Journal of Artificial Intelligence Research 67 (2020) 607-651 Submitted 08/2019; published 03/2020

Solving Delete Free Planning with Relaxed Decision Diagram Based Heuristics

Margarita P. Castro MPCASTRO@MIE.UTORONTO.CA
Chiara Piacentini CHIARAP@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Ontario M5S 3G8, Canada

Andre A. Cire ANDRE.CIRE@UTORONTO.CA
Department of Management, University of Toronto Scarborough
Rotman School of Management
Toronto, Ontario M1E 1A4, Canada

J. Christopher Beck JCB@MIE.UTORONTO.CA
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Ontario M5S 3G8, Canada

Abstract

We investigate the use of relaxed decision diagrams (DDs) for computing admissible heuristics for the cost-optimal delete-free planning (DFP) problem. Our main contributions are the introduction of two novel DD encodings for a DFP task: a multivalued decision diagram that includes the sequencing aspect of the problem and a binary decision diagram representation of its sequential relaxation. We present construction algorithms for each DD that leverage these different perspectives of the DFP task and provide theoretical and empirical analyses of the associated heuristics. We further show that relaxed DDs can be used beyond heuristic computation to extract delete-free plans, find action landmarks, and identify redundant actions. Our empirical analysis shows that while DD-based heuristics trail the state of the art, even small relaxed DDs are competitive with the linear programming heuristic for the DFP task, thus, revealing novel ways of designing admissible heuristics.

1. Introduction

Cost-optimal delete-free planning (DFP) is a variant of classical planning that ignores the delete effects of actions. It is an NP-hard problem (Bylander, 1994) that has been extensively investigated by the planning community as the basis for efficient methodologies to address classical planning problems (Betz & Helmert, 2009; Helmert & Domshlak, 2009). Moreover, some challenging planning problems can be formulated as delete free tasks, such as the minimal seed set problem (Gefen & Brafman, 2011).

In this work, we explore the effectiveness of relaxed decision diagrams (DDs) for computing admissible heuristics for a DFP task.[1] Relaxed DDs are graphical structures that encode a superset of the solutions of a discrete optimization problem. They have recently formed the core of state-of-the-art methodologies in a variety of combinatorial optimiza-

1. A preliminary version of this work appeared in Castro et al. (2019a) which introduces the relaxed binary decision diagram heuristic.

tion problems (Bergman, Cire, van Hoeve, & Hooker, 2016) as a replacement of the linear programming (LP) relaxation. In particular, recent work has shown that relaxed DDs are effective for solving sequencing problems (Cire & van Hoeve, 2013; Castro et al., 2019b).

The paper introduces a new family of heuristics based on relaxed DDs and, thus, proposes a new path for heuristic development in the field. As a first step, this work shows the potential of relaxed DD heuristics over DFP tasks. Nonetheless, these heuristics can be used in more challenging planning variants with minor modifications to the proposed implementation, e.g., inside cost-optimal classical planners. We also relate the DD heuristics to well-known heuristics in the planning community (i.e., critical path and disjunctive landmarks), opening new research avenues using relaxed DDs to combine and create new admissible heuristics. Moreover, we introduce a flexible construction procedure based on node information that can be extended to consider, e.g., numerical variables.

Contributions. We present a multivalued decision diagram (MDD) encoding of a DFP task and a binary decision diagram (BDD) representation of the DFP sequential relaxation. We investigate the structural properties of each graphical model and present a numerical comparison with the DFP LP relaxation (Imai & Fukunaga, 2014, 2015). Specifically, we propose construction procedures for each graphical structure that guarantee the admissibility and consistency of their resulting heuristics. We explore the theoretical properties of our relaxed DDs and relate their heuristics to existing techniques in the literature. Furthermore, we show how to leverage these graphical structures to identify landmarks, redundant actions, and extract delete-free plans.

The paper presents an extensive empirical analysis that highlights the advantages and disadvantages of using relaxed DD-based heuristics in place of the LP relaxation. We show that, even with small relaxed DDs, our heuristics have competitive performance on DFP tasks. While still the state of the art in the majority of the domains, our relaxed MDD and BDD approaches outperform the mixed-integer linear programming (MILP) model (Imai & Fukunaga, 2015) in four delete-free IPC domains.

The rest of the paper is as follows. Section 2 describes DFP and its sequential relaxation. Section 3 presents a brief literature review on DFP tasks and relaxed DDs. Section 4 introduces our relaxed MDD encoding and construction procedure, while Section 5 presents the relaxed MDD heuristic, its theoretical properties and relationship to existing techniques. Similarly, Section 6 describes the relaxed BDD encoding for the sequential relaxation and Section 7 discusses the theoretical properties of the heuristic. Section 8 highlights the main differences between the graphical structures and provides guidance on when each approach is more appropriate. Section 9 presents alternative uses of relaxed DDs beyond heuristic computation. Section 10 explains our planner implementation and Section 11 presents an empirical analysis of our proposed methods. The paper ends with concluding remarks and potential research directions.

2. Problem Definition

This work considers cost-optimal DFP using the STRIPS formalism (Fikes & Nilsson, 1971) 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. Each proposition $p \in \mathcal{P}$ can be either true or false. We define a state s by its set of true propositions, i.e., we say that p is true in s if $p \in s$ and false otherwise.

An action $a \in \mathcal{A}$ is a tuple $\langle \texttt{pre}(a), \texttt{add}(a), \texttt{cost}(a) \rangle$, 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 $\texttt{cost}(a) \geq 0$ 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 $s' = \texttt{succ}(s, a) := s \cup \texttt{add}(a)$. 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, a_2 is applicable in $s_1 = \texttt{succ}(s, a_1)$, and a_i is applicable in $s_{i-1} = \texttt{succ}(s_{i-2}, a_{i-1})$ for all $i \in \{3, \ldots, n\}$. Extending the notation, given a state s and a sequence of applicable actions $(a_1, \ldots, a_n)$, $\texttt{succ}(s, (a_1, \ldots, a_n))$ represents the resulting state after applying the action sequence.

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 $s_G := \texttt{succ}(s_I, \pi)$ satisfies all goal propositions, i.e., $\mathcal{G} \subseteq s_G$. We use notation $\pi \in \Pi^+$ to indicate that the sequence of actions π is a plan for the DFP task Π^+. Similarly, we say that π is an *invalid* plan if it is not a plan, i.e., $\pi \notin \Pi^+$. The cost of a plan is given by the total cost of its actions, i.e., $\texttt{cost}(\pi) := \sum_{a \in \pi} \texttt{cost}(a)$ where $a \in \pi$ represents that action a is part of plan π. In particular, a cost-optimal plan π^* for task Π^+ is a plan with minimal cost, i.e, $\texttt{cost}(\pi^*) \leq \texttt{cost}(\pi)$ for all plans $\pi \in \Pi^+$.

Throughout this paper we assume that the DFP task Π^+ is solvable, i.e., there exists a plan $\pi \in \Pi^+$. Since it is possible to check if Π^+ is solvable in polynomial time (Blum & Furst, 1997), this assumption does not limit the applicability of our approach.

We say that $p \in \mathcal{P}$ is a *propositional landmark* if p is true at some point in all valid plans (Porteous, Sebastia, & Hoffmann, 2001). In the DFP case, $p \in \mathcal{P}$ is a propositional landmark if $p \in \texttt{succ}(s_I, \pi)$ for all plans $\pi \in \Pi^+$.We use the symbol $\mathcal{L}^{\mathcal{P}}$ to represent the set of all propositional landmarks. Notice that, by definition, all goal propositions are propositional landmarks, i.e., $\mathcal{G} \subseteq \mathcal{L}^{\mathcal{P}}$. Similarly, we say that an action $a \in \mathcal{A}$ is an *action landmark* if $a \in \pi$ for all $\pi \in \Pi^+$, i.e., action a is present in all plans of task Π^+.

Recall that it is possible to extract all propositional landmarks in polynomial time for any DFP task, e.g., using the relaxed planning graph (Porteous et al., 2001). However, in classical planning, the problem of finding all propositional landmarks is as hard as solving the planning task itself (i.e., PSPACE-hard).

2.1 Sequential Relaxation

This work considers a relaxation of the DFP task to compute admissible heuristics based on BDDs. The sequential relaxation of a DFP task, also known as a temporal relaxation, ignores the order in which the actions are applied (Imai & Fukunaga, 2015).

Definition 2.1. Given a DFP task Π^+, a valid sequence-relaxed plan for Π^+, $\texttt{sr}$-plan, is a set of actions $\pi_{\texttt{sr}} = \{a_1, \ldots, a_n\}$ such that:

(i) For every action $a \in \pi_{\texttt{sr}}$, each proposition $p \in \texttt{pre}(a)$ is true in s_I or is added by some action $a' \in \pi_{\texttt{sr}}$, $a \neq a'$.

(ii) Each goal $p \in \mathcal{G}$ is true in s_I or is added by some action $a \in \pi_{\texttt{sr}}$.

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

Example 2.1. Consider the following DFP task Π_4^+ of the visit-all domain (García-Olaya, Jiménez, & Linares López, 2011). The set of propositions is given by $\mathcal{P} = \{i_1, v_1, i_2, v_2, i_3, v_3, i_4, v_4\}$, where propositions i_k and v_k represent that the agent *is currently at* and *has visited* room $k \in \{1, 2, 3, 4\}$, respectively. The set of actions $\mathcal{A} = \{a_{1,2}, a_{2,1}, a_{2,3}, a_{3,2}, a_{3,4}, a_{4,3}, a_{1,4}, a_{4,1}\}$ is such that $a_{k,r} \in \mathcal{A}$ represents the movement from room k to r with $\texttt{pre}(a_{k,r}) = \{i_k\}$, $\texttt{add}(a_{k,r}) = \{i_r, v_r\}$ and $\texttt{cost}(a_{k,r}) = 1$. Figure 1 depicts the initial state (left image) and goal conditions (right image), where the human symbol (🚹) represents the position of the agent and the square (■) a visited room.

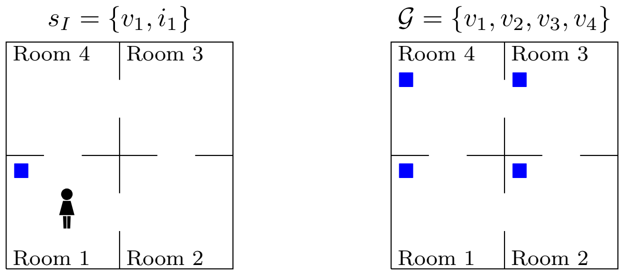


Figure 1: Visit-all domain with 4 rooms.

A cost-optimal plan for task Π_4^+ is $\pi = (a_{1,2}, a_{2,3}, a_{3,4})$ with $\texttt{cost}(\pi) = 3$. A cost-optimal $\texttt{sr}$-plan is $\pi_{\texttt{sr}} = \{a_{2,3}, a_{3,2}, a_{3,4}\}$ with $\texttt{cost}(\pi_{\texttt{sr}}) = 3$. Notice that while π is also a cost-optimal $\texttt{sr}$-plan, $\pi_{\texttt{sr}}$ is an invalid plan since no action $a \in \pi_{\texttt{sr}}$ is applicable in s_I.

3. Related Works

Cost-optimal DFP is a well-studied problem in the planning community and has inspired several state-of-the-art heuristics (Betz & Helmert, 2009). In particular, Bonet and Helmert (2010) show that a DFP task 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 set of disjunctive landmarks (Bonet & Castillo, 2011) and the set-inclusion minimal disjunctive landmarks (Haslum, Slaney, & Thiébaux, 2012) 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.

Imai and Fukunaga (2014) presented a MILP formulation for a DFP task with polynomially many constraints. This formulation is currently regarded as the state-of-the-art solution approach for DFP tasks. Moreover, its LP relaxation coupled with operator counting constraints (Pommerening, Röger, Helmert, & Bonet, 2014) defines an admissible heuristic that achieved competitive performance with respect to state-of-the-art classical planning heuristics (Imai & Fukunaga, 2015).

LP-based heuristics have become a popular technique to compute informative heuristics in delete-free, classical, and numerical planning tasks (Piacentini et al., 2018; Scala et al., 2017). While the LP relaxation is commonly used in the Operations Research (OR) community for bound computation, Andersen, Hadzic, Hooker, and Tiedemann (2007) present an alternative discrete relaxation approach based on decision diagrams (Bryant, 1986). The technique has had a growing interest in the OR community with successful applications to sequencing (Cire & van Hoeve, 2013; Kinable et al., 2017; Castro et al., 2019b) and binary decision problems (Bergman et al., 2016).

Recent work has also shown an interest on relaxed DDs to compute admissible heuristics for planning tasks. Castro, Piacentini, Cire, and Beck (2018) use MDDs 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 techniques proposed in this paper are related to the work by Corrêa, Pommerening, and Francès (2018), who apply relaxed DDs to approximate the state-space of a DFP task. Our methodology, however, differs in three main ways. First, the authors use a BDD encoding with a dynamic action branching per node, while we propose an MDD encoding that focuses on the sequencing aspect of the problem. Second, our BDD representation models the sequential relaxation of a DFP task instead of the full DFP task. Lastly, Corrêa et al. (2018) implement a top-down DD construction, while we propose an iterative splitting procedure for our two DD approaches. While the top-down construction is faster, the iterative splitting leverages the encoded information to create a stronger relaxation.

There are other applications of decision diagrams in planning that are not closely related to our approach but that we note here for completeness. BDDs have been used in planning to succinctly represent sets of states (symbolic states). Using this representation, a symbolic version of the A^* search algorithm achieved state-of-the-art performance in cost-optimal classical planning (Torralba, Linares López, & Borrajo, 2016). Several admissible heuristics have been proposed to guide the search over the symbolic state-space, e.g., abstraction-based heuristics (Edelkamp et al., 2012; Torralba et al., 2013). Lastly, the planning literature has utilized edge-value multivalued decision diagrams to represent cost functions of planning problems with state-dependent actions costs (Keller et al., 2016; Geißer et al., 2016).

4. Relaxed MDD Encoding and Construction Procedure

This section presents an MDD encoding of a DFP task Π^+ and introduces the relaxed MDD construction procedure that will be used to compute an admissible heuristic. The main idea is to represent the sequential aspect of the problem to compactly encode all optimal plans using an MDD. In particular, we construct an MDD where the i-th decision layer represents the i-th action in a cost-optimal plan. Since the number of actions in any cost-optimal plan is unknown, our encoding considers a dummy action $a_{\texttt{noop}}$ that represents the execution of no further actions: $\texttt{pre}(a_{\texttt{noop}}) = \mathcal{G}$, $\texttt{add}(a_{\texttt{noop}}) = \emptyset$, and $\texttt{cost}(a_{\texttt{noop}}) = 0$. We use notation $\mathcal{A}_0 = \mathcal{A} \cup \{a_{\texttt{noop}}\}$ to refer to the set of actions including $a_{\texttt{noop}}$. In the following, we assume that we have an upper bound on the maximum number of actions needed by any cost-optimal plan $\mathtt{m} \leq |\mathcal{A}|$ (see Section 4.5 on how to compute $\mathtt{m}$). Note that $|\mathcal{A}|$ is a trivial upper bound on the maximum number of actions needed for any solvable DFP task Π^+.

An MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ is a directed acyclic graph, where $\mathcal{N}^{\mathcal{M}}$ and $\mathcal{E}^{\mathcal{M}}$ correspond to the set of nodes and edges, respectively. The set of nodes $\mathcal{N}^{\mathcal{M}} = (\mathcal{N}^{\mathcal{M}}_1, ..., \mathcal{N}^{\mathcal{M}}_{\mathtt{m}+1})$ is partitioned into $\mathtt{m}+1$ disjoint node layers, i.e., $\mathcal{N}^{\mathcal{M}}_i \cap \mathcal{N}^{\mathcal{M}}_j = \emptyset$ for all $i \neq j \in \{1, ..., \mathtt{m}+1\}$. The first and last layer have a single node, $\mathcal{N}^{\mathcal{M}}_1 = \{\mathbf{r}\}$ and $\mathcal{N}^{\mathcal{M}}_{\mathtt{m}+1} = \{\mathbf{t}\}$, denoted by the root node $\mathbf{r}$ and terminal node $\mathbf{t}$, respectively. Similarly, the set of edges $\mathcal{E}^{\mathcal{M}} = (\mathcal{E}^{\mathcal{M}}_1, ..., \mathcal{E}^{\mathcal{M}}_{\mathtt{m}})$ is partitioned into $\mathtt{m}$ disjoint edge layers. In particular, each edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}_i$ emanates from a node $u \in \mathcal{N}^{\mathcal{M}}_i$ and points to a node $v \in \mathcal{N}^{\mathcal{M}}_{i+1}$.

In this representation, each edge in layer $\mathcal{E}^{\mathcal{M}}_i$ maps to the i-th action in a plan. Every edge $e \in \mathcal{E}^{\mathcal{M}}_i$ is associated with an action $\mathtt{a}(e) \in \mathcal{A}_0$, and has a cost $\mathtt{c}(e)$ that depends on the cost of its associated action (see Section 4.2). Each node $u \in \mathcal{N}^{\mathcal{M}}$ has at most $|\mathcal{A}_0|$ emanating edges, each one associated with a different action.

An $\mathbf{r}-\mathbf{t}$ path $\rho := (e_1, ..., e_{\mathtt{m}}) \in \mathcal{M}$ is a sequence of edges, where each edge is in a unique layer, and the target node of $e_i \in \mathcal{E}^{\mathcal{M}}_i$ is the source node of $e_{i+1} \in \mathcal{E}^{\mathcal{M}}_{i+1}$ for $i \in \{1, ..., \mathtt{m}-1\}$. Our MDD encoding interprets nodes as reachable states from s_I. Given a node $u \in \mathcal{N}^{\mathcal{M}}$, $\alpha^{\downarrow}(u)$ corresponds to the set of achieved propositions in node u. Starting from the root node with $\alpha^{\downarrow}(\mathbf{r}) := s_I$, for each node $v \in \mathcal{N}^{\mathcal{M}}_i$ $(i > 1)$ we define $\alpha^{\downarrow}(v)$ as:

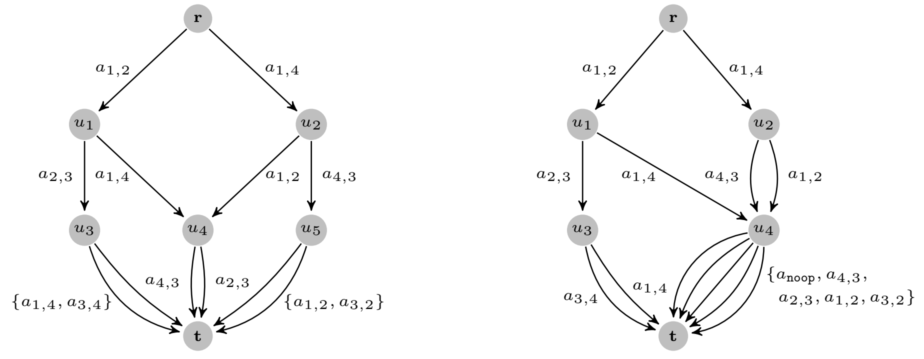
$$\alpha^{\downarrow}(v) := \bigcup_{e=(u,v) \in \mathcal{E}^{\mathcal{M}}_{i-1}} \alpha^{\downarrow}(u) \cup \mathtt{add}(\mathtt{a}(e)).$$

We can identify all $\mathbf{r}-\mathbf{t}$ paths that correspond to valid plans if every node $u \in \mathcal{N}^{\mathcal{M}}$ represents a single state in the search-space, i.e., all $\mathbf{r}-u$ paths achieve the same set of propositions. Then, it is sufficient to check that every edge e emanating from a node $u \in \mathcal{N}^{\mathcal{M}}$ is associated with an applicable action (i.e., $\mathtt{pre}(\mathtt{a}(e)) \subseteq \alpha^{\downarrow}(u)$) and the terminal node achieves all the goals (i.e., $\mathcal{G} \subseteq \alpha^{\downarrow}(u) \cup \mathtt{add}(\mathtt{a}(e))$ for each edge $e = (u, \mathbf{t}) \in \mathcal{E}^{\mathcal{M}}_{\mathtt{m}}$) to ensure that all $\mathbf{r}-\mathbf{t}$ paths in $\mathcal{M}$ are valid plans. We say that an MDD $\mathcal{M}$ is *exact* if every $\mathbf{r}-\mathbf{t}$ path $\rho \in \mathcal{M}$ corresponds to a cost-optimal plan for Π^+ and there exists an $\mathbf{r}-\mathbf{t}$ path in $\mathcal{M}$ for each cost-optimal plan $\pi \in \Pi^+$.

