a_1, \ldots, a_n\}$ such that (i) for every $a \in \pi_{sr}$, each $p \in \mathtt{pre}(a)$ is true in s_I or is added by some action $a' \in \pi_{sr}$ and (ii) each goal $p \in \mathcal{G}$ is true in s_I or is added by some action $a \in \pi_{sr}$.

The cost of an sr-plan is given by $\mathtt{c}(\pi_{sr}) = \sum_{a \in \pi_{sr}} \mathtt{c}(a)$. The sequential relaxation task asks for a minimum cost sr-plan. As every plan for Π is an sr-plan, it follows that any cost-optimal plan π^* has a cost greater or equal to any cost-optimal sr-plan π^*_{sr}, i.e., $\mathtt{c}(\pi^*_{sr}) \leq \mathtt{c}(\pi^*)$. Similarly to DFP, finding an sr-plan can be shown to be NP-hard by a reduction from set covering (Bylander 1994).

2.2 Related Work

Cost-optimal DFP is a well-studied problem in the planning community and has inspired several state-of-the-art heuristics (Betz and Helmert 2009). In particular, Bonet and Helmert (2010) show that a DFP can be reformulated as a hitting set problem with exponentially many subsets, each encoding a separate disjunctive landmark. This result has been extended to derive the necessary (Bonet and Castillo 2011) and the set-inclusion minimal (Haslum et al. 2012) set of disjunctive landmarks required to solve a DFP task. Moreover, Pommerening and Helmert (2012) build on the hitting set representation to design an incremental disjunctive landmark heuristic for DFP.

Alternatively, Imai and Fukunaga (2014) presented a mixed-integer linear programming formulation for the DFP with polynomially many constraints. This formulation is currently regarded as the state-of-the-art solution approach for DFP tasks. Moreover, its LP relaxation, when coupled with additional operator counting constraints (Pommerening et al. 2014), defines an admissible heuristic for classical planning problems that achieves competitive performance with respect to the state of the art (Imai and Fukunaga 2015).

Recent work has also shown an interest on decision diagrams (Bryant 1986) for planning tasks. Castro et al. (2018) use *relaxed multi-valued decision diagrams* to create a relaxed representation of the state-transition graph for classical planning. The approach relates to several well-known techniques, such as critical-path heuristics and abstractions. The authors report preliminary results that indicate the potential of the technique when it is used to extract valid plans.

The technique proposed in this paper is related to the work by Corrêa, Pommerening, and Francès (2018), who apply relaxed BDDs to approximate the state space of a DFP task. Our methodology, however, differs in three main ways. First, we focus on modeling the sequential relaxation using relaxed BDDs, while the previous work approximates the full DFP task. Second, we assign only one action per decision layer to reveal certain desired structure, while the previous representation allows multiple actions to be encoded in each layer. Lastly, Corrêa, Pommerening, and Francès (2018) use a top-down BDD construction, while we propose an iterative splitting procedure. While, the top-down construction is usually faster, the iterative splitting leverages the encoded information to create a stronger relaxation.

3 A BDD Encoding for the Sequential Relaxation

Our BDD representation exploits the fact that an action needs to be applied at most once in any DFP task (and, thus, in its sequential relaxation). We can therefore view the sequential relaxation task as a binary optimization problem that asks for a minimum-cost action set satisfying Definition 2.1, where each variable indicates whether or not an action is included in the set. The BDD representation, in turn, is an encoding of the set of solutions of such problem, i.e., it is a graphical structure such that a path in the graph represents an action set that is an sr-plan.

Formally, the BDD $\mathcal{B} = (\mathcal{N}, \mathcal{E})$ is a layered acyclic graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ the set of directed edges. The set of nodes is partitioned into $m+1$ layers ($m = |\mathcal{A}|$), $\mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_{m+1})$. The first and last node layer have a single node, known as the root, $\mathcal{N}_1 = \{\mathbf{r}\}$, and terminal node, $\mathcal{N}_{m+1} = \{\mathbf{t}\}$, respectively. Similarly, the set of edges $\mathcal{E}$ is partitioned into m layers, $\mathcal{E} = (\mathcal{E}_1, \ldots, \mathcal{E}_m)$, such that each edge $e = (u, v) \in \mathcal{E}_i$ ($i \in \{1, \ldots, m\}$) originates from a node $u \in \mathcal{N}_i$ and points to a node $v \in \mathcal{N}_{i+1}$.

With each layer $\mathcal{E}_i$ we associate an action $\theta(\mathcal{E}_i) \in \mathcal{A}$. Every edge $e \in \mathcal{E}_i$ has (i) a label $v(e) \in \{0, 1\}$ that represents if the action $\theta(\mathcal{E}_i)$ associated with its layer is selected to be included in an sr-plan, and (ii) a cost $\omega(e)$ derived from including (if $v(e) = 1$) or excluding (if $v(e) = 0$) the action to the sr-plan, i.e., $\omega(e) := v(e) \cdot \mathtt{c}(\theta(e))$. A node $u \in \mathcal{N}$ has at most two edges emanating from it, each with a distinct label. Thus, an $\mathbf{r} - \mathbf{t}$ path $\rho := (e_1, ..., e_m) \in \mathcal{B}$ has exactly one edge from each layer, and the labels associated with its edges correspond to the actions that will be included in and excluded from the sr-plan.

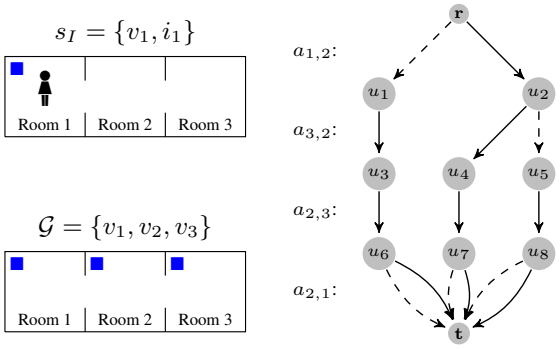
We force $\mathcal{B}$ to encode only sr-plans by keeping track of the set of propositions that are achieved and required in each path. Given an $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$, the set of propositions achieved and required by ρ is given, respectively, by:

$$\alpha^{\downarrow}(\rho) := s_I \cup \bigcup_{e \in \rho : v(e) = 1} \mathtt{add}(\theta(e)),$$

$$\eta^{\downarrow}(\rho) := \bigcup_{e \in \rho : v(e) = 1} \mathtt{pre}(\theta(e)).$$

A BDD $\mathcal{B}$ encodes only sr-plans if each $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$ satisfies the conditions in Definition 2.1, i.e., $\mathcal{G} \subseteq \alpha^{\downarrow}(\rho)$ and $\eta^{\downarrow}(\rho) \subseteq \alpha^{\downarrow}(\rho)$. We say that a BDD is *exact* if every path in $\mathcal{B}$ is an sr-plan and there is one path for each possible sr-plan in a DFP task Π. A shortest path in an exact BDD corresponds to a minimum-cost sr-plan.