Building an exact MDD is intractable since, in the worst case, its size is equivalent to the state-space of the DFP task Π^+. We tackle this problem by considering a *relaxed* MDD instead. A relaxed MDD is a limited size MDD where each node aggregates one or more states, i.e., it over-approximates the set of cost-optimal plans. Formally, $\mathcal{M}$ is a relaxed MDD for a DFP task Π^+ if there exists an $\mathbf{r}-\mathbf{t}$ path $\rho \in \mathcal{M}$ for each cost-optimal plan of size at most $\mathtt{m}$, but some $\mathbf{r}-\mathbf{t}$ paths may correspond to invalid or sub-optimal plans. The following example shows an exact and a relaxed MDD for our running example.

Example 4.1. Consider the DFP task Π^+_4 described in Example 2.1. Figure 2a and 2b, respectively, illustrate an exact and a relaxed MDD for this domain with $\mathtt{m} = 3$. The action associated with each edge is shown next to the edge. When edges are too close to each other, we use a set notation to represent the actions associated with a set of edges.

Notice that every $\mathbf{r}-\mathbf{t}$ path in the exact MDD (Figure 2a) corresponds to a cost-optimal plan for Π^+_4 and every cost-optimal plan for Π^+_4 has a corresponding $\mathbf{r}-\mathbf{t}$ path. In contrast, the relaxed MDD shown in Figure 2b has a path for each cost-optimal plan of Π^+_4 but there exist some paths that are invalid plans. For example, path $\rho = (a_{1,4}, a_{1,2}, a_{\texttt{noop}})$ is an invalid plan because action $a_{\texttt{noop}}$ is not applicable in state $s = \texttt{succ}(s_I, (a_{1,4}, a_{1,2}))$.



(a) Exact MDD.

(b) Relaxed MDD.

Figure 2: Exact and relaxed MDDs for the 4-room visit-all DFP task.

4.1 Relaxed MDD Construction Procedure

In the following, we show how to construct a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ for a DFP task Π^+. We define the width of a relaxed MDD $\mathcal{M}$ as the maximum number of nodes in a layer, i.e., $w(\mathcal{M}) := \max\{|\mathcal{N}_i^{\mathcal{M}}| : i \in \{1, ..., \texttt{m}+1\}\}$. We limit the size of $\mathcal{M}$ by bounding its width, $w(\mathcal{M}) \leq \mathcal{W}$, with $\mathcal{W} \in \mathbb{Z}^+$.

Algorithm 1 presents our relaxed MDD construction scheme. The algorithm starts by constructing a width-one MDD ($\mathcal{W} = 1$). The WidthOneMDD procedure receives the maximum number of layers $\texttt{m}$ and creates $\texttt{m}+1$ node layers and $\texttt{m}$ edge layers. Each node layer has a single node where $\mathbf{r}$ corresponds to the initial state s_I and each node $u \in \mathcal{N}_i^{\mathcal{M}}$ ($i > 1$) represents the union of all states that can be reached after applying i actions. Each edge layer has at most $|\mathcal{A}_0|$ edges, where each edge is associated with a different action in $\mathcal{A}_0$ that is applicable to its source node.

The construction procedure increases the width of the MDD and removes invalid $\mathbf{r} - \mathbf{t}$ paths (i.e., paths corresponding to invalid plans) until the MDD cannot be further updated (i.e., it reaches the maximum width or no more invalid plans are found). The UpdateMDDTop procedure iterates over the node layers starting from $\mathcal{N}_1^{\mathcal{M}}$. For each $i \in \{1, ..., \texttt{m}\}$, the procedure (i) updates the information stored in every node $u \in \mathcal{N}_i^{\mathcal{M}}$ considering all partial $\mathbf{r} - u$ paths (UpdateMDDNodesTop), (ii) splits every node $u \in \mathcal{N}_i^{\mathcal{M}}$ until the maximum width is reached (SplitMDDNodes), and lastly (iii) eliminates all edges that can be proved to only participate in invalid plans (FilterMDDEdges). In contrast, the UpdateMDDBottom procedure iterates over each layer starting with $\mathcal{N}_{\texttt{m}+1}^{\mathcal{M}}$, updates the information stored in every node $u \in \mathcal{N}_i^{\mathcal{M}}$ considering all partial $u - \mathbf{t}$ paths (UpdateMDDNodesBottom), and removes edges that are associated only with invalid plans.

The construction procedure ends when the MDD cannot be further updated and returns the resulting MDD. Hence, it applies the UpdateMDDTop and UpdateMDDBottom procedures at least twice before ending. The following subsections provide a detailed explanation of each procedure.

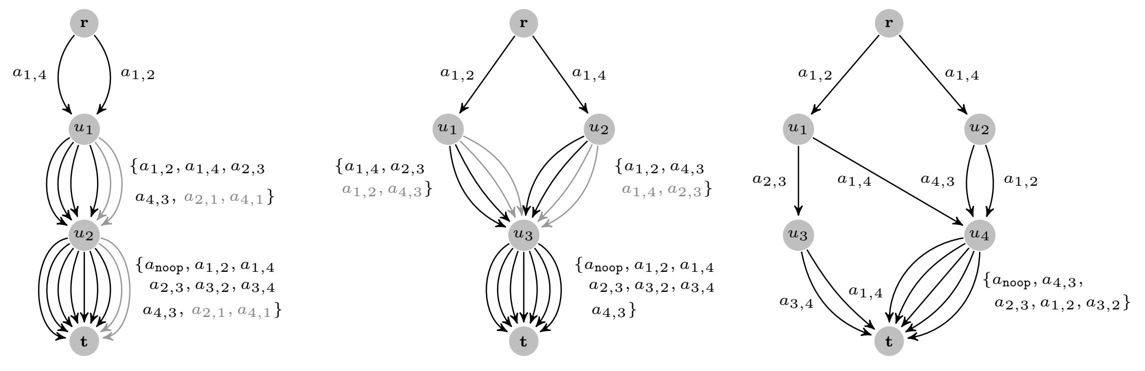
Algorithm 1 Relaxed MDD (BDD) Construction Procedure

```
1: procedure ConstructMDD($\Pi^+$, $\mathcal{W}$, m)
2:     $\mathcal{M}$ := WidthOneMDD(m)
3:     while $\mathcal{M}$ has been modified do
4:         UpdateMDDTop($\mathcal{M}$, $\mathcal{W}$, $\Pi^+$)
5:         UpdateMDDBottom($\mathcal{M}$, $\mathcal{W}$, $\Pi^+$)
6:     return $\mathcal{M}$

7: procedure UpdateMDDTop($\mathcal{M}$, $\mathcal{W}$, $\Pi^+$)
8:     for $i \in \{1, ..., \mathtt{m}\}$ do
9:         UpdateMDDNodesTop($\mathcal{N}_i^{\mathcal{M}}$)
10:        if $i \geq 2$ then SplitMDDNodes($\mathcal{N}_i^{\mathcal{M}}$, $\mathcal{W}$)
11:        FilterMDDEdges($\mathcal{E}_i^{\mathcal{M}}$)
12:    UpdateMDDNodesTop($\mathcal{N}_{\mathtt{m}+1}^{\mathcal{M}}$)

13: procedure UpdateMDDBottom($\mathcal{M}$, $\Pi^+$, $\mathcal{W}$)
14:    $i := \mathtt{m} + 1$
15:    while $i \geq 0$ do
16:        UpdateMDDNodesBottom($\mathcal{N}_i^{\mathcal{M}}$)
17:        FilterMDDEdges($\mathcal{E}_{i-1}^{\mathcal{M}}$)
18:        $i := i - 1$
```

Example 4.2. Consider the DFP task Π_4^+ described in Example 2.1. Figure 3 illustrates some of the steps of Algorithm 1 for a relaxed MDD with $\mathtt{m} = 3$ and $\mathcal{W} = 2$. Figure 3a corresponds to the initial width-one MDD where gray edges will be removed by the FilterMDDEdges procedure (Rule M3, Section 4.3). Figure 3b illustrates the resulting MDD after applying the SplitMDDNodes procedure in the second layer. Lastly, Figure 3c depicts the MDD returned by the construction procedure.


r
$a_{1,4}$
$a_{1,2}$
u_1
$\{a_{1,2}, a_{1,4}, a_{2,3}$
$a_{4,3}, a_{2,1}, a_{4,1}\}$
u_2
$\{a_{\mathtt{noop}}, a_{1,2}, a_{1,4}$
$a_{2,3}, a_{3,2}, a_{3,4}$
$a_{4,3}, a_{2,1}, a_{4,1}\}$
t
r
$a_{1,2}$
$a_{1,4}$
u_1
u_2
$\{a_{1,4}, a_{2,3}$
$a_{1,2}, a_{4,3}\}$
$\{a_{1,2}, a_{4,3}$
$a_{1,4}, a_{2,3}\}$
u_3
$\{a_{\mathtt{noop}}, a_{1,2}, a_{1,4}$
$a_{2,3}, a_{3,2}, a_{3,4}$
$a_{4,3}\}$
t
r
$a_{1,2}$
$a_{1,4}$
u_1
u_2
$a_{2,3}$
$a_{1,4}$
$a_{4,3}$
$a_{1,2}$
u_3
u_4
$a_{3,4}$
$a_{1,4}$
$\{a_{\mathtt{noop}}, a_{4,3},$
$a_{2,3}, a_{1,2}, a_{3,2}\}$
t

(a) Width-one MDD. (b) MDD after one node split. (c) Resulting MDD.

Figure 3: MDD construction example with $\mathcal{W} = 2$.

4.2 MDD Node Representation

Each node $u \in \mathcal{N}^{\mathcal{M}}$ stores information regarding the set of achieved and needed propositions by all paths traversing u. This information is then used in the SplitMDDNodes procedure to decide which nodes to split, and in the FilterMDDEdges procedure to identify invalid plans. The node information is aggregated for the $\mathbf{r} - u$ paths (i.e., top-down node information) and the $u - \mathbf{t}$ paths (i.e., bottom-up node information).

In the following, consider $\mathtt{s}(e) := u$ and $\mathtt{t}(e) := v$ to represent the source and target node of an edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}$, respectively. We use sets $\delta_{\mathtt{in}}(u)$ and $\delta_{\mathtt{out}}(u)$ to represent the set of incoming and outgoing edges of a node $u \in \mathcal{N}^{\mathcal{M}}$, respectively.

Given a node $u \in \mathcal{N}^{\mathcal{M}}$, the top-down information is the set of propositions that are achieved by the $\mathbf{r} - u$ paths. In particular, $\alpha^{\downarrow}_{\mathtt{A}}(u)$ represents the set of propositions that *all* $\mathbf{r} - u$ paths achieve and $\alpha^{\downarrow}_{\mathtt{S}}(u)$ the set of propositions that *some* $\mathbf{r} - u$ paths achieve. Starting with the root node $\mathbf{r}$, the UpdateMDDNodesTop procedure assigns $\alpha^{\downarrow}_{\mathtt{A}}(\mathbf{r}) := \alpha^{\downarrow}_{\mathtt{S}}(\mathbf{r}) := s_I$ and updates each node $u \in \mathcal{N}^{\mathcal{M}} \setminus \{\mathbf{r}\}$ as

$$\alpha^{\downarrow}_{\mathtt{A}}(u) := \bigcap_{e \in \delta_{\mathtt{in}}(u)} \Big(\alpha^{\downarrow}_{\mathtt{A}}(\mathtt{s}(e)) \cup \mathtt{add}(\mathtt{a}(e))\Big),$$

$$\alpha^{\downarrow}_{\mathtt{S}}(u) := \bigcup_{e \in \delta_{\mathtt{in}}(u)} \Big(\alpha^{\downarrow}_{\mathtt{S}}(\mathtt{s}(e)) \cup \mathtt{add}(\mathtt{a}(e))\Big).$$

The UpdateMDDNodesBottom procedure updates the bottom-up information of a node. For each node $u \in \mathcal{N}^{\mathcal{M}}$, set $\alpha^{\uparrow}_{\mathtt{S}}(u)$ keeps track of all the propositions achieved by *some* $u - \mathbf{t}$ path. Starting with the terminal node $\mathbf{t}$, the procedure assigns $\alpha^{\uparrow}_{\mathtt{S}}(\mathbf{t}) := \emptyset$ and updates each $u \in \mathcal{N}^{\mathcal{M}} \setminus \{\mathbf{t}\}$ as

$$\alpha^{\uparrow}_{\mathtt{S}}(u) := \bigcup_{e \in \delta_{\mathtt{out}}(u)} \Big(\alpha^{\uparrow}_{\mathtt{S}}(\mathtt{t}(e)) \cup \mathtt{add}(\mathtt{a}(e))\Big).$$

The procedure also saves the set of propositions needed by *some* $u - \mathbf{t}$ path, $\eta^{\uparrow}_{\mathtt{S}}(u)$. Starting with the terminal node $\mathbf{t}$, it assigns $\eta^{\uparrow}_{\mathtt{S}}(\mathbf{t}) := \mathcal{L}^{\mathcal{P}}$ and updates each node $u \in \mathcal{N}^{\mathcal{M}}$ as

$$\eta^{\uparrow}_{\mathtt{S}}(u) := \bigcup_{e \in \delta_{\mathtt{out}}(u)} \Big(\eta^{\uparrow}_{\mathtt{S}}(\mathtt{t}(e)) \cup \mathtt{pre}(\mathtt{a}(e))\Big).$$

It is sufficient to initialize $\eta^{\uparrow}_{\mathtt{S}}(\mathbf{t})$ with the set of propositional goals $\mathcal{G}$ to ensure that the algorithm is correct. However, initializing the set of needed propositions with $\mathcal{L}^{\mathcal{P}} \supseteq \mathcal{G}$ helps our algorithm to identify a larger number of invalid $\mathbf{r} - \mathbf{t}$ paths (see Section 4.3).

Lastly, our UpdateMDDNodesTop and UpdateMDDNodesBottom procedures save information about the cost of the paths traversing a node $u \in \mathcal{N}^{\mathcal{M}}$ that will be used to compute our MDD heuristic (Section 5) and filter sub-optimal plans (Section 4.3). The UpdateMDDNodesTop procedure underestimates the minimum-cost of all $\mathbf{r} - u$ paths using the critical path computation for MDDs introduced by Castro et al. (2018). The idea is to estimate the minimum cost to achieve a proposition $p \in \mathcal{P}$ for each node $u \in \mathcal{N}^{\mathcal{M}}$, $\omega^{\downarrow}(u, p)$. To do so, the cost of an edge $e \in \mathcal{E}^{\mathcal{M}}$ corresponds to the cost of its associated action plus the cost of its most expensive precondition in its source node, i.e.,

$$\mathtt{c}(e) := \mathtt{cost}(\mathtt{a}(e)) + \max\Big\{\omega^{\downarrow}(\mathtt{s}(e), p) :\ p \in \mathtt{pre}(\mathtt{a}(e))\Big\}.$$

To have a better estimate of the minimum cost of an edge, we distinguish whether its associated action adds a proposition or not. Let $\mathtt{c}_p(e)$ be the cost of an edge $e \in \mathcal{E}^{\mathcal{M}}$ with respect to a proposition $p \in \mathcal{P}$:

$$\mathtt{c}_p(e) := \begin{cases} \mathtt{c}(e), & p \in \mathtt{add}(\mathtt{a}(e)), \\ \max\{\mathtt{c}(e), \mathtt{cost}(\mathtt{a}(e)) + \omega^{\downarrow}(\mathtt{s}(e), p)\}, & p \notin \mathtt{add}(\mathtt{a}(e)),\ p \in \alpha^{\downarrow}_{\mathtt{S}}(\mathtt{s}(e)), \\ \infty, & \text{otherwise.} \end{cases}$$

In the first case, $\mathtt{c}_p(e)$ corresponds to the critical path cost $\mathtt{c}(e)$. For the second case, we can under-approximate $\mathtt{c}_p(e)$ either using the critical path formula $\mathtt{c}(e)$ or summing up the action and source node costs. Since there is no dominance between these two values, we choose the maximum.

We now define the top-down cost of a node $u \in \mathcal{N}^{\mathcal{M}}$ and proposition $p \in \mathcal{P}$ as the minimum cost of each of its incoming edges, i.e., $\omega^{\downarrow}(\mathbf{r}, p) := 0$ for all $p \in \mathcal{P}$ and

$$\omega^{\downarrow}(u, p) := \begin{cases} \min_{e \in \delta_{\mathtt{in}}(u)}\{\mathtt{c}_p(e)\}, & p \in \alpha^{\downarrow}_{\mathtt{S}}(u), \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the cost of proposition p at node u is the minimum cost over all incoming edges if p is achieved by at least one $\mathbf{r} - u$ path. Otherwise, we set the cost to zero to guarantee the admissibility of our heuristic. Proposition 4.1 shows that this cost computation under-approximates the cost of each plan represented as a path in the MDD.

Proposition 4.1. Consider any path $\rho = (e_1, ..., e_{\mathtt{m}}) \in \mathcal{M}$ that corresponds to a plan $\pi = (a_1, ..., a_{\mathtt{m}}) \in \Pi^+$, with $a_i = \mathtt{a}(e_i)$. The cost of any partial plan $\pi_i = (a_1, ..., a_i)$ is an upper bound to the cost of each of proposition $p \in s_{i+1} = \mathtt{succ}(s_I, \pi_i)$ in the corresponding MDD node $u_{i+1} = \mathtt{t}(e_i) \in \mathcal{N}^{\mathcal{M}}_{i+1}$, i.e.,

$$\omega^{\downarrow}(u_{i+1}, p) \leq \mathtt{cost}(\pi_i), \qquad \forall\, p \in s_{i+1},\ i \in \{1, ..., \mathtt{m}\}. \tag{1}$$

Proof. We will prove (1) by induction. In the initial case $i = 1$ we have that $\omega^{\downarrow}(u_2, p) \leq 0 + \mathtt{cost}(a_1) = \mathtt{cost}(\pi_1)$ for all $p \in s_I \cup \mathtt{add}(a_1)$. Next, consider that (1) holds for some $i \geq 1$ and we want to prove that it holds for $i+1$. From (1) we have that $\max_{q \in s_{i+1}}\{\omega^{\downarrow}(u_{i+1}, q)\} \leq \mathtt{cost}(\pi_i)$. Finally, for any $p \in s_{i+2}$ we have that:

$$\begin{aligned} \omega^{\downarrow}(u_{i+2}, p) \leq \mathtt{c}_p(e_{i+1}) &\leq \max_{q \in s_{i+1}}\{\omega^{\downarrow}(u_{i+1}, q)\} + \mathtt{cost}(a_{i+1}) \\ &\leq \mathtt{cost}(\pi_i) + \mathtt{cost}(a_{i+1}) \\ &= \mathtt{cost}(\pi_{i+1}). \end{aligned}$$

□

For the bottom-up cost estimation we use a shortest path computation where the cost of an edge is simply the cost of its associated action. Let $\omega^{\uparrow}(u)$ be the bottom-up cost of a node $u \in \mathcal{N}^{\mathcal{M}}$. Then, we define $\omega^{\uparrow}(\mathbf{t}) := 0$ and, for each node $u \in \mathcal{N}^{\mathcal{M}} \setminus \{\mathbf{t}\}$,

$$\omega^{\uparrow}(u) := \min_{e \in \delta_{\mathtt{out}}(u)} \left\{\omega^{\uparrow}(\mathtt{t}(e)) + \mathtt{cost}(\mathtt{a}(e))\right\}.$$

4.3 MDD Filtering Rules

Our FilterMDDEdges procedure identifies and removes invalid $\mathbf{r} - \mathbf{t}$ paths in $\mathcal{M}$. The procedure relies on a set of rules that identify if all paths traversing an edge correspond to invalid (or sub-optimal) plans. If an edge $e \in \mathcal{E}^{\mathcal{M}}$ violates a rule, then the edge can be safely removed from $\mathcal{M}$. We first present two rules that, as shown in Proposition 4.2, are necessary conditions for any path to be a plan.