Example 3.1. Consider the following problem Π of the visit-all domain: $\mathcal{P} = \{i_1, v_1, i_2, v_2, i_3, v_3\}$, where i_k and v_k represent that the agent *is currently at* and *has visited* room k, respectively; $\mathcal{A} = \{a_{1,2}, a_{2,1}, a_{2,3}, a_{3,2}\}$, where $a_{k,r}$ represents the movement from room k to r with $\mathtt{pre}(a_{k,r}) =$



(a) DFP task (b) Exact BDD

Figure 1: Example of a DFP task.

$\{i_k\}$ and $\texttt{add}(a_{k,r}) = \{i_r, v_r\}$. The initial and goal states are depicted in Figure 1a.

Figure 1b illustrates an exact BDD $\mathcal{B}$ for Π. The dashed edges represent zero-edges ($v(e) = 0$) and the solid edges one-edges ($v(e) = 1$). On the left are the actions associated with each layer. Notice that all paths $\rho \in \mathcal{B}$ correspond to sr-plans and there is exactly one path for each sr-plan.

The size of an exact BDD (i.e., the number of nodes) grows exponentially with the size of the planning task Π (i.e., the number of actions and propositions). We overcome this problem by constructing a *relaxed* BDD: a limited size BDD that over-approximates the set of sr-plans. In particular, we show in Theorem 6.1 that the shortest path of a relaxed BDD is an admissible heuristic for Π.

4 Relaxed BDD Construction Algorithm

Given a task Π, a relaxed BDD $\mathcal{B} = (\mathcal{N}, \mathcal{E})$ is a limited-size BDD that over-approximates the set of sr-plans for Π, i.e., there exists an $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$ for each sr-plan, but some paths are invalid sr-plans. We define the width of $\mathcal{B}$ as the maximum number of nodes in each layer, i.e., $w(\mathcal{B}) := \max\{|\mathcal{N}_i| : i = 1, ..., m+1\}$. We will limit the size of $\mathcal{B}$ by bounding its width, $w(\mathcal{B}) \leq \mathcal{W}$, with $\mathcal{W} \geq 1$.

Algorithm 1 presents our relaxed BDD construction scheme. The algorithm starts by constructing a width-one BDD ($\mathcal{W} = 1$). The InitialWidthOneBDD procedure receives a sequence of pair-wise distinct action labels $\Lambda = (\lambda_1, ..., \lambda_m)$ and assigns each action to a layer such that $a_{\lambda_i} = \theta(\mathcal{E}_i)$, for all $i \in \{1, ..., m\}$. It then constructs a width-one BDD by creating one node $u \in \mathcal{N}_i$ in each node layer $i \in \{1, ..., m+1\}$ and two edges $e, e' \in \mathcal{E}_i$ in each edge layer $i \in \{1, ..., m\}$ with different labels (i.e., $v(e) \neq v(e')$).

The construction procedure increases the size of the BDD and removes invalid sr-plans until the BDD cannot be further updated. The TopDownProcedure iterates over each node layer starting from $\mathcal{N}_1$. It updates the information stored in every node $u \in \mathcal{N}$ considering all partial $\mathbf{r} - u$ paths (UpdateNodesTopDown), eliminates edges that are associated to invalid sr-plans (FilterEdges), and splits every node u until the maximum width is reached (SplitNodes). In contrast, the BottomUpProcedure

Algorithm 1 Relaxed BDD Construction

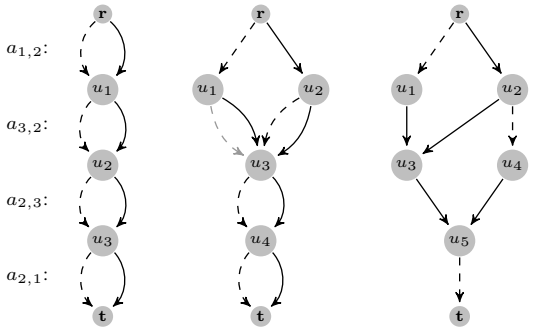
```
1:  procedure ConstructBDD(Π, 𝒲, Λ)
2:      ℬ := InitialWidthOneBDD(Λ)
3:      stop := False
4:      while !stop do
5:          stop := True
6:          TopDownProcedure(ℬ, 𝒲, Π, stop)
7:          BottomUpProcedure(ℬ, 𝒲, Π, stop)
8:      h_ℬ := GetShortestPath(ℬ)
9:      return h_ℬ

10: procedure TopDownProcedure(ℬ, 𝒲, Π, stop)
11:     for 1 ≤ i ≤ m do
12:         UpdateNodesTopDown(𝒩_i)
13:         FilterEdges(ℰ_i, stop)
14:         SplitNodes(𝒩_i, 𝒲, stop)

15: procedure BottomUpProcedure(ℬ, Π, 𝒲, stop)
16:     i := |𝒜| + 1
17:     while i ≥ 0 do
18:         UpdateNodesBottomUp(𝒩_i)
19:         FilterEdges(ℰ_{i−1}, stop)
20:         i := i − 1
```

iterates over each layer starting with $\mathcal{N}_{m+1}$. It updates the information stored in every node $u \in \mathcal{N}$ considering all partial $u - \mathbf{t}$ paths (UpdateNodesBottomUp) and removes edges that are associated to invalid sr-plans (FilterEdges).

Example 4.1. Consider our running example. Figure 2 illustrates some of the steps of Algorithm 1. Figure 2a depicts the width-one BDD created by the InitialWidthOneBDD procedure. Figure 2b illustrates the SplitNodes procedure over $\mathcal{N}_2$ and the FilterEdges procedure for $\mathcal{E}_2$ by a gray edge (eliminated by Rule 2, Section 4.2). Lastly, Figure 2c shows the relaxed BDD obtained at the end of the construction procedure.



(a) Width-one BDD (b) One-split (c) Resulting BDD

Figure 2: BDD construction with $\mathcal{W} = 2$.

4.1 Relaxed Node Representation

With the purpose of identifying invalid sr-plans and deciding how to split a node, in each node $u \in \mathcal{N}$ we store information about the achieved and required propositions by all

paths passing through u. The information is aggregated both for the $\mathbf{r}-u$ and the $u-\mathbf{t}$ partial paths.

Consider $\delta^{in}(u)$ and $\delta^{out}(u)$ as the set of incoming and outgoing edges of a node $u \in \mathcal{N}$, respectively. For each edge $e=(u,v) \in \mathcal{E}$, $\sigma(e)=u$ and $\tau(e)=v$ represent its source and target node, respectively. Also, consider that all zero-edges ($v(e)=0$) have $\texttt{add}(\theta(e)) = \texttt{pre}(\theta(e)) = \emptyset$.

For the top-down information, each $u \in \mathcal{N}$ stores the propositions that are achieved by *all* $\mathbf{r}-u$ paths, $\alpha_A^{\downarrow}(u)$, and the propositions achieved by *at least one* $\mathbf{r}-u$ path, $\alpha_S^{\downarrow}(u)$. Starting with node $\mathbf{r}$, the UPDATENODESTOPDOWN procedure updates these sets for each node in $\mathcal{B}$. The procedure assigns $\alpha_A^{\downarrow}(\mathbf{r}) := \alpha_S^{\downarrow}(\mathbf{r}) := s_I$ and updates each $u \in \mathcal{N}$ as

$$\alpha_A^{\downarrow}(u) := \bigcap_{e \in \delta^{in}(u)} \left(\alpha_A^{\downarrow}(\sigma(e)) \cup \texttt{add}(\theta(e))\right),$$

$$\alpha_S^{\downarrow}(u) := \bigcup_{e \in \delta^{in}(u)} \left(\alpha_S^{\downarrow}(\sigma(e)) \cup \texttt{add}(\theta(e))\right).$$

Similarly, each node $u \in \mathcal{N}$ stores the set of propositions that are required by *all* and *at least one* $\mathbf{r}-u$ paths, $\eta_A^{\downarrow}(u)$ and $\eta_S^{\downarrow}(u)$, respectively. At the root node $\eta_A^{\downarrow}(\mathbf{r}) := \eta_S^{\downarrow}(\mathbf{r}) := \mathcal{G}$, and for any other node $u \in \mathcal{N}$ we have that

$$\eta_A^{\downarrow}(u) := \bigcap_{e \in \delta^{in}(u)} \left(\eta_A^{\downarrow}(\sigma(e)) \cup \texttt{pre}(\theta(e))\right),$$

$$\eta_S^{\downarrow}(u) := \bigcup_{e \in \delta^{in}(u)} \left(\eta_S^{\downarrow}(\sigma(e)) \cup \texttt{pre}(\theta(e))\right).$$

For the bottom-up information of a node $u \in \mathcal{N}$, sets $\alpha_A^{\uparrow}(u)$ and $\eta_A^{\uparrow}(u)$ correspond to the propositions achieved and required by all $u-\mathbf{t}$ paths, respectively. Similarly, sets $\alpha_S^{\uparrow}(u)$ and $\eta_S^{\uparrow}(u)$ represent the propositions achieved and required by at least one $u-\mathbf{t}$ path, respectively. Starting at $\mathbf{t}$ with $\alpha_A^{\uparrow}(\mathbf{t}) := \alpha_S^{\uparrow}(\mathbf{t}) := \eta_A^{\uparrow}(\mathbf{t}) := \eta_S^{\uparrow}(\mathbf{t}) := \emptyset$, the UPDATENODESBOTTOMUP procedure updates these sets as

$$\alpha_A^{\uparrow}(u) := \bigcap_{e \in \delta^{out}(u)} \left(\alpha_A^{\uparrow}(\tau(e)) \cup \texttt{add}(\theta(e))\right),$$

$$\alpha_S^{\uparrow}(u) := \bigcup_{e \in \delta^{out}(u)} \left(\alpha_S^{\uparrow}(\tau(e)) \cup \texttt{add}(\theta(e))\right),$$

$$\eta_A^{\uparrow}(u) := \bigcap_{e \in \delta^{out}(u)} \left(\eta_A^{\uparrow}(\tau(e)) \cup \texttt{pre}(\theta(e))\right),$$

$$\eta_S^{\uparrow}(u) := \bigcup_{e \in \delta^{out}(u)} \left(\eta_S^{\uparrow}(\tau(e)) \cup \texttt{pre}(\theta(e))\right).$$

In addition, each $u \in \mathcal{N}$ maintains the cost of the shortest $\mathbf{r}-u$ and $u-\mathbf{t}$ path, $\omega^{\downarrow}(u)$ and $\omega^{\uparrow}(u)$, respectively. Starting with $\omega^{\downarrow}(\mathbf{r}) := \omega^{\uparrow}(\mathbf{t}) := 0$, the cost of $u \in \mathcal{N}$ is given by

$$\omega^{\downarrow}(u) := \min_{e \in \delta^{in}(u)} \left\{\omega^{\downarrow}(\sigma(e)) + \omega(e)\right\},$$

$$\omega^{\uparrow}(u) := \min_{e \in \delta^{out}(u)} \left\{\omega^{\uparrow}(\tau(e)) + \omega(e)\right\}.$$

The shortest-path information is used both for the BDD heuristic computation (Section 6) and to identify and eliminate sub-optimal sr-plans (Section 4.2).

Example 4.2. Consider node u_3 in Figure 2c for our running example. For the top-down information we have $\alpha_A^{\downarrow}(u_3) = \alpha_S^{\downarrow}(u_3) = \{i_1, i_2, v_1, v_2\}$, $\eta_A^{\downarrow}(u_3) = \{v_1, v_2, v_3, i_3\}$ and $\eta_S^{\downarrow}(u_3) = \eta_A^{\downarrow}(u_3) \cup \{i_1\}$. For the bottom up we have $\alpha_A^{\uparrow}(u_3) = \alpha_S^{\uparrow}(u_3) = \{i_3, v_3\}$ and $\eta_A^{\uparrow}(u_3) = \eta_S^{\uparrow}(u_3) = \{i_2\}$.

4.2 Filtering Rules

The FILTEREDGES procedure removes paths in $\mathcal{B}$ that form invalid sr-plans. To do so, we develop a set of rules to identify if at least one path passing through an edge corresponds to a sr-plan. If an edge $e \in \mathcal{E}$ violates a rule, then all paths passing through e are invalid sr-plans, and edge e can be removed.

Consider an edge $e=(u,v) \in \mathcal{E}$ with $\theta(e)=a$. As we will show in Proposition 4.1, the following two rules are necessary conditions for any sr-plan path.

Rule 1. Assume $v(e)=1$. Every precondition of a has to be added by some action in at least one path passing through e.

$$\texttt{pre}(a) \subseteq \alpha_S^{\downarrow}(u) \cup \alpha_S^{\uparrow}(v).$$

Rule 2. Each proposition required by all paths traversing e needs to be added by at least one action in some path.

$$\eta_A^{\downarrow}(u) \cup \eta_A^{\uparrow}(v) \subseteq \alpha_S^{\downarrow}(u) \cup \alpha_S^{\uparrow}(v), \quad \text{if } v(e)=0,$$

$$\eta_A^{\downarrow}(u) \cup \eta_A^{\uparrow}(v) \subseteq \alpha_S^{\downarrow}(u) \cup \texttt{add}(a) \cup \alpha_S^{\uparrow}(v), \quad \text{if } v(e)=1.$$

Proposition 4.1. Consider a relaxed BDD $\mathcal{B}=(\mathcal{N},\mathcal{E})$. Rules 1 and 2 are necessary conditions for any $\mathbf{r}-\mathbf{t}$ path $\rho \in \mathcal{B}$ to be an sr-plan.

Proof. Consider an sr-plan path $\rho=(e_1,\ldots,e_m) \in \mathcal{B}$ and edge $e=(u,v) \in \rho$. From Section 3, we know that $\bigcup_{e' \in \rho \setminus \{e\}} \texttt{add}(\theta(e')) \subseteq \alpha_S^{\downarrow}(u) \cup \alpha_S^{\uparrow}(v)$, and $\eta_A^{\downarrow}(u) \cup \eta_A^{\uparrow}(v) \subseteq \bigcup_{e' \in \rho \setminus \{e\}} \texttt{pre}(\theta(e')) \cup \mathcal{G}$. Since $\texttt{pre}(a) \cap \texttt{add}(a) = \emptyset$ for all $a \in \mathcal{A}$, Rules 1 and 2 are satisfied by any $e \in \rho$. □

Since we are interested in minimum-cost plans, we develop two additional filtering rules to identify suboptimal plans. Consider an edge $e=(u,v) \in \mathcal{E}$ with $\theta(e)=a$.