Rule M1. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}$ with $a = \mathsf{a}(e) \in \mathcal{A}_0$. Then, action a has to be applicable in its source node, i.e.,

$$\mathtt{pre}(a) \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u).$$

Rule M2. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}$ with $a = \mathsf{a}(e) \in \mathcal{A}_0$. Then, all propositional landmarks $\mathcal{L}^{\mathcal{P}}$ need to be achieved by at least one path traversing e, i.e.,

$$\mathcal{L}^{\mathcal{P}} \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u) \cup \mathtt{add}(a) \cup \alpha^{\uparrow}_{\mathtt{S}}(v),$$

where the right-hand side corresponds to the set of propositions that are achieved by paths traversing edge e.

Proposition 4.2. Consider a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ for a DFP task Π^+. Rules M1 and M2 are necessary conditions for any $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{M}$ to be a plan.

Proof. Consider any $\mathbf{r} - \mathbf{t}$ path $\rho = (e_1, ..., e_{\mathtt{m}}) \in \mathcal{M}$ that corresponds to a plan $\pi \in \Pi^+$. We will show that all edges $e \in \rho$ satisfy rules M1 and M2. Take any edge $e_i = (u, v) \in \rho$ with associated action $a = \mathsf{a}(e_i)$ and let $s_{i-1} = \mathtt{succ}(s_I, (\mathsf{a}(e_1), ..., \mathsf{a}(e_{i-1})))$. By definition of set $\alpha^{\downarrow}_{\mathtt{S}}$ (Section 4.2), we have that $s_{i-1} \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u)$. Then, since ρ is a plan, $\mathtt{pre}(a) \subseteq s_{i-1} \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u)$ so Rule M1 is satisfied. Similarly, by definition of $\alpha^{\downarrow}_{\mathtt{S}}$ and $\alpha^{\uparrow}_{\mathtt{S}}$, we have that $\mathcal{L}^{\mathcal{P}} \subseteq \mathtt{succ}(s_I, \pi) \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u) \cup \mathtt{add}(a) \cup \alpha^{\uparrow}_{\mathtt{S}}(v)$, so rule M2 is satisfied. □

Since we want to create a relaxed MDD that only considers cost-optimal plans, we develop two rules, M3 and M4, that identify sub-optimal plans and, hence, strengthen the relaxation. Rule M3 removes edges that correspond to redundant actions (i.e., do not add any useful proposition). Rule M4 keeps paths that have a cost smaller or equal to any known plan $\pi' \in \Pi^+$. Notice that π' can be obtained, for instance, using a greedy heuristic over the relaxed MDD (see Section 9).

Rule M3. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}$ with $a = \mathsf{a}(e) \in \mathcal{A}$ (i.e., $a \neq a_{\mathtt{noop}}$). Then, action a must add at least one needed proposition that has not been achieved yet.

$$\Big(\mathtt{add}(a) \setminus \alpha^{\downarrow}_{\mathtt{A}}(u)\Big) \cap \eta^{\uparrow}(v) \neq \emptyset.$$

Rule M4. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}$ with $\mathsf{a}(e) \in \mathcal{A}_0$ and any known plan $\pi' \in \Pi^+$. Then, the minimum cost path traversing edge e has to have a cost smaller or equal to π'.

$$\mathsf{c}(e) + \omega^{\uparrow}(v) \leq \mathtt{cost}(\pi').$$

Notice that rule M3 might remove some cost-optimal plans for domains with zero-cost actions, since applicable zero-cost actions can be added to a plan even though they are redundant. Nonetheless, this rule will always maintain all *minimal* cost-optimal plans (i.e., plans without redundant actions). Proposition 4.3 shows that these rules do not remove minimal cost-optimal plans.

Proposition 4.3. Consider a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ for a DFP task Π^+. Any $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{M}$ that corresponds to a minimal cost-optimal plan satisfies M3 and M4.

Proof. Consider any $\mathbf{r} - \mathbf{t}$ path $\rho = (e_1, ..., e_{\mathtt{m}}) \in \mathcal{M}$ that corresponds to a minimal cost-optimal plan $\pi = (a_1, ..., a_{\mathtt{m}}) \in \Pi^+$ with $a_i = \mathtt{a}(e_i)$ and $s_i = \mathtt{succ}(s_I, (a_1, ..., a_i))$. Consider any edge $e_i = (u, v) \in \rho$. Since π is a minimal plan, for all $a_i \in \pi$ there exists a proposition $p \in \mathtt{add}(a_i) \setminus s_{i-1}$ such that p is a precondition for some action a_j $(j > i)$ or $p \in \mathcal{G}$. Then, since $\alpha^{\downarrow}_{\mathtt{A}}(u) \subseteq s_{i-1}$, Rule M3 holds for any minimal plan.

The validity of Rule M4 comes from the fact that, for any $e_i = (u, v) \in \mathcal{E}^{\mathcal{M}}_i$ we have that $\mathtt{c}(e_i) \leq \sum_{j=1}^{i} \mathtt{cost}(a_j)$ and $\omega^{\uparrow}(v) \leq \sum_{j=i+1}^{n} \mathtt{cost}(a_j)$ (see Section 4.2). Then $\mathtt{c}(e) + \omega^{\uparrow}(v) \leq \mathtt{cost}(\pi) \leq \mathtt{cost}(\pi')$. □

Algorithm 2 Relaxed MDD Filtering Edges Procedure

```
1: procedure FilterMDDEdges($\mathcal{E}^{\mathcal{M}}_i$ )
2:     for $e \in \mathcal{E}^{\mathcal{M}}_i$ do
3:         if $e$ violates any of rule M1 to rule M4 then
4:             Eliminate edge $e$ for $\mathcal{E}^{\mathcal{M}}_i$. $\mathcal{M}$ has been modified
```

The FilterMDDEdges procedure (Algorithm 2) iterates over all edges in a layer (line 2) and remove the edges that violate any of our four filtering rules (lines 3-4). The procedure also checks if the MDD has been modified.

4.4 MDD Splitting Algorithm

The SplitMDDNodes procedure increases the size of $\mathcal{M}$ to strengthen the relaxation. Our procedure attempts to split nodes so each node represents exactly one state in the search-space. Given a node $u \in \mathcal{N}^{\mathcal{M}}$, we say that u is *exact* if $\alpha^{\downarrow}_{\mathtt{A}}(u) = \alpha^{\downarrow}_{\mathtt{S}}(u)$. As shown in Proposition 4.4, we can eliminate all invalid plans from an MDD if all its nodes are exact.

Proposition 4.4. Consider a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ where each node $u \in \mathcal{N}^{\mathcal{M}} \setminus \{\mathbf{t}\}$ is exact, i.e., $\alpha^{\downarrow}_{\mathtt{A}}(u) = \alpha^{\downarrow}_{\mathtt{S}}(u)$. Then, filtering rules M1 and M2 are sufficient to identify and remove all invalid paths in $\mathcal{M}$.

Proof. Consider any $\mathbf{r} - \mathbf{t}$ path $\rho = (e_1, ..., e_{\mathtt{m}}) \in \mathcal{M}$ with associated actions $a_i = \mathtt{a}(e_i)$ that form an invalid plan. Assume that $(a_1, ..., a_{i-1})$ is applicable to the initial state, i.e., $s_{i-1} = \mathtt{succ}(s_I, (a_1, ..., a_{i-1}))$, but a_i is not applicable to s_{i-1}. Let $u = \mathtt{s}(e_i)$. We have that $\alpha^{\downarrow}_{\mathtt{A}}(u) = s_{i-1} = \alpha^{\downarrow}_{\mathtt{S}}(u)$ by definition of $\alpha^{\downarrow}_{\mathtt{A}}$ and $\alpha^{\downarrow}_{\mathtt{S}}$, and the fact that node u is exact. Then, rule M1 eliminates edge e_i since $\mathtt{pre}(a_i) \not\subseteq s_{i-1} = \alpha^{\downarrow}_{\mathtt{S}}(u)$.

Now assume that $(a_1, ..., a_{\mathtt{m}})$ is applicable to the initial state but the resulting state omits at least one propositional goal, i.e., $\mathcal{G} \not\subseteq s_{\mathtt{m}} = \mathtt{succ}(s_I, (a_1, ..., a_{\mathtt{m}}))$. Then, rule M2 eliminates edge $e_{\mathtt{m}} = (u, \mathbf{t})$ since $\mathcal{G} \subseteq \mathcal{L}^{\mathcal{P}}$ and $\mathcal{G} \not\subseteq s_{\mathtt{m}} = s_{\mathtt{m}-1} \cup \mathtt{add}(a_{\mathtt{m}}) \cup \emptyset = \alpha^{\downarrow}_{\mathtt{S}}(u) \cup \mathtt{add}(a_{\mathtt{m}}) \cup \alpha^{\uparrow}_{\mathtt{S}}(\mathbf{t})$. □

Our SplitMDDNodes procedure (Algorithm 3) iterates over the set of nodes in a layer and splits inexact nodes into two such that the resulting nodes represent fewer aggregated states. To do so, the procedure sorts the propositions that are not in the initial state, $\mathcal{P}_{\neg s_I} := \mathcal{P} \setminus s_I$, in a priority queue Q and iterates over them (lines 2-3). For each proposition $p \in Q$, the algorithm looks for nodes $u \in \mathcal{N}_i^{\mathcal{M}}$ where proposition p is achieved by some $\mathbf{r} - u$ paths but not all (lines 4-6). We then identify the set of incoming edges where p is always achieved (line 7) and split the node into two: node u' where p is achieved by all $\mathbf{r} - u'$ paths, and node u where p is never achieved (lines 9-10). Lastly, the algorithm duplicates the outgoing edges from the original node u to the resulting node u'. Notice that it might be impossible to split u with respect to p: $\delta_{\texttt{all}}$ can be empty if the nodes in the previous layer correspond to the aggregation of distinct states (line 8).

Algorithm 3 Relaxed MDD Split Nodes Procedure

```
1: procedure SplitMDDNodes($\mathcal{N}_i^{\mathcal{M}}$, $\mathcal{W}$)
2:     Initialize $Q$ with all the propositions in $\mathcal{P}_{\neg s_I}$
3:     while $|Q| > 0$ and $|\mathcal{N}_i^{\mathcal{M}}| < \mathcal{W}$ do
4:         Remove first proposition $p$ in $Q$
5:         for $u \in \mathcal{N}_i^{\mathcal{M}}$ do
6:             if $p \in \alpha_{\mathtt{S}}^{\downarrow}(u)$ and $p \notin \alpha_{\mathtt{A}}^{\downarrow}(u)$ then
7:                 $\delta_{\mathtt{all}} := \{e \in \delta_{\mathtt{in}}(u) : p \in \mathtt{add}(\mathtt{a}(e))$ or $p \in \alpha_{\mathtt{A}}^{\downarrow}(\mathtt{s}(e))\}$
8:                 if $\delta_{\mathtt{all}} \neq \emptyset$ then
9:                     Create new node $u'$. $\mathcal{N}_i^{\mathcal{M}} := \mathcal{N}_i^{\mathcal{M}} \cup \{u'\}$
10:                    Redirect edges: $\delta_{\mathtt{in}}(u') := \delta_{\mathtt{all}}$ and $\delta_{\mathtt{in}}(u) := \delta_{\mathtt{in}}(u) \setminus \delta_{\mathtt{all}}$
11:                    if $\delta_{\mathtt{in}}(u') \neq \emptyset$ then Duplicate outgoing edges from $u$ to $u'$
```

Our algorithm uses a priority queue Q over the propositions in $\mathcal{P}_{\neg s_I}$ to split the nodes using the same propositional order. The queue is divided into three priority levels where goal propositions $p \in \mathcal{G}$ have the highest priority, followed by propositional landmarks ($p \in \mathcal{L}^{\mathcal{P}} \setminus \mathcal{G}$) and lastly the remaining propositions ($p \in \mathcal{P}_{\neg s_I} \setminus \mathcal{L}^{\mathcal{P}}$).

Notice that this splitting procedure needs at most $2^{|\mathcal{P}_{\neg s_I}|}$ nodes in each layer to create an exact MDD. Moreover, if $\mathcal{W} = 2^{|\mathcal{P}_{\neg s_I}|}$, nodes in layer $\mathcal{N}_i^{\mathcal{M}}$ represent all the states that can be reach after applying i actions.

4.5 Number of Layers Estimation

We propose two simple procedures to estimate $\mathtt{m}$ using the cost of any plan $\pi' \in \Pi^+$. Notice that we can generate a plan for a DFP task Π^+ using, for instance, the FF heuristic (Hoffmann & Nebel, 2001).

Algorithm 4 shows our two estimation procedures, MaxLayerActions and MaxLayerPropositions. Procedure MaxLayerActions uses the fact that each action is applied at most once. The procedure sorts the actions in increasing order of cost (line 2) and sums up the cost of each action in the queue until the total cost C is greater than the cost of our known plan π' (lines 3-6). Then, the total number of actions used to compute C is an upper bound on the maximum number of actions in any cost-optimal plan.

The second procedure, MaxLayerPropositions, is valid only for minimal cost-optimal plans (i.e., plans without redundant actions). The procedure estimates the maximum number of

Algorithm 4 Maximum number of layers estimation

```
1: procedure MaxLayerActions($\mathcal{A}$, $\pi'$)
2:     $Q_{\mathcal{A}} = (a_1, ...., a_{|\mathcal{A}|})$, list of actions ordered in increasing value of $\texttt{cost}(a)$
3:     $C := 0$, $\texttt{m} := 0$
4:     while $|Q_{\mathcal{A}}| > 0$ and $C \leq \texttt{cost}(\pi')$ do
5:         Remove first action $a$ in $Q_{\mathcal{A}}$
6:         $C := C + \texttt{cost}(a)$, $\texttt{m} := \texttt{m} + 1$
7:     return m

8: procedure MaxLayerPropositions($\mathcal{A}$, $\mathcal{P}_{\neg s_I}$,$\pi'$)
9:     $C := 0$, $\texttt{m} := 0$
10:    for $p \in \mathcal{P}_{\neg s_I}$ do
11:        $\texttt{cost}(p) := \min\{\texttt{cost}(a) : p \in \texttt{add}(a)\}$
12:    $Q_{\mathcal{P}} = (p_1, ...., p_{|\mathcal{P}_{\neg s_I}|})$, list of propositions ordered in increasing value of $\texttt{cost}(p)$
13:    while $|Q_{\mathcal{P}}| > 0$ and $C \leq \texttt{cost}(\pi')$ do
14:        Remove first proposition $p$ in $Q_{\mathcal{P}}$
15:        $C := C + \texttt{cost}(p)$, $\texttt{m} := \texttt{m} + 1$
16:    return m
```

actions in any minimal cost-optimal plan by assuming that each action in the plan adds at least one proposition that is not present in the previous state. The procedure first calculates the minimum cost of adding each proposition $p \in \mathcal{P}_{\neg s_I}$ by taking the minimum cost action that adds p (lines 10-11). Then, we order the propositions in increasing order of cost (line 12) and sum up the cost of each proposition until the total cost C is greater than $\texttt{cost}(\pi')$ (lines 13-15). Lastly, the total number of propositions in the sum corresponds to the maximum number of actions in any minimal cost-optimal plan.

Our implementation considers the minimum between the two estimations. We notice that MaxLayerPropositions tends to give a tighter bound when Π^+ has zero-cost actions, while MaxLayerActions is more accurate for tasks with action costs greater or equal to one. Both procedures compute the same value of m when all actions costs are the same.

5. Relaxed MDD-based Heuristic

We now present our relaxed MDD heuristic and prove its admissibility and consistency. Given any state s, we can create a relaxed MDD $\mathcal{M}$ for s using Algorithm 1 by updating the initial state $s_I := s$ and using a proper bound on the maximum number of layers m. Then, the relaxed MDD heuristic value for state s, $h_{\mathcal{M}}(s)$, is given by

$$h_{\mathcal{M}}(s) := \max\{\omega^{\downarrow}(\mathbf{t}, p) : p \in \mathcal{L}^{\mathcal{P}}\}. \tag{2}$$

Theorem 5.1. Consider a DFP task, a reachable state s, and a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ for s with $\mathcal{W} \geq 1$ constructed using Algorithm 1. Then, $h_{\mathcal{M}}(s)$ given by $h_{\mathcal{M}}(s) := \max\{\omega^{\downarrow}(\mathbf{t}, p) : p \in \mathcal{L}^{\mathcal{P}}\}$ is an admissible heuristic.

Proof. Propositions 4.2 and 4.3 guarantee that all minimal cost-optimal plans are represented by the MDD. Then, by Proposition 4.1, $h_{\mathcal{M}}(s) \leq \texttt{cost}(\pi)$ for any minimal cost-optimal plan represented by a path in the MDD. □

To avoid constructing a new MDD for each state in the search-space, we instead construct a relaxed MDD for the initial state s_I and update it during search. Given a state s and a sequence of actions that reach s from s_I, $\pi_s = (a_1, ..., a_k)$, we update the relaxed MDD $\mathcal{M}$ by removing any edge $e \in \mathcal{E}_i^{\mathcal{M}}$ $(i \in \{1, ..., k\})$ with $\mathtt{a}(e) \neq a_i$, i.e., the first k layers correspond to single path representing π_s. We then iteratively apply the top-down and bottom-up procedure (Algorithm 1) to update $\mathcal{M}$. In this case, the heuristic is

$$h_{\mathcal{M}}(s) := \begin{cases} \max\{\omega^{\downarrow}(\mathbf{t}, p) : p \in \mathcal{L}^{\mathcal{P}}\} - \mathtt{cost}(\pi_s), & \pi_s \in \mathcal{M}, \\ \infty, & \text{otherwise}, \end{cases} \tag{3}$$

where $\pi_s \in \mathcal{M}$ implies that there exists a path in the MDD where the k initial actions are given by π_s. Notice that since all minimal cost-optimal plans are encoded by $\mathcal{M}$, the condition $h_{\mathcal{M}}(s) = \infty$ holds only when there is no minimal cost-optimal plan from s_I that starts with the sequence of actions π_s. Thus, while the heuristic is inadmissible in these cases, it only gives value ∞ to sub-optimal states. In other words, $h_{\mathcal{M}}(s)$ given by (3) is a *global admissible* heuristic, as defined by Karpas and Domshlak (2012).

We choose this implementation because it speeds up the heuristic computation and it is compatible with our search procedure (see Section 10). In addition, the heuristic is consistent if the order of the propositions in Q remains the same when updating $\mathcal{M}$ for state s (Theorem 5.2).

Theorem 5.2. Consider a DFP task Π^+, a state s, and a relaxed MDD $\mathcal{M}$ with $\mathcal{W} \geq 1$ constructed using Algorithm 1. Then, $h_{\mathcal{M}}(s)$ given by (3) is consistent.

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

We know that

$$h_{\mathcal{M}}(s) = \max_{p \in \mathcal{L}^{\mathcal{P}}}\{\omega^{\downarrow}(\mathbf{t}_s, p)\} - \mathtt{cost}(\pi_s) \text{ and } h_{\mathcal{M}}(s') = \max_{p \in \mathcal{L}^{\mathcal{P}}}\{\omega^{\downarrow}(\mathbf{t}_{s'}, p)\} - \mathtt{cost}(\pi_s) - \mathtt{cost}(a).$$

Then,

$$h_{\mathcal{M}}(s) - \mathtt{cost}(a) - h_{\mathcal{M}}(s') = \max_{p \in \mathcal{L}^{\mathcal{P}}}\{\omega^{\downarrow}(\mathbf{t}_s, p)\} - \max_{p \in \mathcal{L}^{\mathcal{P}}}\{\omega^{\downarrow}(\mathbf{t}_{s'}, p)\} \leq 0,$$

and therefore $h_{\mathcal{M}}(s) \leq \mathtt{cost}(a) + h_{\mathcal{M}}(s')$. □

In this case $h_{\mathcal{M}}$ is consistent because the set of paths encoded in the successor state MDD, $\mathcal{M}_{s'}$, is a subset of the paths encoded by the current state MDD, $\mathcal{M}_s$. Thus, any other implementation that guarantees this condition will result in a consistent heuristic.