Rule 3. Assume $v(e)=1$. Action a adds at least one $p \notin s_I$ that is required by some path traversing e.

$$(\texttt{add}(a) \setminus s_I) \cap \left(\eta_S^{\downarrow}(u) \cup \eta_S^{\uparrow}(v)\right) \neq \emptyset.$$

Rule 4. Consider a plan $\pi' \in \Pi$. The minimum-cost path traversing e has a cost less than or equal to $\texttt{c}(\pi')$.

$$\omega^{\downarrow}(u) + \omega(e) + \omega^{\uparrow}(v) \leq \texttt{c}(\pi').$$

Notice that all cost-optimal plans satisfy these rules. Rule 3 avoids unnecessary actions and Rule 4 removes sr-plans with higher cost than the best plan found so far.

The FILTEREDGES procedure (Algorithm 2) iterates over all edges in a layer (line 2). It removes all edges that violate any of our four filtering rules (lines 3-4). The procedure also updates variable STOP if an edge was removed (line 4), i.e., the BDD has changed.

Algorithm 2 Filtering Edges Procedure

```
1: procedure FILTEREDGES(E_i, STOP )
2:     for e ∈ E_i do
3:         if e violates any of Rule 1 to Rule 4 then
4:             Eliminate e, STOP := FALSE
```

4.3 The Splitting Nodes Procedure

The SPLITNODES procedure aims to split nodes such that, if $\mathcal{W} = \infty$, it guarantees that the resulting BDD is exact (i.e., all paths are sr-plans). Our approach takes advantage of DFP characteristics to create relaxed BDDs with robust worst cases on the maximum width needed per layer.

We start by defining the exact information of a node. Consider a node $u \in \mathcal{N}$ and a proposition p. We say that p is α-exact in u if all $\mathbf{r}-u$ paths add p or none do, i.e., $p \in \alpha_A^{\downarrow}(u)$ or $p \notin \alpha_S^{\downarrow}(u)$, respectively. Similarly, we say that p is η-exact in u if either all $\mathbf{r} - u$ paths require p or none do, $p \in \eta_A^{\downarrow}(u)$ or $p \notin \eta_S^{\downarrow}(u)$, respectively.

Algorithm 3 illustrates the SPLITNODES procedure. The algorithm receives a node layer and the width limit, $\mathcal{W}$. The procedure iterates over a priority queue of propositions $Q = \mathcal{P}_{\neg s_I}$ and splits nodes such that for each $p \in Q$, all nodes are α-exact and η-exact or the width limit is reached.

Algorithm 4 shows how to split any node $u \in \mathcal{N}$ such that for each $p \in \mathcal{P}$, p is α-exact or η-exact in u, respectively. The procedure iterates over the incoming edges of u and redirects the edges to a new node u' accordingly.

Proposition 4.2. Consider a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{E})$ such that for each $p \in \mathcal{P}$ and node $u \in \mathcal{N}$, p is α-exact in u and p is η-exact in u when $p \notin \alpha_A^{\downarrow}(u)$. Then, Rule 1 and Rule 2 are sufficient to remove all invalid sr-plan paths in $\mathcal{B}$.

Proof. Consider a path $\rho \in \mathcal{B}$ and $p \in \mathcal{P}$ such that either there exists an action $a \in \rho$ with $p \in \mathtt{pre}(a)$ or $p \in \mathcal{G}$ but for all $a' \in \rho$, $p \notin \mathtt{add}(a')$. Take the last edge $e \in \rho$, i.e., $\sigma(e) = u \in \mathcal{N}_m$ and $\tau(e) = \mathbf{t}$. Since ρ is an invalid sr-plan, $p \notin \alpha_A^{\downarrow}(u)$ and $p \in \eta_A^{\downarrow}(u) \cup \mathtt{pre}(\theta(e))$. Moreover, $p \notin \alpha_S^{\downarrow}(u)$ since p is α-exact in u. Since $\alpha_S^{\uparrow}(\mathbf{t}) = \emptyset$ (Section 4.1), either Rule 1 or Rule 2 will eliminate edge e and, therefore, remove ρ from $\mathcal{B}$. □

Proposition 4.2 implies that for each proposition $p \in \mathcal{P}$ we need at most three nodes in each layer $\mathcal{N}_i$, i.e., a node $u \in \mathcal{N}_i$ where $p \in \alpha_A^{\downarrow}(u)$, a node $u' \in \mathcal{N}_i$ where $p \notin \alpha_A^{\downarrow}(u')$ and $p \in \eta_A^{\downarrow}(u')$, and a node $u'' \in \mathcal{N}_i$ where $p \notin \alpha_A^{\downarrow}(u'')$ and $p \notin \eta_A^{\downarrow}(u'')$. Since for all $p \in s_I$ and $u \in \mathcal{N}$, $p \in \alpha_A^{\downarrow}(u)$ (Section 4.1), there is no need to split nodes with respect to propositions in the initial state. Similarly, for all $p \in \mathcal{G}$ and $u \in \mathcal{N}$, $p \in \eta_A^{\downarrow}(u)$ (Section 4.1), so each goal proposition needs at most two nodes in each layer. Then, the maximum width needed to construct an exact BDD is $O(3^{|\mathcal{P}_{\neg s_I, \neg \mathcal{G}}|} \cdot 2^{|\mathcal{G}_{\neg s_I}|})$, where $\mathcal{G}_{\neg s_I} = \mathcal{G} \setminus s_I$ (i.e., all goals omitted in the initial state) and $\mathcal{P}_{\neg s_I, \neg \mathcal{G}} = (\mathcal{P} \setminus \mathcal{G}) \setminus s_I$ (i.e., all non-goal proposition omitted in the initial state).