5.1 Relationship to Critical Path Heuristics

We now compare our heuristic with the critical path heuristic h^{max} (Bonet & Geffner, 1999), and relate the MDD graphical structure to the planning graph (Blum & Furst, 1997). The h^{max} heuristic computes the minimum cost to reach each proposition from the initial state. Specifically, consider $h(p)$ as the minimum cost to reach proposition $p \in \mathcal{P}$ and $h(a)$ as the minimum cost to use action $a \in \mathcal{A}$. These values can be computed recursively using the formula below and by setting $h(p) := 0$ for all $p \in s_I$, $h(p) := \infty$ for any $p \notin s_I$, and $h(a) := \infty$.

$$\begin{aligned} h(p) &:= \min\left\{h(p), \min\{h(a) : p \in \texttt{add}(a),\ a \in \mathcal{A}\}\right\} && \forall p \in \mathcal{P}, \\ h(a) &:= \texttt{cost}(a) + \max\{h(q) : q \in \texttt{pre}(a)\} && \forall a \in \mathcal{A}. \end{aligned}$$

The heuristic is defined as $h^{max} := \max\{h(p) : p \in \mathcal{G}\}$. Notice that our MDD top-down node and edge cost computations (Section 6.2) resemble the $h(p)$ and $h(a)$ recursions, respectively. Lemma 5.1 and Theorem 5.3 show that in fact our MDD heuristic dominates the h^{max} heuristic for any width $\mathcal{W} \geq 1$.

Lemma 5.1. Consider a DFP task Π^+, a reachable state s, and a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ for s with $\mathcal{W} \geq 1$. Then,

$$h(p) \leq \omega^{\downarrow}(u,p) \qquad \forall u \in \mathcal{N}^{\mathcal{M}},\ p \in \alpha^{\downarrow}_{\texttt{S}}(u). \tag{4}$$

Proof. We prove (4) by induction over the layers of $\mathcal{M}$. By construction, (4) holds for $\mathcal{N}^{\mathcal{M}}_1 = \{\mathbf{r}\}$. Now consider that (4) is true for all nodes $u \in \mathcal{N}^{\mathcal{M}}_i$ and $p \in \alpha^{\downarrow}_{\texttt{S}}(u)$ ($i \geq 1$). Consider any node $v \in \mathcal{N}^{\mathcal{M}}_{i+1}$ and a proposition $p \in \alpha^{\downarrow}_{\texttt{S}}(v)$. By construction, there exists an edge $e \in \delta_{\texttt{in}}(v)$ such that $\texttt{c}_p(e) = \omega^{\downarrow}(v,p)$. Consider action $a = \texttt{a}(e)$ and node $u = \texttt{s}(e) \in \mathcal{N}^{\mathcal{M}}_i$. There are two cases, either $p \in \texttt{add}(a)$ or not. If $p \in \texttt{add}(a)$, then

$$\begin{aligned} \omega^{\downarrow}(v,p) = \texttt{c}_p(e) &= \texttt{cost}(a) + \max\{\omega^{\downarrow}(u,q) : q \in \texttt{pre}(a)\} \\ &\geq \texttt{cost}(a) + \max\{h(q) : q \in \texttt{pre}(a)\} \geq h(p). \end{aligned}$$

If $p \notin \texttt{add}(a)$, we necessarily have that $p \in \alpha^{\downarrow}_{\texttt{S}}(u)$. Since $u \in \mathcal{N}^{\mathcal{M}}_i$, we have $h(p) \leq \omega^{\downarrow}(u,p)$. Then it follows that $h(p) + \texttt{cost}(a) \leq \omega^{\downarrow}(u,p) + \texttt{cost}(a) \leq \texttt{c}_p(e) = \omega^{\downarrow}(v,p)$. □

Theorem 5.3. Consider a classical planning task Π^+, a reachable state s, and a relaxed MDD $\mathcal{M} = (\mathcal{N}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}})$ for s with $\mathcal{W} \geq 1$. Then, $h_{\mathcal{M}}(s) \geq h^{max}(s)$.

Proof. From Lemma 5.1 we have that $h(p) \leq \omega^{\downarrow}(\mathbf{t},p)$. Then, $h^{max}(s) = \max\{h(p) : p \in \mathcal{G}\} \leq \max\{\omega^{\downarrow}(\mathbf{t},p) : p \in \mathcal{L}^{\mathcal{P}}\} = h_{\mathcal{M}}(s)$. □

We can interpreted our relaxed MDD heuristic as a type of critical path heuristic over the relaxed MDD graph instead of the planning graph (Blum & Furst, 1997). In fact, the planning graph can be seen as a relaxed MDD with $\mathcal{W} = 1$ where each node corresponds to a propositional layer in the planning graph and each edge layer to an action layer. As such, a relaxed MDD is a generalization of the planning graph since a relaxed MDD with $\mathcal{W} > 1$ considers more than one node per layer.

6. Relaxed BDD Encoding and Construction Procedure

We now investigate an alternative approach to compute admissible heuristics based on DDs: we use a binary decision diagram (BDD) to encode the sequential relaxation of the DFP. Previous work has empirically shown that the sequential relaxation is an accurate approximation to the DFP task in most IPC domains (Imai & Fukunaga, 2015). Thus, a BDD encoding of the sequential relaxation can potentially compute informative heuristics using a smaller graphical structure than our relaxed MDD (see Section 8).

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 cost-optimal action set satisfying Definition 2.1, where each variable indicates whether or not an action is included in a `sr`-plan. 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 `sr`-plan.

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

We associate an action $a \in \mathcal{A}$ with each layer $\mathcal{E}^{\mathcal{B}}_i$, $\mathtt{a}(\mathcal{E}^{\mathcal{B}}_i) = a$, such that no two layers are associated with the same action. Every edge $e \in \mathcal{E}^{\mathcal{B}}_i$ has (i) a label $\mathtt{l}(e) \in \{0, 1\}$ that represents if its associated action $\mathtt{a}(e) = \mathtt{a}(\mathcal{E}^{\mathcal{B}}_i)$ is selected to be in an `sr`-plan or not, and (ii) a cost $\mathtt{c}(e)$ derived from including (if $\mathtt{l}(e) = 1$) or excluding (if $\mathtt{l}(e) = 0$) the action to the `sr`-plan, i.e., $\mathtt{c}(e) := \mathtt{l}(e) \cdot \mathtt{cost}(\mathtt{a}(e))$. A node $u \in \mathcal{N}^{\mathcal{B}}$ has at most two edges emanating from it, each with a distinct label. Thus, an $\mathbf{r} - \mathbf{t}$ path $\rho := (e_1, ..., e_{\mathtt{m}}) \in \mathcal{B}$ 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.

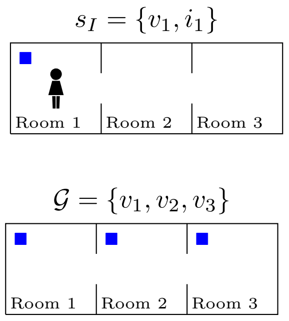
We force $\mathcal{B}$ to encode only `sr`-plans by keeping track of the set of propositions that are achieved and needed in each path. Given an $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$, the set of propositions achieved and needed by ρ are given, respectively, by:

$$\alpha^{\downarrow}(\rho) := s_I \cup \bigcup_{e \in \rho : \mathtt{l}(e) = 1} \mathtt{add}(\mathtt{a}(e)),$$
$$\eta^{\downarrow}(\rho) := \mathcal{G} \cup \bigcup_{e \in \rho : \mathtt{l}(e) = 1} \mathtt{pre}(\mathtt{a}(e)).$$

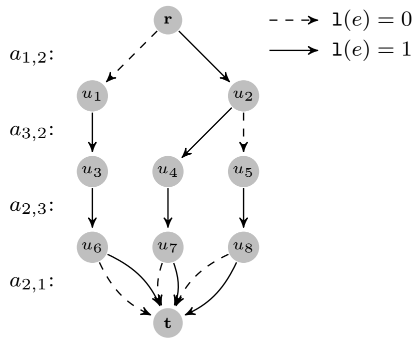
A BDD $\mathcal{B}$ encodes `sr`-plans if each $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$ satisfies the conditions in Definition 2.1, i.e., $\mathcal{G} \subseteq \alpha^{\downarrow}(\rho)$ and $\eta^{\downarrow}(\rho) \subseteq \alpha^{\downarrow}(\rho)$. We say that a BDD is *exact* if every 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 cost-optimal `sr`-plan.

Example 6.1. From now on we will consider a smaller task of the visit-all domain with three rooms, Π^+_3, since it results in a shorter BDD. The set of actions is $\mathcal{A} = \{a_{1,2}, a_{2,1}, a_{2,3}, a_{3,2}\}$ and the set of propositions is $\mathcal{P} = \{i_1, v_1, i_2, v_2, i_3, v_3\}$. The meaning of each proposition and action is the same as in Example 2.1. Figure 4a depicts the initial state and goal conditions.

Figure 4b illustrates an exact BDD $\mathcal{B}$ for Π^+_3. The dashed edges represent zero-edges (i.e., $\mathtt{l}(e) = 0$) and the solid edges one-edges (i.e., $\mathtt{l}(e) = 1$). On the left-hand-side are the actions associated with each layer. Notice that there is a one-to-one association between `sr`-plans and paths.



(a) Visit-all with 3 rooms.



(b) Exact BDD.

Figure 4: Three room running example and corresponding exact BDD.

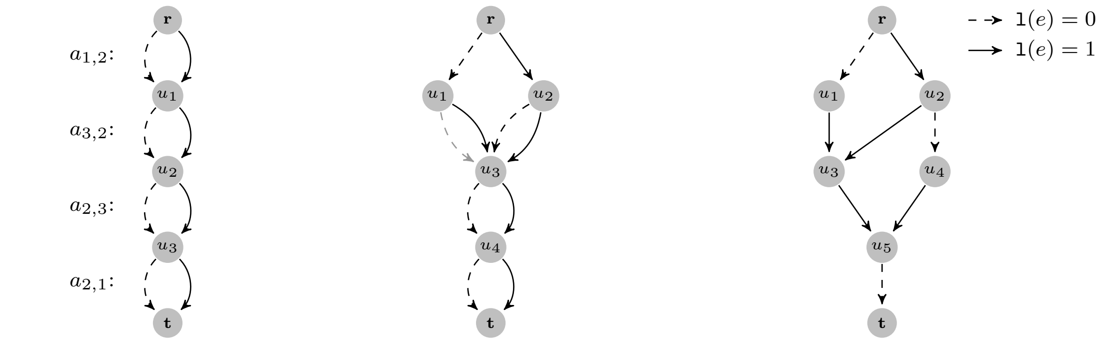
As with our MDD presented above, the size of an exact BDD (i.e., the number of nodes) grows exponentially with the size of the planning task Π^+ (i.e., the number of actions and propositions). We overcome this problem by constructing a *relaxed* BDD: a limited size BDD that over-approximates the set of $\mathtt{sr}$-plans. In particular, we show in Theorem 7.1 that the shortest path of a relaxed BDD is an admissible heuristic for Π^+.

6.1 Relaxed BDD Construction Algorithm

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

We use the same iterative construction procedure introduces in Algorithm 1 to create relaxed BDDs. The BDD construction procedure only differs on the low-level method implementations, specifically the initial BDD (WidthOneBDD), the update nodes procedures (UpdateBDDNodesTop and UpdateBDDNodesBottom), the filtering rules (FilterBDDEdges), and the splitting nodes algorithm (SplitBDDNodes). In particular, the WidthOneBDD procedure receives a sequence of pair-wise distinct action labels $\Lambda = (\lambda_1, ..., \lambda_{\mathtt{m}})$ and assigns each action to a layer such that $a_{\lambda_i} = \mathtt{a}(\mathcal{E}_i^{\mathcal{B}})$, for all $i \in \{1, ..., \mathtt{m}\}$. It then constructs a width-one BDD by creating one node $u \in \mathcal{N}_i^{\mathcal{B}}$ in each node layer $i \in \{1, ..., \mathtt{m}+1\}$ and two edges $e, e' \in \mathcal{E}_i^{\mathcal{B}}$ with different labels (i.e., $\mathtt{l}(e) \neq \mathtt{l}(e')$) in each edge layer $i \in \{1, ..., \mathtt{m}\}$.

Example 6.2. Consider our running example task Π_3^+ with three rooms. Figure 5 illustrates some of the steps of the BDD construction procedure. Figure 5a depicts the width-one BDD created by the WidthOneBDD procedure. Figure 5b illustrates the SplitBDDNodes procedure over $\mathcal{N}_2^{\mathcal{B}}$ and the FilterBDDEdges procedure for $\mathcal{E}_2^{\mathcal{B}}$ by a gray edge (eliminated by Rule B2, Section 6.3). Lastly, Figure 5c shows the resulting relaxed BDD.



(a) Width-one BDD. (b) BDD after one split. (c) Relaxed BDD.

Figure 5: BDD construction with $\mathcal{W} = 2$.

6.2 BDD Node Representation

With the purpose of identifying invalid $\mathtt{sr}$-plans and deciding how to split a node, each node $u \in \mathcal{N}^{\mathcal{B}}$ stores information about the achieved and needed propositions by all paths passing through u. As in the MDD encoding, the information is aggregated both for the $\mathbf{r} - u$ and the $u - \mathbf{t}$ partial paths. Once again, consider $\delta_{\mathtt{in}}(u)$ and $\delta_{\mathtt{out}}(u)$ as the set of incoming and outgoing edges of a node $u \in \mathcal{N}^{\mathcal{B}}$, respectively, and let $\mathtt{s}(e) = u$ and $\mathtt{t}(e) = v$ represent the source and target node of an edge $e = (u, v) \in \mathcal{E}^{\mathcal{M}}$, respectively. To simplify the exposition, we will consider that all zero-edges ($\mathtt{l}(e) = 0$) have $\mathtt{add}(\mathsf{a}(e)) = \mathtt{pre}(\mathsf{a}(e)) = \emptyset$.

Procedure UpdateBDDNodesTop updates the top-down information of all nodes. For each node $u \in \mathcal{N}^{\mathcal{B}}$, we store the propositions that are achieved by *all* $\mathbf{r} - u$ paths, $\alpha^{\downarrow}_{\mathtt{A}}(u)$, and the propositions achieved by *some* $\mathbf{r} - u$ path, $\alpha^{\downarrow}_{\mathtt{S}}(u)$. These sets have the same meaning as in the MDD case and are computed in the same fashion (see Section 4.2). In addition, each node $u \in \mathcal{N}^{\mathcal{B}}$ stores the set of propositions that are needed by all and at least one $\mathbf{r} - u$ paths, $\eta^{\downarrow}_{\mathtt{A}}(u)$ and $\eta^{\downarrow}_{\mathtt{S}}(u)$, respectively. At the root node $\eta^{\downarrow}_{\mathtt{A}}(\mathbf{r}) := \eta^{\downarrow}_{\mathtt{S}}(\mathbf{r}) := \mathcal{L}^{\mathcal{P}}$, and for any other node $u \in \mathcal{N}^{\mathcal{B}}$ we have that

$$\eta^{\downarrow}_{\mathtt{A}}(u) := \bigcap_{e \in \delta_{\mathtt{in}}(u)} \left(\eta^{\downarrow}_{\mathtt{A}}(\mathtt{s}(e)) \cup \mathtt{pre}(\mathsf{a}(e)) \right),$$
$$\eta^{\downarrow}_{\mathtt{S}}(u) := \bigcup_{e \in \delta_{\mathtt{in}}(u)} \left(\eta^{\downarrow}_{\mathtt{S}}(\mathtt{s}(e)) \cup \mathtt{pre}(\mathsf{a}(e)) \right).$$

For the bottom-up information of $u \in \mathcal{N}^{\mathcal{B}}$, sets $\alpha^{\uparrow}_{\mathtt{A}}(u)$ and $\eta^{\uparrow}_{\mathtt{A}}(u)$ correspond to the propositions achieved and needed by all $u - \mathbf{t}$ paths, respectively. Similarly, sets $\alpha^{\uparrow}_{\mathtt{S}}(u)$ and $\eta^{\uparrow}_{\mathtt{S}}(u)$ represent the propositions achieved and needed by some $u - \mathbf{t}$ path, respectively. Starting at $\mathbf{t}$ with $\alpha^{\uparrow}_{\mathtt{A}}(\mathbf{t}) := \alpha^{\uparrow}_{\mathtt{S}}(\mathbf{t}) := \eta^{\uparrow}_{\mathtt{A}}(\mathbf{t}) := \eta^{\uparrow}_{\mathtt{S}}(\mathbf{t}) := \emptyset$, UpdateBDDNodesBottom instantiates $\alpha^{\uparrow}_{\mathtt{S}}(u)$ and $\eta^{\uparrow}_{\mathtt{S}}(u)$ for each node $u \in \mathcal{N}^{\mathcal{B}}$ as described in Section 4.2, and sets $\alpha^{\uparrow}_{\mathtt{A}}(u)$ and $\eta^{\uparrow}_{\mathtt{A}}(u)$ as

$$\alpha^{\uparrow}_{\mathtt{A}}(u) := \bigcap_{e \in \delta_{\mathtt{out}}(u)} \left(\alpha^{\uparrow}_{\mathtt{A}}(\mathtt{t}(e)) \cup \mathtt{add}(\mathtt{a}(e)) \right),$$
$$\eta^{\uparrow}_{\mathtt{A}}(u) := \bigcap_{e \in \delta_{\mathtt{out}}(u)} \left(\eta^{\uparrow}_{\mathtt{A}}(\mathtt{t}(e)) \cup \mathtt{pre}(\mathtt{a}(e)) \right).$$

Notice that the BDD keeps track of more propositional sets than the MDD. The latter can avoid storing top-down information of the needed propositions because it represents the sequencing aspect of the problem. In contrast, as the BDD only encodes sr-plans, we require the propositions in both directions to identify invalid paths.

In addition, each node $u \in \mathcal{N}^{\mathcal{B}}$ 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 $u \in \mathcal{N}^{\mathcal{B}}$ is

$$\omega^{\downarrow}(u) := \min_{e \in \delta_{\mathtt{in}}(u)} \left\{ \omega^{\downarrow}(\mathtt{s}(e)) + \mathtt{c}(e) \right\},$$
$$\omega^{\uparrow}(u) := \min_{e \in \delta_{\mathtt{out}}(u)} \left\{ \omega^{\uparrow}(\mathtt{t}(e)) + \mathtt{c}(e) \right\}.$$

The shortest path information is used both for the BDD heuristic computation (Section 7) and to identify and eliminate sub-optimal sr-plans (Section 6.3).

Example 6.3. Consider node u_3 in Figure 5c for our running example. For the top-down information we have $\alpha^{\downarrow}_{\mathtt{A}}(u_3) = \alpha^{\downarrow}_{\mathtt{S}}(u_3) = \{i_1, i_2, v_1, v_2\}$, $\eta^{\downarrow}_{\mathtt{A}}(u_3) = \{v_1, v_2, v_3, i_3\}$ and $\eta^{\downarrow}_{\mathtt{S}}(u_3) = \eta^{\downarrow}_{\mathtt{A}}(u_3) \cup \{i_1\}$. For the bottom up we have $\alpha^{\uparrow}_{\mathtt{A}}(u_3) = \alpha^{\uparrow}_{\mathtt{S}}(u_3) = \{i_3, v_3\}$ and $\eta^{\uparrow}_{\mathtt{A}}(u_3) = \eta^{\uparrow}_{\mathtt{S}}(u_3) = \{i_2\}$.

6.3 BDD 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 $e \in \mathcal{E}^{\mathcal{B}}$ violates a rule, then all paths passing through e are invalid sr-plans, and edge e can be removed.

Rule B1. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{B}}$ with action $\mathtt{a}(e) = a$ and label $\mathtt{l}(e) = 1$. Every precondition of a has to be achieved by at least one $\mathbf{r} - \mathbf{t}$ path that includes edge e.

$$\mathtt{pre}(a) \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u) \cup \alpha^{\uparrow}_{\mathtt{S}}(v).$$

Rule B2. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{B}}$ with $\mathtt{a}(e) = a$. Then, each proposition needed by all $\mathbf{r} - \mathbf{t}$ paths traversing e needs to be achieved.

$$\eta^{\downarrow}_{\mathtt{A}}(u) \cup \eta^{\uparrow}_{\mathtt{A}}(v) \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u) \cup \alpha^{\uparrow}_{S}(v), \qquad \text{if } \mathtt{l}(e) = 0,$$
$$\eta^{\downarrow}_{\mathtt{A}}(u) \cup \eta^{\uparrow}_{\mathtt{A}}(v) \subseteq \alpha^{\downarrow}_{\mathtt{S}}(u) \cup \mathtt{add}(a) \cup \alpha^{\uparrow}_{S}(v), \qquad \text{if } \mathtt{l}(e) = 1.$$

Notice that rules B1 and B2 are similar to rules M1 and M2 (see Section 4.3). Both sets of rules make sure that the actions are applicable and that the needed propositions are covered. However, rules B1 and B2 are slightly more complicated since the BDD considers the sequential relaxation of the problem. Proposition 6.1 shows that rules B1 and B2 are necessary conditions for any sr-plan path.

Proposition 6.1. Consider a relaxed BDD $\mathcal{B} = (\mathcal{N}^{\mathcal{B}}, \mathcal{E}^{\mathcal{B}})$ for a DFP task Π^+. Rules B1 and B2 are necessary conditions for any $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$ to be an $\texttt{sr}$-plan.

Proof. Consider a path $\rho = (e_1, \ldots, e_{\mathtt{m}}) \in \mathcal{B}$ that corresponds to an $\texttt{sr}$-plan and any edge $e = (u, v) \in \rho$. From Section 6.2, we know that $\bigcup_{e' \in \rho \setminus \{e\}} \texttt{add}(\texttt{a}(e')) \subseteq \alpha^{\downarrow}_{\texttt{S}}(u) \cup \alpha^{\uparrow}_{\texttt{S}}(v)$, and $\eta^{\downarrow}_{\texttt{A}}(u) \cup \eta^{\uparrow}_{\texttt{A}}(v) \subseteq \bigcup_{e' \in \rho \setminus \{e\}} \texttt{pre}(\texttt{a}(e')) \cup \mathcal{L}^{\mathcal{P}}$. Since $\texttt{pre}(a) \cap \texttt{add}(a) = \emptyset$ for all actions $a \in \mathcal{A}$, rules B1 and B2 are satisfied by any edge $e \in \rho$. $\square$

In addition, we develop two additional filtering rules to identify sub-optimal plans. These rules are analogous to M3 and M4 (see Section 4.3).

Rule B3. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{B}}$ with $\texttt{a}(e) = a$ and $\texttt{l}(e) = 1$. Action a adds at least one proposition $p \notin s_I$ that is needed by some path traversing e.

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

Rule B4. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{B}}$ with $\texttt{a}(e) = a$ and a plan $\pi' \in \Pi^+$. The minimum-cost path traversing e has a cost less than or equal to $\texttt{cost}(\pi')$.

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

All minimal cost-optimal plans satisfy rules B3 and B4, and, as a consequence, the rules are valid for any minimal cost-optimal $\texttt{sr}$-plan. Rule B3 avoids unnecessary actions and rule B4 removes $\texttt{sr}$-plans with higher cost than the best plan found so far. Notice that Rule B3 can be strengthened to remove all $\texttt{sr}$-plans with redundant actions as shown in rule B3-A. However, this rule can potentially remove some minimal cost-optimal plans. We decided to use rule B3 to guarantee that all minimal cost-optimal plans are represented in the BDD.

Rule B3-A. Consider an edge $e = (u, v) \in \mathcal{E}^{\mathcal{B}}$ with $\texttt{a}(e) = a$ and $\texttt{l}(e) = 1$. Action a adds at least one proposition $p \notin s_I$ that is needed by some path traversing e.

$$\left(\texttt{add}(a) \setminus (\alpha^{\downarrow}_{\texttt{A}}(u) \cup \alpha^{\uparrow}_{\texttt{A}}(v))\right) \cap \left(\eta^{\downarrow}_{\texttt{S}}(u) \cup \eta^{\uparrow}_{\texttt{S}}(v)\right) \neq \emptyset.$$

The FilterBDDEdges procedure (Algorithm 5) 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 keeps track of changes in the BDD.

Algorithm 5 Relaxed BDD Filtering Edges Procedure

```
1: procedure FilterBDDEdges($\mathcal{E}^{\mathcal{B}}_i$)
2:     for $e \in \mathcal{E}^{\mathcal{B}}_i$ do
3:         if $e$ violates any of rule B1 to rule B4 then
4:             Eliminate edge $e$ from $\mathcal{E}^{\mathcal{B}}_i$. $\mathcal{B}$ has been modified
```

6.4 BDD Splitting Algorithm

The SplitBDDNodes 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 the DFP characteristics to create relaxed BDDs with tight worst cases on the maximum width needed per layer. We start by defining the exact information of a node.

Definition 6.1. Consider a node $u \in \mathcal{N}^{\mathcal{B}}$ and a proposition $p \in \mathcal{P}$.

(i) Proposition $p \in \mathcal{P}$ is α-exact in node u if all $\mathbf{r} - u$ paths add p or none do, i.e., $p \in \alpha^{\downarrow}_{\mathtt{A}}(u)$ or $p \notin \alpha^{\downarrow}_{\mathtt{S}}(u)$, respectively.

(ii) Proposition $p \in \mathcal{P}$ is η-exact in node u if either all $\mathbf{r} - u$ paths require p or none do, i.e., $p \in \eta^{\downarrow}_{\mathtt{A}}(u)$ or $p \notin \eta^{\downarrow}_{\mathtt{S}}(u)$, respectively.

Algorithm 6 Relaxed BDD Split Nodes Procedures

```
1: procedure SplitBDDNodes($\mathcal{N}_i^{\mathcal{B}}$, $\mathcal{W}$)
2:     Initialize $Q$ with all the propositions in $\mathcal{P}_{\neg s_I}$
3:     while $|Q| > 0$ and $|\mathcal{N}_i^{\mathcal{B}}| < \mathcal{W}$ do
4:         Remove first proposition $p$ in $Q$
5:         if $i \leq \gamma(p) + 1$ then
6:             for $u \in \mathcal{N}_i^{\mathcal{B}}$ do
7:                 if $p \in \alpha^{\downarrow}_{\mathtt{S}}(u)$ and $p \notin \alpha^{\downarrow}_{\mathtt{A}}(u)$ then
8:                     SplitBDDNodeAchieved($u$, $p$)
9:                 if $|\mathcal{N}_i^{\mathcal{B}}| = \mathcal{W}$ then return
10:            for $u \in \mathcal{N}_i^{\mathcal{B}}$ do
11:                if $p \in \eta^{\downarrow}_{\mathtt{S}}(u)$ and $p \notin \eta^{\downarrow}_{\mathtt{A}}(u)$ and $p \notin \alpha^{\downarrow}_{\mathtt{A}}(u)$ then
12:                    SplitBDDNodeNeeded($u$, $p$)
13:                if $|\mathcal{N}_i^{\mathcal{B}}| = \mathcal{W}$ then return
```

Algorithm 6 illustrates the SplitBDDNodes procedure. The algorithm receives a node layer and the width limit, $\mathcal{W}$. The procedure iterates over a priority queue of propositions Q and splits nodes such that for each $p \in Q$, all nodes are α-exact and η-exact or the width limit is reached. As in the MDD case (Section 4.4), goal propositions have higher priority, followed by propositional landmarks, and then the remaining propositions. Algorithm 6 avoids splitting nodes with respect to proposition p after layer index $\gamma(p) + 1$ (line 5), a valid condition explained in Proposition 6.3.

Algorithm 7 shows how to split any node $u \in \mathcal{N}^{\mathcal{B}}$ with respect to a proposition $p \in \mathcal{P}$. The procedure iterates over the incoming edges of u and redirects the edges to a new node u' accordingly. If p is α-exact (respectively, η-exact) in all nodes in the previous layer, our splitting procedure guarantees that p will be α-exact in u (respectively, η-exact).

Proposition 6.2. Consider a BDD $\mathcal{B} = (\mathcal{N}^{\mathcal{B}}, \mathcal{E}^{\mathcal{B}})$ such that for each $p \in \mathcal{P}$ and node $u \in \mathcal{N}^{\mathcal{B}}$, p is α-exact in u and p is η-exact in u when $p \notin \alpha^{\downarrow}_{\mathtt{A}}(u)$. Then, rules B1 and B2 are sufficient to remove all invalid sr-plan paths in $\mathcal{B}$.

Algorithm 7 Relaxed BDD Split Single Node Procedures

```
1: procedure SplitBDDNodeAchieved($u$, $p$ )
2:     Create a new node $u'$ and update $\mathcal{N}_i^{\mathcal{B}} = \mathcal{N}_i^{\mathcal{B}} \cup \{u'\}$
3:     $\delta_{\texttt{achieved}} := \{e \in \delta_{\texttt{in}}(u) : \ p \in \alpha^{\downarrow}_{\texttt{A}}(\texttt{s}(e))$ or $p \in \texttt{add}(\texttt{a}(e))\}$
4:     Redirect edges: $\delta_{\texttt{in}}(u') := \delta_{\texttt{achieved}}$ and $\delta_{\texttt{in}}(u) := \delta_{\texttt{in}}(u) \setminus \delta_{\texttt{achieved}}$
5:     if $\delta_{\texttt{in}}(u') \neq \emptyset$ then Duplicate outgoing edges from $u$ to $u'$

6: procedure SplitBDDNodeNeeded($u$, $p$ )
7:     Create a new node $u'$ and update $\mathcal{N}_i^{\mathcal{B}} = \mathcal{N}_i^{\mathcal{B}} \cup \{u'\}$
8:     $\delta_{\texttt{needed}} := \{e \in \delta_{\texttt{in}}(u) : \ p \in \eta^{\downarrow}_{\texttt{A}}(\texttt{s}(e))$ or $p \in \texttt{pre}(\texttt{a}(e))\}$
9:     Redirect edges: $\delta_{\texttt{in}}(u') := \delta_{\texttt{needed}}$ and $\delta_{\texttt{in}}(u) := \delta_{\texttt{in}}(u) \setminus \delta_{\texttt{needed}}$
10:    if $\delta_{\texttt{in}}(u') \neq \emptyset$ then Duplicate outgoing edges from $u$ to $u'$
```

Proof. Consider a path $\rho \in \mathcal{B}$ and a proposition $p \in \mathcal{P}_{\neg s_I}$ such that either (i) there exists an action $a \in \rho$ with $p \in \texttt{pre}(a)$ or (ii) $p \in \mathcal{L}^{\mathcal{P}}$ but for all $a' \in \rho$, $p \notin \texttt{add}(a')$. Take the last edge $e \in \rho$, i.e., $\texttt{s}(e) = u \in \mathcal{N}_{\texttt{m}}^{\mathcal{B}}$ and $\texttt{t}(e) = \texttt{t}$. Since ρ is an invalid $\texttt{sr}$-plan, $p \notin \alpha^{\downarrow}_{\texttt{A}}(u)$ and $p \in \eta^{\downarrow}_{\texttt{A}}(u) \cup \texttt{pre}(\texttt{a}(e))$. Moreover, $p \notin \alpha^{\downarrow}_{\texttt{S}}(u)$ since p is α-exact in u. Since $\alpha^{\uparrow}_{\texttt{S}}(\texttt{t}) = \emptyset$ (Section 6.2), either rule B1 or B2 will eliminate edge e and, therefore, remove ρ from $\mathcal{B}$. □

Proposition 6.2 implies that for each proposition $p \in \mathcal{P}$ we need at most three nodes in each layer $\mathcal{N}_i^{\mathcal{B}}$, i.e., a node $u \in \mathcal{N}_i^{\mathcal{B}}$ where $p \in \alpha^{\downarrow}_{\texttt{A}}(u)$, a node $u' \in \mathcal{N}_i^{\mathcal{B}}$ where $p \notin \alpha^{\downarrow}_{\texttt{A}}(u')$ and $p \in \eta^{\downarrow}_{\texttt{A}}(u')$, and a node $u'' \in \mathcal{N}_i^{\mathcal{B}}$ where $p \notin \alpha^{\downarrow}_{\texttt{A}}(u'')$ and $p \notin \eta^{\downarrow}_{\texttt{A}}(u'')$. Since for all $p \in s_I$ and $u \in \mathcal{N}^{\mathcal{B}}$, $p \in \alpha^{\downarrow}_{\texttt{A}}(u)$ (Section 6.2), there is no need to split nodes with respect to propositions in the initial state. Similarly, for all propositions $p \in \mathcal{L}^{\mathcal{P}}$ and nodes $u \in \mathcal{N}^{\mathcal{B}}$, $p \in \eta^{\downarrow}_{\texttt{A}}(u)$ (see Section 6.2), so each propositional landmark needs at most two nodes in each layer. Then, a conservative estimate on the maximum width needed to construct an exact BDD is $O(3^{|\mathcal{P}_{\neg s_I, \neg \mathcal{L}^{\mathcal{P}}}|} \cdot 2^{|\mathcal{L}^{\mathcal{P}}{}_{\neg s_I}|})$, where $\mathcal{L}^{\mathcal{P}}{}_{\neg s_I} = \mathcal{L}^{\mathcal{P}} \setminus s_I$ (i.e., all propositional landmarks omitted in the initial state) and $\mathcal{P}_{\neg s_I, \neg \mathcal{L}^{\mathcal{P}}} = (\mathcal{P} \setminus \mathcal{L}^{\mathcal{P}}) \setminus s_I$ (i.e., all propositions omitted in the initial state that are not landmarks).

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 edge layer index at which an action either adds or requires p, i.e., for $i = \gamma(p)$, $p \in \texttt{add}(\texttt{a}(\mathcal{E}_i^{\mathcal{B}})) \cup \texttt{pre}(\texttt{a}(\mathcal{E}_i^{\mathcal{B}}))$ and for all $j > \gamma(p)$, $p \notin \texttt{add}(\texttt{a}(\mathcal{E}_j^{\mathcal{B}})) \cup \texttt{pre}(\texttt{a}(\mathcal{E}_j^{\mathcal{B}}))$.

Proposition 6.3. Consider a BDD $\mathcal{B} = (\mathcal{N}^{\mathcal{B}}, \mathcal{E}^{\mathcal{B}})$ such that for each $p \in \mathcal{P}$ and node $u \in \mathcal{N}_i^{\mathcal{B}}$, with $i \leq \gamma(p) + 1$, p is α-exact in u and p is η-exact in u when $p \notin \alpha^{\downarrow}_{\texttt{A}}(u)$. Then, rules B1 and B2 are sufficient to remove all invalid $\texttt{sr}$-plan paths in $\mathcal{B}$.

Proof. Consider a path $\rho \in \mathcal{B}$ with a proposition $p \in \mathcal{P}_{\neg s_I}$ such that $p \in \mathcal{L}^{\mathcal{P}}$ or for some action $a \in \rho$, $p \in \texttt{pre}(a)$ 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., $u \in \mathcal{N}_{\gamma(p)}^{\mathcal{B}}$ and $v \in \mathcal{N}_{\gamma(p)+1}^{\mathcal{B}}$. By definition of last layer, $p \notin \alpha^{\uparrow}_{\texttt{S}}(v)$ and $p \notin \eta^{\uparrow}_{\texttt{S}}(v)$. By hypothesis over ρ, $p \notin \alpha^{\downarrow}_{\texttt{A}}(u)$ (and, $p \notin \alpha^{\downarrow}_{\texttt{S}}(u)$) and $p \in \eta^{\downarrow}_{\texttt{A}}(u) \cup \texttt{pre}(\texttt{a}(e))$. Then, either rule B1 or B2 will remove edge e and, hence, the invalid $\texttt{sr}$-plan path ρ. □

Notice that Algorithm 6 uses Proposition 6.3 to avoid splitting nodes with respect to a proposition $p \in Q$ when the current layer is greater than $\gamma(p)$ (line 5), and avoids splitting nodes $u \in \mathcal{N}^{\mathcal{B}}$ if $p \in \alpha^{\downarrow}_{\mathtt{A}}(u)$ (line 11).

6.5 Maximum BDD Width and Action Ordering

Given the BDD construction procedure in Section 6.1, 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 edge layer where p is either added or needed, i.e., $\phi(p) := \min\{i : p \in \mathtt{add}(\mathsf{a}(\mathcal{E}^{\mathcal{B}}_i)) \cup \mathtt{pre}(\mathsf{a}(\mathcal{E}^{\mathcal{B}}_i)), i \in \{1, ..., \mathtt{m}\}\}$. We define $\psi(i)$ as the set of propositions that need to be considered for splitting in node layer $\mathcal{N}^{\mathcal{B}}_i$, i.e., $\psi(i) := \{p \in \mathcal{P}_{\neg s_I} : \phi(p) + 1 \leq i \leq \gamma(p) + 1\}$. In particular, let $\psi_{\mathcal{L}^{\mathcal{P}}}(i) := \psi(i) \cap \mathcal{L}^{\mathcal{P}}$ be the set of propositional landmarks that need splitting in layer $\mathcal{N}^{\mathcal{B}}_i$, and $\psi_{\mathcal{P}}(i) := \psi(i) \setminus \mathcal{L}^{\mathcal{P}}$ be the set of non-propositional landmarks that need splitting in layer $\mathcal{N}^{\mathcal{B}}_i$.

Corollary 6.1. Consider a DFP task Π^+ and an exact BDD $\mathcal{B}$ constructed as described in Section 6.1. The maximum width of layer $\mathcal{N}^{\mathcal{B}}_i$ is $O(2^{|\psi_{\mathcal{L}^{\mathcal{P}}}(i)|} \cdot 3^{|\psi_{\mathcal{P}}(i)|})$. Then, an upper bound on the maximum width for $\mathcal{B}$ is given by $O(\max_{i \in \{1,...,\mathtt{m}\}}\{2^{|\psi_{\mathcal{L}^{\mathcal{P}}}(i)|} \cdot 3^{|\psi_{\mathcal{P}}(i)|}\})$.

Proof. Follows directly from Proposition 6.3. □

Notice that $\psi(i)$ depends on the action ordering Λ used to assign actions to layers (WidthOneBDD). 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 $\max_{i \in \{1,...,\mathtt{m}\}}\{2^{|\psi_{\mathcal{L}^{\mathcal{P}}}(i)|} \cdot 3^{|\psi_{\mathcal{P}}(i)|}\}$ 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, & Hooker, 2011; Bergman, Cire, van Hoeve, & Hooker, 2012). We develop a simple action ordering heuristic that takes advantage of the following proposition.

Proposition 6.4. Consider a DFP task Π^+ and a relaxed BDD $\mathcal{B}$ constructed as described in Section 6.1 with action ordering Λ. Assume that for a given $p \in \mathcal{P}_{\neg s_I, \neg \mathcal{L}^{\mathcal{P}}}$ 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 $\mathcal{E}^{\mathcal{B}}_i$ such that $a = \mathsf{a}(\mathcal{E}^{\mathcal{B}}_i)$ requires p. Take a node $u \in \mathcal{N}^{\mathcal{B}}_j$ $(j > i)$. If $p \in \alpha^{\downarrow}_{\mathtt{A}}(u)$, u does not need to be η-exact (Proposition 6.3). If $p \notin \alpha^{\downarrow}_{\mathtt{S}}(u)$, rules B1 and B2 eliminate edges that require p, so no split is needed. □

Our BDDActionOrdering procedure (Algorithm 8) receives the set of actions and a priority queue of propositions Q. In our implementation, we use the same propositional priority queue used in the SplitBDDNodes procedure (Algorithm 6). 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 (lines 4-6) and then actions $a \in \mathcal{A} \setminus \Lambda$ that require p (lines 8-10). Notice that when iterating over a propositional landmark p, we omit including actions that require p since p is needed in any `sr`-plan. However, these actions will be added later on when iterating over other propositions.