Algorithm 3 Split Nodes Procedures

```
1: procedure SPLITNODES(N_i, W, STOP)
2:     Q = P_{¬s_I}, priority queue of propositions
3:     while Q.notEmpty() and |N_i| < W do
4:         p = Q.pop()
5:         if i > γ(p) + 1 then continue
6:         for u ∈ N_i do
7:             if p ∈ α_S^↓(u), p ∉ α_A^↓(u) then
8:                 SPLITNODEACHIEVED(u, p)
9:                 STOP := FALSE
10:            if |N_i| = W then return
11:        for u ∈ N_i do
12:            if p ∈ η_S^↓(u), p ∉ η_A^↓(u), p ∉ α_A^↓(u) then
13:                SPLITNODENEEDED(u, p)
14:                STOP := FALSE
15:            if |N_i| = W then return
```

Algorithm 4 Split Single Node Procedure

```
1: procedure SPLITNODEACHIEVED(u, p )
2:     Create a new node u' and update N_i = N_i ∪ {u'},
3:     for e ∈ δ^in(u) do
4:         if p ∈ α_A^↓(σ(e)) or p ∈ add(θ(e)) then
5:             Redirect edge e: e ∉ δ^in(u), e ∈ δ^in(u')
6:     Duplicate edges from u to u' if δ^in(u') ≠ ∅

7: procedure SPLITNODENEEDED(u, p )
8:     Create a new node u' and update N_i = N_i ∪ {u'},
9:     for e ∈ δ^in(u) do
10:        if p ∈ η_A^↓(σ(e)) or p ∈ pre(θ(e)) then
11:            Redirect edge e: e ∉ δ^in(u), e ∈ δ^in(u')
12:    Duplicate edges from u to u' if δ^in(u') ≠ ∅
```

Even though the proposed splitting approach is valid, we prove that it is possible to create an exact BDD where not all nodes are α-exact or η-exact. Given a BDD $\mathcal{B}$ and a proposition $p \in \mathcal{P}$, we define the *last layer* of p, $\gamma(p)$, as the maximum layer index at which an action either adds or requires p, i.e., for $i = \gamma(p)$, $p \in \mathtt{add}(\theta(\mathcal{E}_i)) \cup \mathtt{pre}(\theta(\mathcal{E}_i))$ and for all $j > \gamma(p)$, $p \notin \mathtt{add}(\theta(\mathcal{E}_j)) \cup \mathtt{pre}(\theta(\mathcal{E}_j))$.

Proposition 4.3. Consider $\mathcal{B} = (\mathcal{N}, \mathcal{E})$ such that for each $p \in \mathcal{P}$ and node $u \in \mathcal{N}_i$, with $i \leq \gamma(p) + 1$, p is α-exact in u and p is η-exact in u when $p \notin \alpha_A^{\downarrow}(u)$. Then, Rule 1 and 2 are sufficient to remove all invalid sr-plan paths in $\mathcal{B}$.

Proof. Consider a path $\rho \in \mathcal{B}$ with a proposition p such that $p \in \mathcal{G}$ or for some action $a \in \rho$, $p \in pre(a)$ but p is not added by any action in ρ. Now take edge $e = (u, v) \in \rho$ in the last layer of p, i.e., $u \in \mathcal{N}_{\gamma(p)}$ and $v \in \mathcal{N}_{\gamma(p)+1}$. By the definition of last layer $p \notin \alpha_S^{\uparrow}(v)$ and $p \notin \eta_S^{\uparrow}(v)$. By hypothesis over ρ, $p \notin \alpha_A^{\downarrow}(u)$ (and, $p \notin \alpha_S^{\downarrow}(u)$) and $p \in \eta_A^{\downarrow}(u) \cup \mathtt{pre}(\theta(e))$. Then, either Rule 1 or Rule 2 will remove edge e and, hence, the invalid sr-plan path ρ. □

Notice that Algorithm 3 uses Proposition 4.3 to avoid splitting nodes with respect to $p \in Q$ when the current layer is greater than $\gamma(p)$ (line 5), and avoids splitting nodes $u \in \mathcal{N}$ if $p \in \alpha_A^{\downarrow}(u)$ (line 12).

5 On Width Bounds and Action Ordering

Given the BDD construction procedure in Section 4, we present an upper bound for the maximum width needed in each layer of an exact BDD. We show how these bounds depend on the action-layer assignment and develop a simple heuristic procedure to create good action-layer orderings.

For a given $p \in \mathcal{P}_{\neg s_I}$, consider the first layer where p is either added or required, i.e., $\phi(p) = i$ if $p \in \texttt{add}(\theta(\mathcal{E}_i)) \cup \texttt{pre}(\theta(\mathcal{E}_i))$ and for all $j < \phi(p)$, $p \notin \texttt{add}(\theta(\mathcal{E}_j)) \cup \texttt{pre}(\theta(\mathcal{E}_j))$. Now consider $\psi(i)$ as the set of propositions that need to be considered for splitting in layer $\mathcal{N}_i$, i.e., $\psi(i) = \{p \in \mathcal{P}_{\neg s_I} : \phi(p) < i \leq \gamma(p)\}$. In particular, consider $\psi_{\mathcal{G}}(i) = \psi(i) \cap \mathcal{G}$ as the set of goal propositions that need splitting in layer $\mathcal{N}_i$, and $\psi_{\mathcal{P}}(i) = \psi(i) \setminus \mathcal{G}$ as the set of non-goal proposition that need splitting in layer $\mathcal{N}_i$.

Corollary 5.1. Consider a DFP task Π and an exact BDD $\mathcal{B}$ constructed using Algorithm 1. The maximum width of layer $\mathcal{N}_i$ is $O(2^{|\psi_{\mathcal{G}}(i)|} \cdot 3^{|\psi_{\mathcal{P}}(i)|})$. Then, an upper bound on the maximum width for $\mathcal{B}$ is given by $O(\max_{i\in\{1,...,m\}}\{2^{|\psi_{\mathcal{G}}(i)|} \cdot 3^{|\psi_{\mathcal{P}}(i)|}\})$.

Proof. This follows directly from Proposition 4.3. □

Notice that $\psi(i)$ depends on the action ordering Λ used to assign actions to layers (Algorithm 1, line 2). In particular, we would like to minimize the number of propositions that need to be split in every layer, i.e., find a Λ such that $\max_{i\in\{1,...,m\}}\{2^{|\psi_{\mathcal{G}}(i)|} \cdot 3^{|\psi_{\mathcal{P}}(i)|}\}$ is minimized. This NP-hard problem has been studied for knapsack constraints (Behle 2008) and for the set covering and independent set problems (Bergman, van Hoeve, and Hooker 2011; Bergman et al. 2012).

Action Ordering. We develop a simple action ordering heuristic that takes advantage of the following proposition.

Proposition 5.1. Consider a DFP task Π and a relaxed BDD $\mathcal{B}$ constructed using Algorithm 1 with action ordering Λ. Assume that for a given $p \in \mathcal{P}_{\neg s_I, \neg \mathcal{G}}$ all actions that add p are ordered before all actions that require p in Λ. Then, it is sufficient to have $\mathcal{W} = 2$ to guarantee that all paths $\rho \in \mathcal{B}$ that require p have an action that adds p.

Proof. Since all actions that add p are ordered first, we need two nodes in a layer to ensure that p is α-exact in each node. Consider the first edge layer $\mathcal{E}_i$ such that $a = \theta(\mathcal{E}_i)$ requires p. Take a node $u \in \mathcal{N}_j$ ($j > i$). If $p \in \alpha_A^{\downarrow}(u)$, u does not need to be η-exact (Proposition 4.3). If $p \notin \alpha_A^{\downarrow}(u)$, Rule 1 and 2 eliminate edges that require p, so no split is needed. □

Our action ordering Λ starts by creating a priority queue of propositions $Q = \mathcal{P}_{\neg s_I}$. This priority queue, also used in Algorithm 3, is such that all goals and propositional landmarks are ordered first. All other propositions are sorted in decreasing order according to the number of actions that require them. Then, the action ordering is as follows: for each proposition $p \in Q$ we insert actions $a \in \mathcal{A} \setminus \Lambda$ that add p into Λ and then actions $a \in \mathcal{A} \setminus \Lambda$ that require p.