Algorithm 8 BDD Action Ordering

```
procedure BDDActionOrdering($\mathcal{A}$, $Q$)
    while $|Q| > 0$ and $|\mathcal{A}| > 0$ do
        Remove first proposition $p$ in $Q$
        for $a \in \mathcal{A}$ do
            if $p \in \texttt{add}(a)$ then
                Add $a$ to $\Lambda$ and remove $a$ from $\mathcal{A}$
        if $p \notin \mathcal{L}^{\mathcal{P}}$ then
            for $a \in \mathcal{A}$ do
                if $p \in \texttt{pre}(a)$ then
                    Add $a$ to $\Lambda$ and remove $a$ from $\mathcal{A}$
    return $\Lambda$
```

7. Relaxed BDD-based Heuristic

We now present our relaxed BDD heuristic for a DFP task Π^+ and show its admissibility and consistency. Given any reachable 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 7.1. Consider a DFP task Π^+, a state s, and a relaxed BDD $\mathcal{B}$ for s with $\mathcal{W} \geq 1$ constructed using Algorithm 1. Then, $h_{\mathcal{B}}(s)$ given by the shortest path in $\mathcal{B}$ is admissible.

Proof. Propositions 6.1 guarantees that no $\texttt{sr}$-plan is eliminated, while rules B3 and B4 guarantee the presence of at least one cost-optimal plan. Then, $h_{\mathcal{B}}(s) \leq \texttt{cost}(\pi^*_{\texttt{sr}}) \leq h^+(s)$, where $\pi^*_{\texttt{sr}}$ is the cost-optimal $\texttt{sr}$-plan from s and h^+ the perfect heuristic for Π^+. □

As in the MDD case, our implementation constructs a relaxed BDD $\mathcal{B}$ for s_I and updates $\mathcal{B}$ during search. Given a state s and the sequence of actions to reach s from s_I, $\pi_s = (a_1, .., a_k)$, we update $\mathcal{B}$ by removing all edges $e \in \mathcal{E}^{\mathcal{B}}$ with $\texttt{a}(e) \in \pi_s$ and $\texttt{l}(e) = 0$. We then iteratively apply the top-down and bottom-up procedures over $\mathcal{B}$ (lines 3 to 6, Algorithm 1) and compute the heuristic as:

$$h_{\mathcal{B}}(s) := \begin{cases} \omega^{\downarrow}(\mathbf{t}) - \texttt{cost}(\pi_s), & \pi_s \in \mathcal{B}, \\ \infty, & \text{otherwise}, \end{cases} \tag{5}$$

where $\pi_s \in \mathcal{B}$ represents that there exists a path $\rho \in \mathcal{B}$ with an edge $e \in \rho$ with $\texttt{l}(e) = 1$ and $\texttt{a}(e) = a$ for all $a \in \pi_s$. As in the MDD case (Section 5), $h_{\mathcal{B}}$ computed via (5) is a globally admissible heuristic (Karpas & Domshlak, 2012), which is sufficient to guarantee that a best-first type of search will find the optimal solution. In addition, (5) computes a consistent heuristic if neither the action nor proposition order changes when updating $\mathcal{B}$ for any state s (Theorem 7.2). As previously noted, the main reason for the consistency of $h_{\mathcal{B}}$ is that the successor state BDD is a subset of the current state BDD.

Theorem 7.2. 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 (5) 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 $h_{\mathcal{B}}(s) = \omega^{\downarrow}(\mathbf{t}_s) - \texttt{cost}(\pi_s)$ and $h_{\mathcal{B}}(s') = \omega^{\downarrow}(\mathbf{t}_{s'}) - \texttt{cost}(\pi_s) - \texttt{cost}(a)$. Then, $h_{\mathcal{B}}(s) - \texttt{cost}(a) - h_{\mathcal{B}}(s') = \omega^{\downarrow}(\mathbf{t}_s) - \omega^{\downarrow}(\mathbf{t}_{s'}) \leq 0$, and so $h_{\mathcal{B}}(s) \leq \texttt{cost}(a) + h_{\mathcal{B}}(s')$. □

7.1 Relationship with Disjunctive Landmarks

The set of propositional landmarks $\mathcal{L}^{\mathcal{P}}$ plays a key role on the relaxed BDD construction, specifically on the splitting and action ordering algorithms. We show that the accuracy of our heuristic is linked to both $\mathcal{L}^{\mathcal{P}}$ and the heuristics based on disjunctive action landmarks.

Given a state s, a *disjunctive action landmark* (Zhu & Givan, 2003; Helmert & Domshlak, 2009) is a set of actions $D \subseteq \mathcal{A}$ such that at least one action in D must be present in any plan from state s. Notice that each propositional landmark $p \in \mathcal{L}^{\mathcal{P}} \setminus s$ defines a disjunctive action landmark $\mathcal{A}(p) = \{a \in \mathcal{A} : p \in \texttt{add}(a)\}$, i.e., any plan from s needs an action that adds proposition p. In fact, any subset of propositional landmarks $L \subseteq \mathcal{L}^{\mathcal{P}} \setminus s$ defines a set of disjunctive landmarks $\mathcal{D} = \{\mathcal{A}(p) : p \in L\}$

The disjunctive action landmark problem is defined as follows: given a set of disjunctive landmarks $\mathcal{D}$, we look for a minimum-cost set of actions such that there is one action for each disjunctive action landmark $D \in \mathcal{D}$. The MILP model $\mathcal{D}_{MILP}$ represents this hitting set problem (Bonet & Helmert, 2010) for any disjunctive landmark set $\mathcal{D}$, where $x_a \in \{0, 1\}$ is binary variable representing if action $a \in \mathcal{A}$ is chosen or not. We will consider the optimal value of this problem to be the perfect disjunctive landmark heuristic $h^{\mathcal{D}}$. Notice that the hitting set problem is an NP-hard problem (Karp, 1972), so relaxations of the problem (e.g., LP relaxation) are often used as heuristics.

$$\begin{aligned} h^{\mathcal{D}} := \min & \sum_{a \in \mathcal{A}} \texttt{cost}(a) x_a & & (\mathcal{D}_{MILP}) \\ \text{s.t.} & \sum_{a \in D} x_a \geq 1 & \forall D \in \mathcal{D} & \\ & x_a \in \{0, 1\} & \forall a \in \mathcal{A} & \end{aligned}$$

Proposition 7.1 shows that the relaxed BDD heuristic is highly related to $h^{\mathcal{D}}$ when set $\mathcal{D}$ is defined over a subset of propositional landmarks $L \subseteq \mathcal{L}^{\mathcal{P}}$. In fact, $h_{\mathcal{B}}$ dominates $h^{\mathcal{D}}$ if all the propositions $p \in L$ are α-exact in all the BDD nodes.

Proposition 7.1. Consider a planning task Π^+, a reachable state s, a relaxed BDD $\mathcal{B} = (\mathcal{N}^{\mathcal{B}}, \mathcal{E}^{\mathcal{B}})$ for s and a set of propositional landmarks $\mathcal{L}^{\mathcal{P}}$ for s. Let $L \subseteq (\mathcal{L}^{\mathcal{P}} \setminus s)$ be a subset of propositional landmarks such that each proposition $p \in L$ is α-exact in every node $u \in \mathcal{N}^{\mathcal{B}}$. Then, $h^{\mathcal{D}}(s) \leq h_{\mathcal{B}}(s)$ where $\mathcal{D}$ is defined over L.

Proof. For any $\mathbf{r} - \mathbf{t}$ path $\rho \in \mathcal{B}$, consider $\pi_{\texttt{sr}}(\rho)$ as the set of actions associated with ρ, i.e., $a \in \pi_{\texttt{sr}}(\rho)$ if and only if there exist an edge $e \in \rho$ with $\texttt{l}(e) = 1$ and $\texttt{a}(e) = a$. We know that in every node $u \in \mathcal{N}^{\mathcal{B}}$ each proposition $p \in L$ is α-exact and also η-exact by definition ($\eta^{\downarrow}_{\texttt{A}}(\mathbf{r}) := \mathcal{L}^{\mathcal{P}}$, Section 6.2). Then, for each path $\rho \in \mathcal{B}$ and $p \in L$, there exists at least one action $a \in \pi_{\texttt{sr}}(\rho)$ that is also in $\mathcal{A}(p)$. Hence, the set of actions $\pi_{\texttt{sr}}(\rho)$ for any $\rho \in \mathcal{B}$ is a feasible solution for $\mathcal{D}_{MILP}$, where $\mathcal{D} = \{\mathcal{A}(p) : p \in L\}$. It follows that $h^{\mathcal{D}}(s) \leq \texttt{cost}(\pi_{\texttt{sr}}(\rho))$ for all $\rho \in \mathcal{B}$, therefore $h^{\mathcal{D}}(s) \leq h_{\mathcal{B}}(s)$. □

8. Relaxed MDD and BDD Comparison

We now summarize the main differences in our relaxed MDD and BDD approaches and present guidelines on when to use each technique. Table 1 compares the graphical structures in terms of encoding, size, and heuristic. As presented in Section 4, our relaxed MDD $\mathcal{M}$ encodes the DFP task, where paths in $\mathcal{M}$ correspond to sequences of actions. In contrast, our relaxed BDD $\mathcal{B}$ represents the sequential relaxation of a DFP task and, hence, paths in $\mathcal{B}$ correspond to sets of actions (see Section 6). These differences affect the meaning of edges and nodes. While an MDD edge in layer i represents the i-th action in the sequence, a BDD edge in the same layer encodes the decision of selecting the associated action. Also, nodes in $\mathcal{M}$ represent the aggregation of planning states, while nodes in $\mathcal{B}$ are aggregated sets of achieved and needed propositions.

Table 1: Relaxed DDs comparison

		Relaxed MDD $\mathcal{M}$	Relaxed BDD $\mathcal{B}$
Encoding	Problem	DFP task	Sequential relaxation
	Path	Sequence of actions (plan)	Set of actions (`sr`-plan)
	Edges	Applicable actions	Selected actions
	Nodes	Aggregated states	Aggregated achieved & needed prop.
Size	$\lvert\mathcal{N}\rvert$	$O(\mathtt{m} \cdot \mathcal{W})$	$O(\mathcal{W} \cdot \lvert\mathcal{A}\rvert)$
	$\lvert\mathcal{E}\rvert$	$O(\mathtt{m} \cdot \mathcal{W} \cdot (\lvert\mathcal{A}\rvert + 1))$	$O(2 \cdot \mathcal{W} \cdot \lvert\mathcal{A}\rvert)$
	$\lvert\mathcal{N}\rvert + \lvert\mathcal{E}\rvert$	$O(\mathtt{m} \cdot \mathcal{W} \cdot (\lvert\mathcal{A}\rvert + 2))$	$O(3 \cdot \mathcal{W} \cdot \lvert\mathcal{A}\rvert)$
Heuristic	Method	Critical path	Shortest path
	Relates to	h^{max} and planning graph	Disjunctive landmarks

The encoding difference translates into a size difference. The number of layers in $\mathcal{M}$ is an upper bound on the number of actions in a cost-optimal plan, $\mathtt{m} < |\mathcal{A}|$, while $\mathcal{B}$ has exactly $|\mathcal{A}|$ layers. Hence, given a maximum width $\mathcal{W}$, $\mathcal{M}$ has at most $\mathtt{m} \cdot \mathcal{W}$ nodes and $\mathcal{B}$ has at most $|\mathcal{A}| \cdot \mathcal{W}$. Since the maximum number of edges emanating from an MDD node is $|\mathcal{A}_0| = |\mathcal{A}| + 1$, the maximum number of edges in $\mathcal{M}$ is $\mathtt{m} \cdot \mathcal{W} \cdot (|\mathcal{A}| + 1)$. In contrast, $\mathcal{B}$ has at most $2 \cdot \mathcal{W} \cdot |\mathcal{A}|$ edges because each node has at most two emanating edges. Notice that, even though $\mathcal{M}$ usually has a smaller number of nodes than $\mathcal{B}$, the number of edges in $\mathcal{M}$ is higher than in $\mathcal{B}$. Overall, as long as $\mathtt{m} \geq 3$, $\mathcal{B}$ is smaller than $\mathcal{M}$ (i.e., $|\mathcal{N}^{\mathcal{B}}| + |\mathcal{E}^{\mathcal{B}}| < |\mathcal{N}^{\mathcal{M}}| + |\mathcal{E}^{\mathcal{M}}|$).

We also use different methods to calculate the relaxed DD heuristics and, thus, each one relates to different techniques. Since $h_{\mathcal{M}}$ is computed by a critical path procedure over $\mathcal{M}$, it has a strong relationship with critical path heuristics and the planning graph (see Section 5). On the other hand, $h_{\mathcal{B}}$ corresponds to the value of shortest path inside $\mathcal{B}$ and it relates to disjunctive landmark heuristics (see Section 7).

Lastly, these differences give us guidelines on when to use each graphical structure. Relaxed MDDs should be preferred when the sequential aspect of the problem is predominant and the number of actions is small. In contrast, relaxed BDDs are suited for domains where the sequential relaxation is a good approximation to the original DFP task and there is a high number of propositional landmarks. Our empirical results support the validity of these guidelines (see Section 11).

9. Exploiting the Relaxed DD Structure

Besides their use to compute admissible heuristics, we can exploit the graphical structure of relaxed MDDs and BDDs to reduce the search-space and dynamically improve their filtering rules. In the following, we explain how to extract plans from both graphical structures. We also show how we can leverage the BDD structure to find action landmarks and identify redundant actions.

9.1 Plan Extraction Procedures

One of the main advantages of representing all the cost-optimal plans in a graphical structure, either a relaxed MDD or BDD, is that we can extract plans by traversing the graph. The cost of such plans can be used both to improve the cost-based filtering rules (Rules M4 and B4) and also to avoid sub-optimal states during search (see Section 10).

Algorithm 9 MDD Plan Extraction Procedure

```
1: procedure MDDPlanExtraction($\mathcal{M}_s$, $s$)
2:     $\pi := \emptyset$, $u := \mathbf{r}$
3:     for $i \in \{1, ..., \mathtt{m}\}$ do
4:         $e :=$ SelectEdgeMDD($s$, $\mathcal{M}_s$, $u$)
5:         $a := \mathsf{a}(e)$, $u := \mathtt{t}(e)$, $s := s \cup \mathtt{add}(a)$
6:         Add action $a$ to $\pi$
7:         if $\mathcal{G} \subseteq s$ then return $\pi$
8:     return $\emptyset$
```

Given a reachable state s and its relaxed MDD $\mathcal{M}_s$, Algorithm 9 shows our plan extraction procedure. The algorithm starts from an empty plan and greedily chooses a path in the MDD that has the potential to be a plan. Specifically, the algorithm starts in the root node $\mathbf{r} \in \mathcal{N}^{\mathcal{M}}$ and iterates over each layer choosing an outgoing edge from the current node (lines 3-4). The SelectEdgeMDD procedure iterates over all the outgoing edges of a node u and selects an edge e such that the corresponding action $a = \mathsf{a}(e)$ is applicable in s and adds a new useful proposition. We give priority to actions that add a higher number of new propositions, in particular, if they add propositional landmarks. The algorithm then updates the current node, state, and plan using the selected edge (lines 5-6). The procedure ends when the current state s is a goal state or it has iterated over all the layers.

Algorithm 10 BDD Plan Extraction Procedure

```
1: procedure BDDPlanExtraction($\mathcal{B}_s$, $s$)
2:     $\pi := \emptyset$
3:     while $\mathcal{G} \not\subseteq s$ do
4:         UpdatePathsBDD($\pi$, $\mathcal{B}_s$)
5:         $L :=$ FindApplicableActions($s$)
6:         $a :=$ SelectActionBDD($L$, $\mathcal{B}_s$)
7:         if $a = \emptyset$ then return $\emptyset$
8:         Add action $a$ to $\pi$ and update state $s := s \cup \mathtt{add}(a)$
9:     return $\pi$
```

Since the relaxed BDD ignores the sequential aspect of the problem, we use a different approach to exact plans shown in Algorithm 10. 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 s that add at least one new proposition, stores them in a list L, and uses $\mathcal{B}_s$ to select the most promising action (lines 5-6). 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., $a' = \text{argmin}_{a \in L}\{\min\{\omega^{\downarrow}(u) + \mathsf{c}(e) + \omega^{\uparrow}(v) : e = (u, v) \in \mathcal{E}^{\mathcal{B}}, \mathsf{a}(e) = a, \mathtt{l}(e) = 1\}\}$. Notice that the extraction procedure ends if $\mathsf{SelectActionBDD}$ does not select any action, i.e., it returns an empty set (line 7). Lastly, if a is an action, we insert a to π and update state s (line 8).

9.2 Relaxed BDDs for Action Pre-Processing

We also develop a simple procedure to identify cost-optimal action landmarks and redundant actions using a relaxed BDD $\mathcal{B}$. Given a planning task Π^+ and a reachable state s, we say that an action $a \in \mathcal{A}$ is a *cost-optimal action landmark* if every cost-optimal plan from s has action a. Notice that the cost-optimal action landmark definition is more restrictive than the action landmark definition since it only considers cost-optimal plans. Any action landmark is a cost-optimal action landmark but the inverse is not necessarily true.

Given a relaxed BDD $\mathcal{B}$ for a DFP task Π^+ we identify cost-optimal action landmarks as follows. Consider an edge layer $\mathcal{E}_i^{\mathcal{B}}$ ($i \in \{1, ..., \mathtt{m}\}$) such that all edges $e \in \mathcal{E}_i^{\mathcal{B}}$ have label $\mathtt{l}(e) = 1$. Then, all cost-optimal plans have action $a = \mathsf{a}(\mathcal{E}_i^{\mathcal{B}})$, i.e., a is an action landmark for any cost-optimal plan. Similarly, we identify redundant actions by considering an edge layer $\mathcal{E}_i^{\mathcal{B}}$ such that all edges $e \in \mathcal{E}_i^{\mathcal{B}}$ have label $\mathtt{l}(e) = 0$. Then, no cost-optimal plan uses action $a = \mathsf{a}(\mathcal{E}_i^{\mathcal{B}})$, so a is a redundant action that can be removed from $\mathcal{A}$.

10. Implementation

In the following, we give a detailed explanation of our tree search algorithm and how it leverages the information from extracted plans. We also explain how the relaxed MDD and BDD are updated during search and the pre-processing procedures.[2]

10.1 Binary Tree Search

We implement a branch-and-bound binary tree search procedure (Land & Doig, 1960), where our relaxed DD structures can be easily integrated. This type of search has had successful results when used with other admissible heuristics (Pommerening & Helmert, 2012) and optimization techniques (Imai & Fukunaga, 2015) that tackle cost-optimal DFP.

Our binary tree search implementation is a variant of the one used by Pommerening and Helmert (2012) where we use a best-first strategy instead of a depth-first. We define a state of our search as $S = \langle s, \pi_{in}, \pi_{out}, c, h \rangle$ where s is a planning task state (i.e., set of propositions), π_{in} and π_{out} correspond to the tree search decisions, c is the cost to reach S, and h the heuristic value. In particular, π_{in} is the set of actions that are currently in the plan and π_{out} the set of actions that are not allowed to be part of the plan.

2. Our code is available at: `https://github.com/MargaritaCastro/dd-planning`.

Algorithm 11 illustrates the full binary tree search procedure. For notation purposes, we represent the components of a search state S as $S.s$, $S.\pi_{in}$, $S.\pi_{out}$, $S.c$ and $S.h$. The algorithm starts with a single search state $S_0 = \langle s_I, \emptyset, \emptyset, 0, 0\rangle$ associated with the initial state when no decisions have been made (i.e., $S_0.\pi_{in} = S_0.\pi_{out} = \emptyset$). The cost of S_0 is 0, and the heuristic value is given by any admissible heuristic (e.g., $h_{\mathcal{M}}$ or $h_{\mathcal{B}}$). The algorithm keeps a priority queue of search states Q_S ordered in increasing value of $S.c + S.h$ with ties broken preferring higher values of $S.c$. In addition, the algorithm keeps track of the best plan found so far, π, and the best upper and lower bound (UB and LB, respectively) of the cost-optimal plan. Lines 2 to 6 initialize all of these values. Our specific implementation uses the FF procedure (Hoffmann & Nebel, 2001) to extract an initial plan (line 5).