6 Relaxed BDD Heuristic

Given a DFP task Π and a state s, we can construct a relaxed BDD $\mathcal{B}$ for s using Algorithm 1 and updating the initial state $s_I := s$. Then, the relaxed BDD heuristic $h_{\mathcal{B}}(s)$ corresponds to the shortest path in $\mathcal{B}$, i.e., $h_{\mathcal{B}}(s) := \omega^{\downarrow}(\mathbf{t})$.

Theorem 6.1. Consider a delete-free task Π, a state s and a relaxed BDD $\mathcal{B}$ for s with $\mathcal{W} \geq 1$ constructed using Algorithm 1. Then, $h_{\mathcal{B}}(s)$ is admissible.

Proof. Propositions 4.1 guarantees that no $\texttt{sr}$-plan is eliminated, while Rule 3 and 4 guarantee the presence of at least one cost-optimal plan. Then, $h_{\mathcal{B}}(s) \leq \texttt{c}(\pi^*_{sr}) \leq h^+(s)$, where π^*_{sr} is the cost-optimal $\texttt{sr}$-plan from s and h^+ the perfect delete-free heuristic. □

In our implementation, we construct a relaxed BDD $\mathcal{B}$ for s_I and update $\mathcal{B}$ during search. Given a state s and the sequence of actions to achieve s from s_I, $\pi_{s_I,s} = (a_1, .., a_k)$, we update $\mathcal{B}$ by removing all edges $e \in \mathcal{E}$ with $\theta(e) \in \pi_{s_I,s}$ and $v(e) = 0$. We then iteratively apply the top-down and bottom-up procedures over $\mathcal{B}$ (lines 4 to 8, Algorithm 1) and compute the heuristic as:

$$h_{\mathcal{B}}(s) := \omega^{\downarrow}(\mathbf{t}) - \texttt{c}(\pi_{s_I,s}). \quad (1)$$

Notice that $h_{\mathcal{B}}$ is still admissible (Theorem 6.1) and, in this case, consistent (Theorem 6.2). However, the consistency of $h_{\mathcal{B}}$ depends on the BDD construction procedure and whether the action and proposition ordering changes.

Theorem 6.2. Consider a delete-free task Π, a state s and a relaxed BDD $\mathcal{B}$ with $\mathcal{W} \geq 1$ constructed using Algorithm 1. Then, $h_{\mathcal{B}}(s)$ given by (1) is consistent.

Proof. Consider a state s, an applicable action a and its successor state $s' = \texttt{succ}(s, a)$. Given the relaxed BDD $\mathcal{B}_{s_I}$ for s_I, let $\mathcal{B}_s$ and $\mathcal{B}_{s'}$ be the updated BDDs for state s and s', respectively. Since each BDD is updated from $\mathcal{B}_{s_I}$ without changing the action and proposition order, every path $\rho \in \mathcal{B}_{s'}$ is also in $\mathcal{B}_s$. Then, we have $\omega^{\downarrow}(\mathbf{t}_s) \leq \omega^{\downarrow}(\mathbf{t}_{s'})$.

We know that $h_{\mathcal{B}}(s) = \omega^{\downarrow}(\mathbf{t}_s) - \texttt{c}(\pi_{s_I,s})$ and $h_{\mathcal{B}}(s') = \omega^{\downarrow}(\mathbf{t}_{s'}) - \texttt{c}(\pi_{s_I,s}) + \texttt{c}(a)$. Then, $h_{\mathcal{B}}(s) - \texttt{c}(a) - h_{\mathcal{B}}(s') = \omega^{\downarrow}(\mathbf{t}_s) - \omega^{\downarrow}(\mathbf{t}_{s'}) \leq 0$, and so $h_{\mathcal{B}}(s) \leq \texttt{c}(a) + h_{\mathcal{B}}(s')$. □

7 Exploiting the Relaxed BDD Structure

Given a relaxed BDD $\mathcal{B}$ for a DFP task Π, we explain how to identify redundant and landmark actions. Consider an edge layer $\mathcal{E}_i$ ($i \in \{1, ..., m\}$) such that all edges $e \in \mathcal{E}_i$ have label $v(e) = 0$. Then, no minimum-cost plan uses action $a = \theta(\mathcal{E}_i)$, so a is a redundant action that can be removed from $\mathcal{A}$. Similarly, consider an edge layer $\mathcal{E}_i$ ($i \in \{1, ..., m\}$) such that all edges $e \in \mathcal{E}_i$ have label $v(e) = 1$. Then, all minimum-cost plans have action $a = \theta(\mathcal{E}_i)$, i.e., a is an action landmark for any cost-optimal plan.

BDD for Plan Extraction. Algorithm 5 shows how we can use relaxed BDDs to extract plans for a DFP task Π. Given a state s and its relaxed BDD $\mathcal{B}_s$, the procedure starts with an empty plan π (line 2) and adds actions to π until all goals are satisfied. In each iteration, the procedure updates

$\mathcal{B}_s$ by keeping only the paths that have all the actions in π (line 4). Then, the procedure looks for all applicable actions in state s that add at least one new proposition, stores them on a list L (line 5), and uses $\mathcal{B}_s$ to select the most *promising* action (line 6). Specifically, given a list of actions L, we greedily look for the action that has a path in $\mathcal{B}_s$ with the minimum cost, i.e., $a^* = \text{argmin}_{a \in L}\{\min\{\omega^{\downarrow}(u) + \omega(e) + \omega^{\uparrow}(v) : e = (u, v) \in \mathcal{E}, \theta(e) = a, v(e) = 1\}\}$. Lastly, we insert a^* to π and update state s (line 7).

Algorithm 5 Plan Extraction Procedure

```
1: procedure PLANEXTRACTION(B_s, s)
2:     π := ∅
3:     while G ⊈ s do
4:         UPDATEPATHSBDD(π, B_s)
5:         L := FINDAPPLICABLEACTIONS(s)
6:         a* := SELECTACTIONBDD(L, B_s)
7:         π.insert(a*), s := s ∪ add(a*)
8:     return π
```

8 Empirical Analysis

We present an empirical analysis of the relaxed BDD heuristic with the main objective to compare how the BDD-based heuristic compares to the LP relaxation for a DFP task. We also show the strength of BDDs in action pruning.

8.1 Implementation Details

We implement a branch-and-bound best-first-search algorithm (Land and Doig 1960). Similar to Pommerening and Helmert (2012), our search algorithm branches over each action, i.e., either the action is considered in the cost-optimal plan or not. However, we use a best-first instead of a depth-first strategy. We also avoid branching on zero-cost actions as their inclusion does not affect a plan cost. Notice that we can update our BDD using these two branching decisions by removing all edges $e \in \mathcal{E}_i$ ($\theta(\mathcal{E}_i) = a$) with $v(e) = 0$ and $v(e) = 1$, respectively.

Our search procedure starts at s_I and branches according to its applicable actions. In particular, for a state s in the search and an applicable action a, we create two successors, one where a is applied and one where a is never applied. Hence, all states in the search are reachable states in Π.

For the branch-and-bound, the global lower bound LB is given by the cost to reach the last state expanded plus its heuristic value. We use Algorithm 5 to extract plans in every state of the search and we use the minimum-cost plan seen so far as our global upper bound UB. The search ends when either $LB = UB$ or the next state to expand is a goal state.

We also implement a subset of the improvements and preprocessing techniques used by Imai and Fukunaga (2014; 2015). In particular, we use the AND-OR graph landmark extraction (Keyder, Richter, and Helmert 2010) and relevance and dominance analysis to eliminate actions and propositions. We treat propositional landmarks as goal propositions during the BDD construction procedure.

We implement the mixed-integer linear programming (MIP) model for DFP (Imai and Fukunaga 2014; 2015) and its LP relaxation, and compare both against a BDD-based heuristic with $\mathcal{W} \in \{2, 4, 8\}$. The BDD and LP are used as heuristics within our search algorithm, while the MIP is solved once to produce the cost-optimal delete-free plan.

We test all approaches over domains from the last three IPC competitions (Table 1). We restrict ourselves to domains with no negative preconditions and no conditional effects.[1] Our experiments consider a 30 minute time limit and a 2GB memory limit. We use Gurobi 8.0 to solve the LP and MIP models. Everything is coded in `C++`.

To make the comparison as fair as possible, we calculate an initial upper bound using the FF heuristic (Hoffmann and Nebel 2001) and implement a plan extraction procedure for the LP approach. Whenever the LP returns an integer solution, we check if it is a plan. If so, we update the global upper bound if the cost is lower than the current incumbent.

8.2 Relaxed BDD Heuristic Quality

We start our analysis by measuring the heuristic quality of the BDD relaxation in comparison to the LP relaxation. For each instance where the optimal solution h^+ is known, we compute the optimality gap at the initial state as $gap = (h^+ - h(s_I))/h^+$. Table 1 shows the average optimality gap for each domain and technique.

Table 1: Heuristic Quality at s_I.

	Average Optimality Gap			
domain	LP	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$
barman-opt11	**0.73**	0.76	0.76	0.76
barman-opt14	**0.13**	0.24	0.24	0.19
childsnack-opt14	**0.22**	0.33	0.30	0.26
elevators-opt11	**0.48**	0.61	0.61	0.61
floortile-opt11	**0.05**	0.27	0.27	0.27
floortile-opt14	**0.07**	0.29	0.29	0.29
ged-opt14	**0.88**	1.00	1.00	1.00
nomystery-opt11	**0.00**	0.02	0.02	0.01
openstacks-opt11	**0.00**	1.00	1.00	1.00
parking-opt11	**0.13**	0.18	0.18	0.18
parking-opt14	**0.11**	0.15	0.15	0.15
pegsol-opt11	**0.43**	0.99	0.99	0.95
scanalyzer-opt11	**0.04**	**0.04**	**0.04**	**0.04**
sokoban-opt11	**0.05**	0.21	0.19	0.18
transport-opt11	**0.94**	0.99	0.99	0.99
transport-opt14	**0.81**	0.97	0.97	0.97
visitall-opt11	**0.03**	0.06	0.05	0.05
visitall-opt14	**0.02**	0.04	0.04	0.04
woodworking-opt11	**0.12**	0.24	0.20	0.17

We can see that our small-width BDD relaxations have a gap close to LP heuristic in most domains, even when the LP encodes the sequence component of a DFP task. The competitive BDD gap is mostly due to our construction algorithm and action ordering. In fact, the BDD computes close-to-optimal heuristics in *no-mystery*, *scanalyzer* and *visit-all*.

[1] No domains from IPC2018 satisfies these requirements.

Table 2: Average performance over IPC delete-free planning domains.

	Coverage						Average Time (sec)					Average States Evaluated			
domain	#	MIP	LP	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	MIP	LP	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	LP	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$
barman-opt11	20	8	0	0	0	0	-	-	-	-	-	-	-	-	-
barman-opt14	14	14	9	**14**	**14**	**14**	1.1	988.9	**4.6**	9.5	15.5	29,351.8	17,669.3	16,742.9	**14,745.1**
childsnack-opt14	20	20	6	18	18	**20**	0.5	293.1	6.7	**2.8**	2.9	12,019.7	21,310.0	9,371.0	**9,102.8**
elevators-opt11	20	18	17	**20**	**20**	19	425.7	240.5	**27.3**	50.1	100.4	116,954.5	70,181.6	68,650.4	**67,184.5**
floortile-opt11	20	20	**14**	4	4	4	0.4	**19.7**	86.7	87.1	86.6	**6,666.5**	173,597.0	116,613.3	67,609.5
floortile-opt14	20	20	**16**	1	1	1	0.3	**12.7**	89.5	128.2	179.1	**3,738.0**	210,121.0	178,847.0	126,276.0
ged-opt14	20	20	**20**	**20**	**20**	**20**	391.1	39.5	**1.4**	2.1	3.7	25,773.8	**2,110.9**	**2,110.9**	**2,110.9**
nomystery-opt11	20	20	**18**	16	**18**	**18**	49.8	39.2	**0.8**	0.9	1.0	104.4	**1.0**	**1.0**	**1.0**
openstacks-opt11	20	20	**20**	**20**	**20**	**20**	0.9	0.8	**0.1**	**0.1**	**0.1**	**1.0**	2.0	2.0	2.0
parking-opt11	20	18	5	**6**	4	3	9.2	245.4	**19.3**	31.9	51.0	**409.0**	566.0	566.0	566.7
parking-opt14	20	20	7	**9**	**9**	**9**	22.9	550.8	**44.5**	74.7	119.4	**539.0**	752.4	752.6	755.4
pegsol-opt11	20	16	**20**	19	19	18	193.5	**21.9**	35.3	45.9	66.4	**23,539.4**	94,754.9	92,869.6	89,887.7
scanalyzer-opt11	20	6	5	**7**	**7**	6	38.0	185.9	**28.4**	49.3	96.4	**28,069.2**	37,136.6	36,877.2	37,130.4
sokoban-opt11	20	20	**20**	18	18	19	142.4	**6.7**	17.5	9.2	7.2	**1,056.9**	20,372.4	6,527.0	2,932.3
transport-opt11	20	0	0	**1**	**1**	**1**	-	-	-	-	-	-	-	-	-
transport-opt14	20	3	0	**1**	**1**	**1**	-	-	-	-	-	-	-	-	-
visitall-opt11	20	20	**16**	**16**	**16**	**16**	0.4	0.8	**0.5**	0.7	1.0	**244.8**	2,350.3	2,011.3	1,290.2
visitall-opt14	20	20	**17**	**17**	**17**	16	6.7	1.0	0.5	0.3	**0.2**	**30.6**	1,878.4	713.7	189.6
woodworking-opt11	20	20	14	14	**18**	17	1.0	125.2	55.8	**2.9**	4.4	**2,098.3**	168,813.7	5,876.3	5,283.5
Total/Average	374	303	224	221	**225**	222	80.2	173.3	**26.2**	31.0	46.0	**15,662.3**	51,351.1	33,658.2	26,566.7

In these domains, the proposition landmarks play a key role in the heuristic quality and the relaxed BDDs eliminate most paths that do not achieve these propositions.