Algorithm 11 Binary Tree Search

```
1: procedure Binary Search($\Pi^+$)
2:     $S_0 := \langle s_I, \emptyset, \emptyset, 0, 0\rangle$
3:     $S_0.h :=$ComputeHeuristic($S_0$)
4:     $Q_S := (S_0)$
5:     $\pi :=$GetInitialPlan($\Pi^+$)
6:     $UB :=$ cost($\pi$), $LB := S_0.h$
7:     while $|Q_S| > 0$ do
8:         Retrieve first state $S$ from $Q_S$
9:         if $LB < S.c + S.h$ then $LB := S.c + S.h$
10:        if $LB \geq UB$ then return $\pi$
11:        $a :=$ChooseAction($S$)
12:        if $a = \emptyset$ then continue
13:        % Create first child: action is never selected %
14:        if cost($a$) $> 0$ then
15:            $S' := \langle S.s, S.\pi_{in}, S.\pi_{out} \cup \{a\}, S.c, S.h\rangle$
16:            Insert state $S'$ to $Q_S$
17:        % Create second child: action is selected %
18:        $S'' := \langle S.s \cup$ add($a$)$, S.\pi_{in} \cup \{a\}, S.\pi_{out}, S.c +$ cost($a$)$, 0\rangle$
19:        $S''.h :=$ ComputeHeuristic($S''$)
20:        if $S''.c + S''.h \leq UB$ then
21:            Insert state $S''$ to $Q_S$
22:            $\pi' :=$ ExtractPlan($S''$)
23:            if $\pi' \neq \emptyset$ and cost($\pi'$) $<$ cost($\pi$) then
24:                $\pi := \pi'$, $UB :=$ cost($\pi$)
25:    return $\pi$

26: procedure ChooseAction($S$)
27:    for $a \in \mathcal{A}$ do
28:        if $a \notin S.\pi_{out}$ and pre($a$) $\subseteq S.s$ and add($a$) $\not\subseteq S.s$ then
29:            return a
30:    return $\emptyset$
```

The binary tree search procedure iterates over the states in Q_S until it is empty (line 7) or we can prove that our current plan (i.e., incumbent solution) is optimal (line 10). For each search state, we update the lower bound (line 9) and choose an action to branch on (line 11). The ChooseAction procedure looks for any non-forbidden action that is applicable in the current state $S.s$ and that adds at least one proposition that is not in the current state (lines 26-29). If there is no action that satisfies these conditions the algorithm returns an empty set (line 30). When ChooseAction finds an action, our search procedure creates two child states: one where the action is forbidden and one where the action is applied to the current state. If action a has strictly positive cost, the first child is identical to its parent state with the exception that now a is in the forbidden set of actions (lines 14-16). In this case, we use the heuristic value of the parent state instead of computing the heuristic value again. Since zero-cost actions do not affect the cost of a plan, we avoid creating the first child if a is a zero-cost action.

The second child state corresponds to the decision of including action a to the current plan. As such, the state $S''.s$ considers the add effects of a, $S''.\pi_{in}$ includes a, and the cost and heuristic value are updated accordingly (line 18-19). If the cost estimation to a goal state is smaller or equal to our upper bound, we add the state into the queue (lines 20-21) and extract a plan. When the heuristic is given by either $h_{\mathcal{M}}$ or $h_{\mathcal{B}}$, we use their corresponding extraction plan procedures (Section 9). When using other heuristics this step can be simply checking that $S''.\pi_{in}$ is a valid plan. If the extracted plan has a smaller cost than our incumbent, we update π and our upper bound (lines 23-24).

10.2 Updating the BDD and MDD During Search

The computational effort to create a relaxed DD (MDD or BDD, accordingly) can be quite expensive depending on the number of actions and the chosen maximum width $\mathcal{W}$. To overcome the computational cost, our implementation creates a relaxed DD at the initial state and uses it to compute the heuristic for the other states in the search. For each search state S, we duplicate the initial state relaxed DD and update it considering the tree search decisions, π_{in} and π_{out}. Hence, we keep two relaxed DD at all times during search: one for the initial state and one for the current search state S that it is been evaluated.

Given the initial state MDD $\mathcal{M}_{s_I}$ and any state of the search S, $\mathcal{M}_S$ is a copy of $\mathcal{M}_{s_I}$ where we omit all the edges associated with actions in π_{out}. In addition, the first $k = |\pi_{in}|$ layers of $\mathcal{M}_S$ have a single edge where edge $e \in \mathcal{E}_i^{\mathcal{M}}$ $(i \leq k)$ is associated with the i-th action in π_{in}. Similarly, the relaxed BDD for state S, $\mathcal{B}_S$, is a copy of $\mathcal{B}_{s_I}$ where each edge e with $\mathtt{a}(e) \in \pi_{in}$ has label $\mathtt{l}(e) = 1$ and each edge e with $\mathtt{a}(e) \in \pi_{out}$ has a label $\mathtt{l}(e) = 0$.

Besides the computational gain, updating the relaxed DDs during search results in a consistent heuristic (see Sections 5 and 7). Moreover, our search algorithm remains complete since the relaxed DD heuristics only prune sub-optimal states.

10.3 Pre-processing Steps

Our planner includes a set of pre-processing tools to extract landmarks and identify redundant actions. We implement the same pre-processing algorithms used by Imai and Fukunaga (2014, 2015) since we mostly compare to their LP heuristic and MILP model. The landmark extraction algorithm is the same as the one described by Imai and Fukunaga (2015), which is a variant of existing procedures (Zhu & Givan, 2003; Keyder, Richter, & Helmert, 2010). The algorithm iterates over the actions and their add effects to identify propositional landmark candidates and finally extract propositional and action landmarks.

We also include the Iterative Variable Elimination procedure by Imai and Fukunaga (2015) to identify redundant actions. The algorithm includes a Relevance Analysis that starts from the set of goals and iterates over the set of actions to identify relevant actions, i.e., actions that add goal propositions or preconditions of other relevant actions. We also consider their Dominated Action Elimination procedure that checks if an action dominates another in terms of cost and add effects. Lastly, the authors present an Immediate Action Application procedure for zero-cost actions, that, in our case, is implemented inside our binary tree search algorithm (Algorithm 11, line 14).

11. Empirical Evaluation

We now present an empirical study of our relaxed MDD and BDD heuristics. We analyze our heuristics' performance using different maximum widths $\mathcal{W} \in \{2, 4, 8, 16, 32, 64\}$ and compare their performance against a MILP model for DFP (Imai & Fukunaga, 2014, 2015) and its LP relaxation. The MDD, BDD, and LP are used as heuristics within our binary search algorithm (see Section 10.1), while the MILP is solved once using an external solver.

Table 2: Selected IPC domain names and abbreviations.

Name	Abbre.	Name	Abbre.	Name	Abbre.
barman-opt11	bar11	nomystery-opt11	nom11	transport-opt11	tra11
barman-opt14	bar14	openstacks-opt11	ope11	transport-opt14	tra14
childsnack-opt14	chi14	parking-opt11	par11	visitall-opt11	vis11
elevators-opt11	ele11	parking-opt14	par14	visitall-opt14	vis14
floortile-opt11	flo11	pegsol-opt11	peg11	woodworking-opt11	woo11
floortile-opt14	flo14	scanalyzer-opt11	sca11		
ged-opt14	ged14	sokoban-opt11	sok11		

We test all approaches over delete-free version of domains from the IPC2011 and IPC2014 competitions. We restrict ourselves to domains with no negative preconditions and no conditional effects.[3] Table 2 shows the selected domains and their abbreviated names. Our experiments consider a 30 minute time limit and a 2GB memory limit. We use Gurobi 8.0 to solve the LP and MILP models. Everything is coded in `C++`.

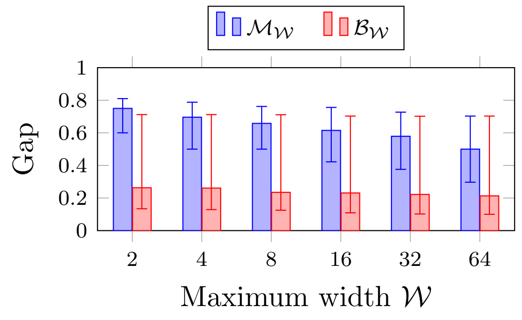
To make the comparison as fair as possible, we calculate an initial upper bound using the FF heuristic (Hoffmann & Nebel, 2001) and implement a plan extraction procedure for the LP heuristic. Whenever the LP returns an integer solution, we check if it is a plan. If so, we update the global upper bound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

11.1 Relaxed DD Heuristics Analysis

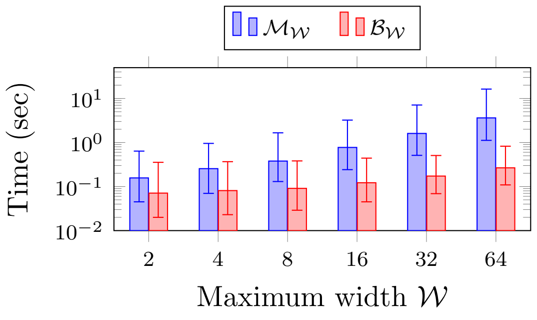
We first analyze the heuristic quality of our relaxed DDs in the initial state s_I. To do so, we compute the optimality gap (i.e., relative distance to the perfect heuristic) for a heuristic h at the initial state as $gap(h) = (h^+(s_I) - h(s_I))/h^+(s_I)$. Notice that an optimality gap closer to zero means a more informative heuristic.

3. No domains from IPC2018 satisfy these requirements.

Figure 6a compares the median optimality gap for each relaxed DD heuristic using different maximum widths. The error bars correspond to the first and third quartile of the gap. These values are calculated over all the instances where the cost-optimal plan (i.e., $h^+(s_I)$) was available, 314 out of 374 instances. The figure shows that as the maximum width $\mathcal{W}$ increases the gap decreases. However, there is a trade-off between informative heuristics and computational time. Figure 6b shows the median time to construct and compute a relaxed DD heuristic in the initial state, where the error bars correspond to the first and third quartile. Since the relaxed DD width grows exponentially, the time of its construction does too.



(a) Median optimality gap.



(b) Median heuristic time computation.

Figure 6: Initial heuristic comparison for different maximum widths.

From Figure 6b we also observe that the time to construct a relaxed MDD is much higher than the time for a relaxed BDD. This is mostly explained by the size discrepancy between the two graphical structures (see Section 8).

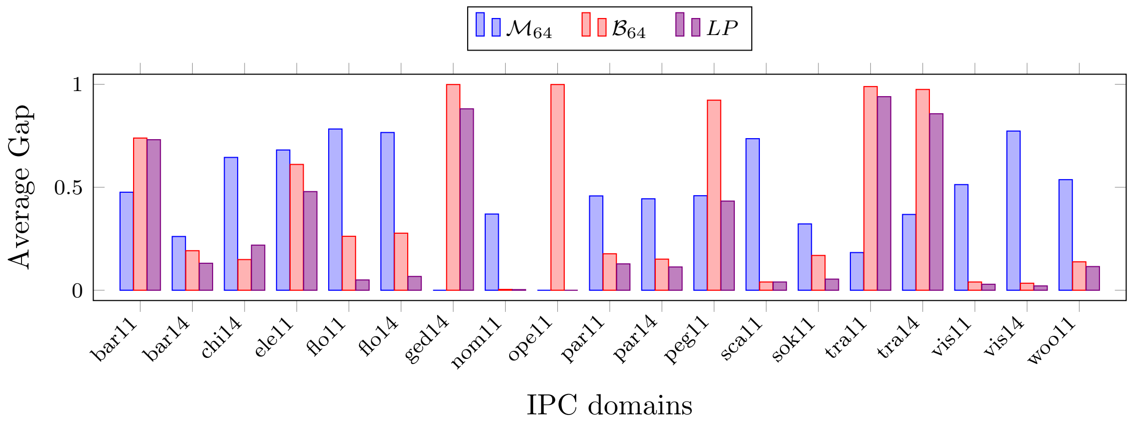


Figure 7: Initial heuristic comparison (best).

While Figure 6a shows that in median the relaxed BDD heuristic is more informative, this tendency depends on the DFP domain. Figure 7 compares the average optimality gap for our best relaxed MDD and BDD heuristics (i.e., with $\mathcal{W} = 64$) and the LP heuristic (see Appendix A for a complete list of optimality gaps for all widths). The figure shows that in most domains the LP heuristic has the smallest gap and $\mathcal{M}_{64}$ the largest ones. Nonetheless, the relaxed MDD heuristic has an exceptional performance in the two transport domains (*tra11* and *tra14*). Additionally, the $\mathcal{B}_{64}$ heuristic is perfect in *nom11*, a domain with a large number of propositional landmarks and where the optimal `sr`-plan cost is equal to the cost of the optimal DFP plan in all instances.

Figure 7 also shows that the $\mathcal{M}_{64}$ heuristic is perfect for all instances in domains *ged14* and *ope11* (i.e., gap = 0). These two domains have several zero-cost actions and in all instances the optimal plan needs exactly one action with positive cost. Since our search procedure skips branching on applicable zero-cost actions (see Section 10.1), our MDD can easily identify the needed positive cost action. In contrast, $\mathcal{B}_{64}$ evaluates to zero in all instances of these domains since the optimal cost of the sequential relaxation is zero. This explains the larger gap of $\mathcal{B}_{64}$, which is always equal to 1.

11.2 Effectiveness of MDD and BDD Heuristics

As discussed in the previous section, the quality of our relaxed DD heuristics depends on the specific domain at hand. We believe that relaxed MDDs generate informative heuristics when the sequential aspect of the problem is predominant (e.g., in the transport domain), while relaxed BDDs have better performance when the sequential relaxation is a good approximation to the original DFP task and there are a high number of propositional landmarks. These hypotheses arise from our construction procedures, their theoretical relationship with existing techniques, and the results presented in Figure 7.

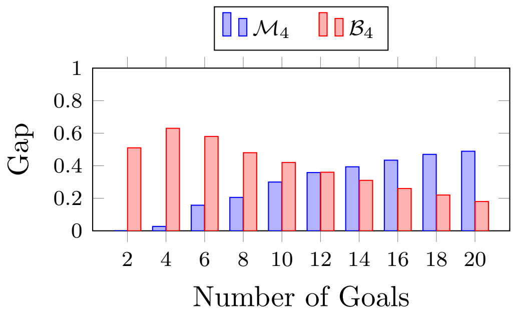
(a) Full grid: goals in any cell. (b) Half grid: goals in the right half of the grid.

Figure 8: Examples of random instances for 10×4 visit-all domain with $|G| = 4$.

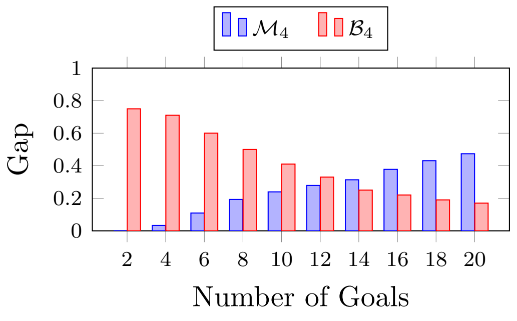
We test our hypotheses in modified instances from the visit-all domain using $\mathcal{M}$ and $\mathcal{B}$ with $\mathcal{W} = 4$. We consider a 10×4 grid with different number of randomly placed goals, $|\mathcal{G}| \in \{2, 4, 6, 8, 10, \ 12, 14, 16, 18, 20\}$. In all instances, the agent starts in the bottom left corner cell. We consider two different grid settings depicted in Figure 8. The first one, Figure 8a, considers one goal in the initial cell and the others placed randomly in the grid. The second grid setting also allocates one goal in the initial cell but all the other goals are allocated randomly in the right half of the grid, i.e., the blue region in Figure 8b. We create 10 random instances for each grid configuration (i.e., full and half) and number of goals.

Notice that the half grid setting is adversarial for the sequential relaxation. Since `sr`-plans are not forced to consider actions applicable to the initial state, cost-optimal `sr`-plans can avoid using actions in the left half of the grid. In addition, changing the number of goals allows us to analyze the effectiveness of our heuristic with different numbers of propositional landmarks.

Figures 9a and 9b present the average optimality gap at the initial state for each heuristic and configuration. The figures show that $\mathcal{M}_4$ is more informative when the number of goals



(a) Full grid.



(b) Half grid.

Figure 9: Initial heuristic comparison for different maximum widths.

is small (i.e., $|\mathcal{G}| \leq 10$). In these cases, our critical path heuristic to the most expensive goals is a good estimate for the cost-optimal plan. However, when the number of goals increases, the cost to reach the most expensive goals is a weak estimate.

When all the goals are in the right half of the grid the sequential relaxation is a bad approximation for the DFP task since there is no need to use any actions in the left side of the room. As a consequence, the optimality gap for $\mathcal{B}_4$ is worse in the half grid setting than in the full grid. Also, notice that in this domain the number of propositional landmarks is equivalent to the number of goals. Hence, fewer goals mean a less informative disjunctive landmark heuristic. This statement correlates with the behavior of $\mathcal{B}_4$, where we see an improvement in the optimality gap when the number of goals increases.

These experiments illustrate our claim for these domains: $\mathcal{M}$ is more informative when we have a small number of goals and the action sequencing is important. In contrast, $\mathcal{B}$ is informative in the presence of a large number of propositional landmarks and when the sequential relaxation is a good estimate of the DFP task.

11.3 Overall Performance Evaluation

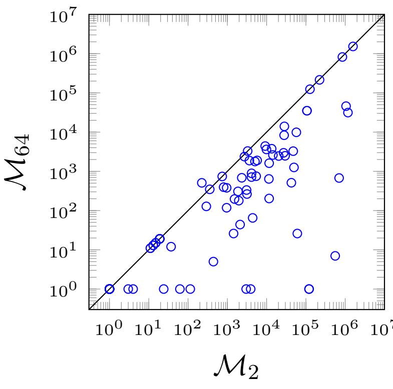
We now present the performance of our approaches and compare them with state-of-the-art techniques. Table 3 shows the coverage (i.e., number of instances solved) using our relaxed DD heuristics, where column "#" corresponds to the number of instances in each domain. Overall, the relaxed BDD heuristics solve approximately 100 more instances than when using the relaxed MDD heuristics. This correlates with the results presented in Figures 6 and 7 which show that the relaxed BDD heuristics are less computationally expensive and more informative in most domains. The only exception is the two transport domains where the relaxed MDD heuristics are more informative and solve more instances. Table 3 also highlights the trade-off of using different maximum widths. In some domains, it is preferable to use a smaller relaxed DDs while in others bigger diagrams solve more instances.

Figure 10 compares the number of states evaluated when using relaxed DDs of extreme widths (i.e., $\mathcal{W} = 2$ and $\mathcal{W} = 64$) for instances where both techniques find optimal solutions. The x and y coordinates of points in the plots correspond to the number of states evaluated by a relaxed DD with $\mathcal{W} = 2$ and $\mathcal{W} = 64$, respectively. We can see that for both relaxed DDs, most of the points are below the diagonal, which shows that bigger widths result in

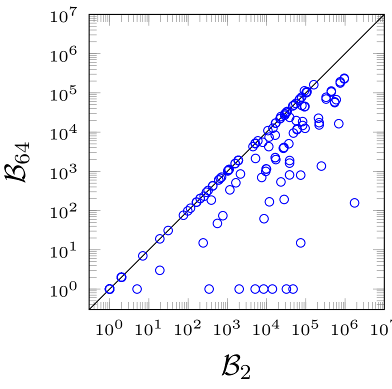
Table 3: Coverage for different relaxed MDDs and BDDs maximum widths.