In contrast, the LP relaxation has a significantly smaller gap in *floortiles*, *openstack*, *pegsol* and *sokoban*. In *floortiles*, propositional landmarks are not crucial to compute accurate heuristics. The other domains have a large number of zero-cost actions that add propositional landmarks, which explains the poor BDD heuristic quality.

8.3 Overall Performance Comparison

Table 2 presents the overall performance for the BDD and LP based heuristics. We also include the MIP model as the state-of-the-art approach for the DFP. The first set of columns, *Coverage*, shows the number of instances solved, where column '#' indicates the total number of instances per domain. The second and third set of columns, *Average Time* and *Average States Evaluated*, respectively, present the corresponding average performance for the instances that all techniques solved.

Despite the fact that the LP relaxation usually computes a stronger heuristic in s_I (Table 1), the BDD heuristic has equal or better coverage in 15 of 19 domains while the LP has equal or better coverage in 9. In *nomystery*, *barman-opt14* and *woodworking* the BDDs extract optimal plans early during search and need to explore fewer states to prove optimality. In *elevators*, *scanalyzer* and *transport* the BDD heuristic quality significantly improves when it is close to a goal state while the LP model struggles to improve the upper bound in these domains. In contrast, the BDD has weak performance in the two *floortile* domains and *sokoban*, which is mostly due to its poor heuristic quality.

Our results demonstrate that our three BDD heuristics perform differently. A larger width translates into a stronger heuristics and, consequently, fewer states evaluated. This is particularly true for *childsnack*, *sokoban* and *woodworking*, where $\mathcal{B}_4$ and $\mathcal{B}_8$ expand significantly fewer states and have higher coverage than $\mathcal{B}_2$. However, a larger width results in a more computationally expensive heuristic, which explains the higher coverage of $\mathcal{B}_4$ in comparison to $\mathcal{B}_8$ in most domains. We observed a similar performance decay when experimenting with bigger widths (i.e., $\mathcal{W} \in \{16, 32, 64\}$).

Lastly, we highlight the performance difference between the MIP model and its LP relaxation embedded in our search algorithm. As both approaches use the same heuristic (i.e., the LP relaxation), the difference is likely due to the sophisticated branch-bound-and-cut search in Gurobi, which include primal heuristics and cutting planes. A different search strategy may also improve BDD performance.

8.4 Relaxed BDDs as a Preprocessing Tool

Table 3 shows the average percentage of redundant actions that a relaxed BDD identifies after eliminating actions using

Table 3: Redundant actions found by BDDs.

	Average % of redundant actions		
domain	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$
barman-opt11	1.2%	1.2%	1.2%
barman-opt14	1.0%	1.0%	1.0%
childsnack-opt14	0.0%	0.0%	0.0%
elevators-opt11	0.3%	0.3%	0.4%
floortile-opt11	0.0%	0.0%	0.0%
floortile-opt14	0.0%	0.0%	0.0%
ged-opt14	6.2%	6.2%	6.2%
nomystery-opt11	21.6%	22.4%	22.9%
openstacks-opt11	0.0%	0.0%	0.0%
parking-opt11	0.8%	0.8%	0.8%
parking-opt14	0.8%	0.8%	0.8%
pegsol-opt11	0.0%	0.0%	0.0%
scanalyzer-opt11	0.0%	0.0%	0.0%
sokoban-opt11	8.8%	8.7%	8.7%
transport-opt11	0.0%	0.0%	0.0%
transport-opt14	0.0%	0.0%	0.0%
visitall-opt11	5.3%	5.3%	5.3%
visitall-opt14	0.4%	0.4%	0.4%
woodworking-opt11	0.0%	0.0%	0.0%

traditional techniques (see Section 8.1). We can see that the average percentage is high in some domains, especially in *ged*, *nomystery* and *sokoban* with 6% to 22%.

With respect to action landmarks, the BDD uncovers the same number of action landmarks as the AND-OR graph algorithm (Keyder, Richter, and Helmert 2010).

9 Conclusions

This work presents a new admissible heuristic for delete-free planning tasks based on relaxed binary decision diagrams (BDDs). Our experimental results show that relaxed BDDs have competitive performance compared to an LP-based heuristic, especially in domains where propositional landmarks play an important role. We show that relaxed BDDs can be used beyond heuristic computation. In particular, they enable the extraction of high-quality delete-free plans and the identification of redundant actions.

Acknowledgements

We would like to thank the anonymous reviewers whose valuable feedback helped improve the final paper. The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (Discovery Grant) and CONICYT (Becas Chile).

References

Behle, M. 2008. On threshold BDDs and the optimal variable ordering problem. *Journal of Combinatorial Optimization* 16(2):107–118.

Bergman, D.; Cire, A. A.; van Hoeve, W.-J.; and Hooker, J. N. 2012. Variable ordering for the application of BDDs to the maximum independent set problem. In *CPAIOR*, 34–49. Springer.

Bergman, D.; Cire, A. A.; van Hoeve, W.-J.; and Hooker, J. N. 2016. Discrete optimization with decision diagrams. *INFORMS Journal on Computing* 28(1):47–66.

Bergman, D.; van Hoeve, W.-J.; and Hooker, J. N. 2011. Manipulating mdd relaxations for combinatorial optimization. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 20–35. Springer.

Betz, C., and Helmert, M. 2009. Planning with h+ in theory and practice. In *Annual Conference on Artificial Intelligence*, 9–16. Springer.

Bonet, B., and Castillo, J. 2011. A complete algorithm for generating landmarks. In *ICAPS*, 315–318.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *ECAI*, volume 215, 329–334.

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

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69(1-2):165–204.

Castro, M. P.; Piacentini, C.; Cire, A. A.; and Beck, J. C. 2018. Relaxed decision diagrams for cost-optimal classical planning. In *Workshop HSDIP, ICAPS*, 50–58.

Corrêa, A. B.; Pommerening, F.; and Francès, G. 2018. Relaxed decision diagrams for delete-free planning. *Workshop on Constraints and AI Planning, CP*.

Haslum, P.; Slaney, J. K.; Thiébaux, S.; et al. 2012. Minimal landmarks for optimal delete-free planning. In *ICAPS*, 353–357.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *ICAPS*, 162–169.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Imai, T., and Fukunaga, A. 2014. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In *ECAI*, 459–464.

Imai, T., and Fukunaga, A. 2015. On a practical, integer-linear programming model for delete-free tasks and its use as a heuristic for cost-optimal planning. *Journal of Artificial Intelligence Research* 54:631–677.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *ECAI*, 335–340.

Land, A. H., and Doig, A. G. 1960. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society* 497–520.

Pommerening, F., and Helmert, M. 2012. Optimal planning for delete-free tasks with incremental LM-cut. In *ICAPS*, 363–367.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *ICAPS*, 226–234.