		Number of Instances Solved											
domain	#	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
bar11	20	0	1	**4**	**4**	**4**	**4**	**0**	**0**	**0**	**0**	**0**	**0**
bar14	14	0	0	3	3	**6**	**6**	**14**	**14**	**14**	**14**	**14**	**14**
chi14	20	**4**	**4**	**4**	**4**	**4**	**4**	18	18	**20**	**20**	**20**	**20**
ele11	20	**16**	14	12	11	8	7	**20**	**20**	19	18	16	15
flo11	20	**2**	**2**	**2**	**2**	1	0	4	4	4	**5**	4	4
flo14	20	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
ged14	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
nom11	20	5	6	6	6	**8**	**8**	16	18	18	18	18	**20**
ope11	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
par11	20	0	0	0	0	**1**	**1**	**6**	4	3	3	3	3
par14	20	0	0	0	0	0	**1**	**9**	**9**	**9**	**9**	7	6
peg11	20	**19**	**19**	**19**	**19**	**19**	17	**19**	**19**	18	18	18	17
sca11	20	**1**	**1**	**1**	**1**	**1**	**1**	**7**	**7**	6	6	5	5
sok11	20	**20**	18	18	18	17	16	18	18	19	19	19	**20**
tra11	20	**2**	**2**	**2**	**2**	1	1	**1**	**1**	**1**	**1**	0	0
tra14	20	1	1	**3**	**3**	**3**	2	**1**	**1**	**1**	**1**	**1**	0
vis11	20	**9**	**9**	**9**	**9**	**9**	**9**	**16**	**16**	**16**	**16**	**16**	**16**
vis14	20	**3**	**3**	**3**	**3**	**3**	**3**	**17**	**17**	16	16	16	16
woo11	20	2	3	3	3	4	**6**	14	**18**	17	17	17	17
Total	374	124	123	**129**	128	**129**	126	221	**225**	222	222	215	214

fewer evaluated states.[4] However, points on the diagonal indicate that bigger widths do not always affect the number of states evaluated. This behavior is particularly true for relaxed BDDs that have a marginal increase in heuristic quality when the width increases (see Figure 6a).



(a) Relaxed MDD heurisitcs.

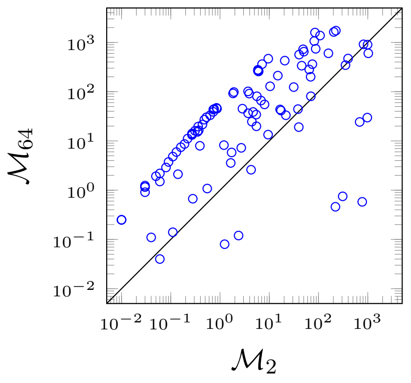


(b) Relaxed BDD heuristics.

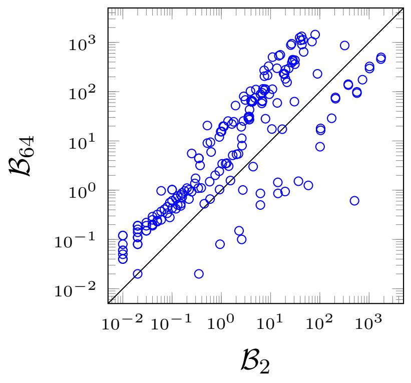
Figure 10: Number of states evaluated.

4. The only point above the diagonal corresponds to an instance where, by chance, $\mathcal{W} = 64$ extracted the cost-optimal plan later during search.

While using a larger width can lead to a significant reduction of states evaluated, Figure 11 shows that this reduction might not translate to faster solving time. Both plots have most points above the diagonal, illustrating that small-width relaxed DDs solve the problem faster. Nevertheless, there are some exceptional cases where larger and more informative relaxed DDs reduce the computational time by orders of magnitude.



(a) Relaxed MDD heuristics.



(b) Relaxed BDD heuristics.

Figure 11: Time (sec) to solve an instance.

Lastly, we compare the best performing relaxed MDDs and BDDs with the LP heuristic and the MILP model. The first set of columns in Table 4 shows the number of instances solved by each approach. MILP has the best coverage, followed by our best performing relaxed BDD and the LP heuristic. Nonetheless there are some domains where our relaxed MDD heuristic outperforms the MILP model (i.e., the first *transport* domain and *pegsol*). Also, our BDD heuristic outperforms the MILP model in *elevators*, *pegsol* and *scanalyzer*.

We notice that the DD heuristics achieve higher coverage than MILP in domains where the LP relaxation is weak. In particular, the MILP model has a large number of Big-M constraints when the sequencing aspect of the problem is predominant and, therefore, a weak relaxation (i.e., *transport* domains). We also notice that our tree-search procedure handles domains with zero-cost actions better (see Section 10.1), which explain the superior coverage of our approach in *ele11* and *peg11*. It is also interesting to see the large coverage difference between the MILP model and LP heuristic, which is due to the highly optimized MILP solvers and sophisticated techniques implemented on them (e.g., cut generation).

There are a number of domains where the BDD heuristic has a higher coverage than the LP heuristic even though the later has a smaller average gap (i.e., domains bar14, par11, par14, and woo11). There are two factors that played a key role in reducing the number of states evaluated and, thus, explaining these results. First, in these domains, the BDD heuristic is usually more informative than the LP heuristic when evaluating states closer to a goal state. Second, in general the BDD plan extraction procedure finds cost-optimal plans earlier during search than the LP methodology.

The second and third set of columns of Table 4 present the average run time and states evaluated by each approach over the instances that all of them solved. Since the MILP model is not implemented inside our search procedure, we only present the average states

Table 4: Comparison with state-of-the-art approaches.

		Coverage					Average Time (sec)					Average # States Evaluated			
domain	#	MILP	LP	$\mathcal{B}_4$	$\mathcal{M}_8$	$\mathcal{M}_{32}$	MILP	LP	$\mathcal{B}_4$	$\mathcal{M}_8$	$\mathcal{M}_{32}$	LP	$\mathcal{B}_4$	$\mathcal{M}_8$	$\mathcal{M}_{32}$
bar11	20	**8**	0	0	4	4	-	-	-	-	-	-	-	-	-
bar14	14	**14**	9	**14**	3	6	**0.7**	454.8	7.1	257.2	2.7	23,602.7	15,765.7	143,597.7	**506.0**
chi14	20	**20**	6	18	4	4	**0.4**	89.7	1.4	313.2	339.8	6,440.5	**5,996.5**	675,891.8	671,269.3
ele11	20	18	17	**20**	12	8	366.3	114.7	**46.0**	531.4	887.3	64,572.0	64,818.9	14,515.1	**5,771.4**
flo11	20	**20**	14	4	2	1	**0.3**	3.8	12.0	607.8	1417.3	**1,681.0**	23,998.0	222,565.0	146,945.0
flo14	20	**20**	16	1	0	0	-	-	-	-	-	-	-	-	-
ged14	20	**20**	**20**	**20**	**20**	**20**	391.1	39.5	**2.1**	8.6	40.8	25,773.8	2,110.9	**9.1**	**9.1**
nom11	20	**20**	18	18	6	8	0.9	0.8	**0.1**	11.4	0.5	8.5	**1.0**	2,100.5	11.0
ope11	20	**20**	**20**	**20**	**20**	**20**	0.9	0.8	**0.1**	1.1	6.1	**1.0**	2.0	**1.0**	**1.0**
par11	20	**18**	5	4	0	1	-	-	-	-	-	-	-	-	-
par14	20	**20**	7	9	0	0	-	-	-	-	-	-	-	-	-
peg11	20	16	**20**	19	19	19	180.5	**9.7**	158.0	116.2	251.3	9,907.7	228,516.2	9,130.1	**3,411.3**
sca11	20	6	5	**7**	1	1	0.3	0.3	**0.0**	0.2	0.1	20.0	**19.0**	348.0	348.0
sok11	20	**20**	**20**	18	18	17	151.2	**7.0**	9.7	94.5	203.3	**1,104.4**	6,895.9	5,885.6	3,197.6
tra11	20	0	0	1	**2**	1	-	-	-	-	-	-	-	-	-
tra14	20	**3**	0	1	**3**	**3**	-	-	-	-	-	-	-	-	-
vis11	20	**20**	16	16	9	9	**0.3**	1.0	1.1	84.4	101.6	**408.4**	3,574.8	50,985.2	14,804.4
vis14	20	**20**	17	17	3	3	**0.3**	0.5	1.4	234.2	265.5	**158.7**	3,802.0	145,634.3	45,795.7
woo11	20	**20**	14	18	3	4	0.6	6.1	**0.2**	473.9	282.8	**372.7**	450.0	54,806.0	8,154.0
Total\Ave.	374	**303**	224	225	129	129	143.6	33.9	**28.5**	121.6	187.9	**11,828.5**	40,311.1	43,175.9	29,588.8

evaluated for the other techniques. The results illustrate that there is a wide performance variability and different heuristics should be preferred in specific domains.

In some domains (e.g., *nom11* and *ope11*) the heuristic methods find optimal plans in the initial state, which is unusual in A^* search. Since our search algorithm has a bounding procedure (see Section 10.1), it stops as soon as the best lower bound is equal to the incumbent. Hence, if the heuristic at the initial state is perfect and the extracted plan has optimal cost, there is no need to evaluate any other state.

11.4 Using BDDs as a Pre-Processing Tool

Our last set of experiments evaluates the use of relaxed BDDs as a pre-processing tool. Table 5 shows the average percentage of redundant actions identified by a relaxed BDD in the initial state *after* applying traditional techniques (see Section 10.3). We can see that the average percentage is high in some domains, especially in *ged*, *nomystery* and *sokoban* with 6% to 22%. Notice that the identified redundant actions correspond to actions that, when used, will lead to sub-optimal or non-minimal plans. Hence, our approach identifies more redundant actions in instances with an accurate initial incumbent solution and where the sequential relaxation is a good approximation to the DFP task.

Lastly, Table 6 shows the average number of cost-optimal action landmarks found by our relaxed BDDs in the initial state after applying the standard pre-processing techniques (see Section 10.3). Our approach was able to identify a significant number of cost-optimal landmarks in some domains, in particular *ged* and *sokoban*. Both domains have a large number of zero-cost actions, which might explain why our relaxed BDDs identify more cost-optimal landmarks.

Table 5: Using Relaxed BDDs to find redundant actions.

	Average % of Redundant Actions					
domain	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
barman-opt11	1.2%	1.2%	1.2%	1.2%	1.2%	1.2%
barman-opt14	1.0%	1.0%	1.0%	1.0%	1.0%	1.0%
childsnack-opt14	0%	0%	0%	0%	0%	0%
elevators-opt11	0.3%	0.3%	0.4%	0.4%	0.4%	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%	6.2%	6.2%	6.2%
nomystery-opt11	21.6%	22.4%	22.9%	23.8%	24.7%	25.8%
openstacks-opt11	0%	0%	0%	0%	0%	0%
parking-opt11	0.8%	0.8%	0.8%	0.8%	0.8%	0.8%
parking-opt14	0.8%	0.8%	0.8%	0.8%	0.8%	0.8%
pegsol-opt11	0%	0%	0%	0%	0%	0%
scanalyzer-opt11	0%	0%	0%	0%	0%	0%
sokoban-opt11	8.7%	8.7%	8.7%	8.9%	8.9%	9.1%
transport-opt11	0%	0%	0%	0%	0%	0%
transport-opt14	0%	0%	0%	0%	0%	0%
visitall-opt11	5.3%	5.3%	5.3%	5.3%	5.3%	5.3%
visitall-opt14	0.4%	0.4%	0.4%	0.4%	0.4%	0.4%
woodworking-opt11	0%	0%	0%	0%	0%	0%

Table 6: Using Relaxed BDDs to find cost-optimal action landmarks.

	Average Number of Cost-Optimal Landmarks					
domain	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
barman-opt11	0	0	0	0	0	0
barman-opt14	0	0	0	0	0	0
childsnack-opt14	0	0	0	0	0	0
elevators-opt11	4.7	4.7	4.7	4.7	4.7	4.7
floortile-opt11	0	0	0	0	0	0
floortile-opt14	0	0	0	0	0	0
ged-opt14	148.4	148.4	148.4	148.4	148.4	148.4
nomystery-opt11	7.5	7.5	7.5	7.5	7.5	7.5
openstacks-opt11	0	0	0	0	0	0
parking-opt11	0	0	0	0	0	0
parking-opt14	0	0	0	0	0	0
pegsol-opt11	0.1	0.1	0.1	0.1	0.1	0.1
scanalyzer-opt11	0	0	0	0	0	0
sokoban-opt11	63.8	63.8	63.8	63.8	63.8	63.8
transport-opt11	0	0	0	0	0	0
transport-opt14	0	0	0	0	0	0
visitall-opt11	2.8	2.8	2.8	2.8	2.8	2.8
visitall-opt14	3.4	3.4	3.4	3.4	3.4	3.4
woodworking-opt11	0.5	0.5	0.5	0.5	0.5	0.5

12. Conclusions

This work presents new admissible heuristics for delete-free planning tasks based on relaxed decision diagrams. We introduce a novel relaxed MDD encoding for a planning task and a relaxed BDD representation for its sequential relaxation. The paper presents a theoretical analysis of both heuristics and relates them to existing techniques in the literature. We show that relaxed DDs can be used beyond heuristic computation. In particular, they enable the extraction of high-quality delete-free plans and relaxed BDDs can identify cost-optimal landmarks and redundant actions.

Our experimental results show that relaxed MDDs are suited for domains with a high number of sequential decisions, while relaxed BDDs perform better in domains with a large number of propositional landmarks. Overall, relaxed BDDs have competitive performance compared to an LP-based heuristic, but are still far from state-of-the-art MILP techniques. Nonetheless, our two relaxed DD heuristics achieve better coverage than a MILP model in four IPC domains.

While the paper focuses on delete-free planning tasks, our propose DD heuristics are novel to the planning community and can be used in a wider variety of planning problems. In particular, the DD heuristics are admissible for classical planning tasks and could be implemented in any planner that uses the STRIP formalism. However, it is not clear to us how to efficiently implement these heuristics, since it might require the planner to build a new DD in each node of the search space.

Another future direction is to combine DDs with other heuristics to solve, e.g., cost-optimal classical planning tasks. In particular, we can encode the DD graphs as a LP network flow model and use them inside the operator counting framework (Pommerening et al., 2014). Another alternative is to use the LP model of our DDs and combine it with other LP-based heuristics using the recently introduced Lagrangian decomposition framework in classical planning (Pommerening et al., 2019).

Lastly, our DD heuristics could be used to solve numerical planning tasks by extending our node encoding to consider numerical variables. In particular, it would be interesting to see how our MDD encoding could be extended to, e.g., represent the interval relaxation of numerical planning tasks (Piacentini et al., 2018).

Acknowledgments

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

Appendix A. Average Optimality Gaps

Table 7 presents the average optimality gap for each tested heuristic in the initial states. The average is computed over the 314 instances where the cost-optimal plans is known.

Table 7: Average optimality gap.

domain	LP	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
bar11	0.73	0.64	0.57	0.54	0.51	0.49	**0.48**	0.76	0.76	0.76	0.74	0.74	0.74
bar14	**0.13**	0.69	0.56	0.47	0.40	0.32	0.26	0.24	0.24	0.19	0.19	0.19	0.19
chi14	0.22	0.80	0.77	0.73	0.70	0.66	0.65	0.33	0.30	0.26	0.24	0.21	**0.15**
ele11	**0.48**	0.78	0.77	0.77	0.77	0.74	0.68	0.61	0.61	0.61	0.61	0.61	0.61
flo11	**0.05**	0.83	0.81	0.81	0.80	0.79	0.78	0.27	0.27	0.27	0.27	0.26	0.26
flo14	**0.07**	0.82	0.80	0.79	0.78	0.77	0.77	0.29	0.29	0.29	0.28	0.28	0.28
ged14	0.88	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1.00	1.00	1.00	1.00	1.00	1.00
nom11	**0.00**	0.70	0.63	0.56	0.48	0.41	0.37	0.02	0.02	0.01	0.01	0.01	**0.00**
ope11	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1.00	1.00	1.00	1.00	1.00	1.00
par11	**0.13**	0.78	0.72	0.66	0.60	0.52	0.46	0.18	0.18	0.18	0.18	0.18	0.18
par14	**0.11**	0.77	0.71	0.65	0.59	0.51	0.44	0.15	0.15	0.15	0.15	0.15	0.15
peg11	**0.43**	0.61	0.58	0.56	0.56	0.53	0.46	0.99	0.99	0.95	0.93	0.92	0.92
sca11	**0.04**	0.79	0.77	0.76	0.75	0.74	0.74	0.04	0.04	0.04	0.04	0.04	0.04
sok11	**0.05**	0.47	0.44	0.40	0.37	0.35	0.32	0.21	0.19	0.18	0.17	0.17	0.17
tra11	0.94	0.19	0.19	0.19	0.18	0.18	**0.18**	0.99	0.99	0.99	0.99	0.99	0.99
tra14	0.86	0.40	0.39	0.38	0.38	0.37	**0.37**	0.97	0.97	0.97	0.97	0.97	0.97
vis11	**0.03**	0.70	0.64	0.61	0.58	0.55	0.51	0.06	0.05	0.05	0.04	0.04	0.04
vis14	**0.02**	0.85	0.83	0.82	0.80	0.79	0.77	0.04	0.04	0.04	0.04	0.04	0.03
woo11	**0.12**	0.73	0.70	0.67	0.63	0.58	0.54	0.24	0.20	0.17	0.16	0.15	0.14

Appendix B. Solving Time and States Evaluated

Table 8 shows the average solving time and states evaluated for all our relaxed MDD heuristics. The average values are calculated over the instances where all relaxed MDD methods solve the task. Column "#" shows the number of instances where all approaches found a cost-optimal plan, i.e., the number of instances used to compute the average performance metrics. Similarly, Table 9 shows the average performance metrics for the relaxed BDD heuristics.

Table 8: Overall MDD Performance.

		Average Time (sec)						Average # States Evaluated					
domain	#	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$
bar11	0	-	-	-	-	-	-	-	-	-	-	-	-
bar14	0	-	-	-	-	-	-	-	-	-	-	-	-
chi14	4	331.3	320.7	**313.2**	317.5	339.8	377.7	694,820.5	685,161.8	675,891.8	671,269.8	671,269.3	**671,269.3**
ele11	7	**116.4**	252.4	543.3	797.4	815.1	948.9	16,271.4	16,181.4	15,626.0	10,800.9	5,852.6	**3,399.3**
flo11	0	-	-	-	-	-	-	-	-	-	-	-	-
flo14	0	-	-	-	-	-	-	-	-	-	-	-	-
ged14	20	**1.9**	4.1	8.6	18.5	40.8	89.2	**9.1**	**9.1**	**9.1**	**9.1**	**9.1**	**9.1**
nom11	5	260.8	61.0	5.7	0.6	**0.4**	**0.4**	160,914.8	22,354.8	1,257.0	38.8	7.0	**2.2**
ope11	20	**0.3**	0.5	1.1	2.6	6.1	14.8	**1.0**	**1.0**	**1.0**	**1.0**	**1.**	**1.0**
par11	0	-	-	-	-	-	-	-	-	-	-	-	-
par14	0	-	-	-	-	-	-	-	-	-	-	-	-
peg11	17	**15.1**	29.6	51.4	78.3	119.0	170.2	6,629.7	5,542.1	4,255.1	2,770.9	1,909.2	**1,140.4**
sca11	1	**0.1**	**0.1**	0.2	**0.1**	**0.1**	**0.1**	357.0	**348.0**	**348.0**	**348.0**	**348.0**	**348.0**
sok11	16	**62.0**	66.3	81.0	103.6	146.5	236.4	9,481.3	6,731.4	5,391.7	3,877.4	2,783.6	**1,952.5**
tra11	1	16.6	**14.3**	15.8	22.3	27.7	43.6	1,525.0	656.0	449.0	321.0	266.0	**197.0**
tra14	1	352.4	362.0	419.5	485.0	477.6	**345.5**	49,690.0	24,330.0	13,600.0	8,718.0	4,105.0	**1,258.0**
vis11	9	118.9	89.1	**84.4**	96.9	101.6	140.6	130,189.4	84,190.7	50,985.2	29,250.3	14,804.4	**9,031.7**
vis14	3	370.2	277.3	**234.2**	235.8	265.5	305.2	442,995.3	268,824.7	145,634.3	84,368.3	45,795.7	**23,254.0**
woo11	2	508.9	296.4	163.6	97.2	47.8	**24.4**	371,424.5	125,073.5	30,653.5	8,376.5	2,305.0	**594.0**
Average		78.4	**71.8**	91.3	118.4	138.5	184.3	68,466.8	47,240.4	37,259.6	32,195.5	29,087.1	**27,488.2**

Table 9: Overall BDD Performance.

		Average Time (sec)						Average # States Evaluated					
domain	#	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
bar11	0	-	-	-	-	-	-	-	-	-	-	-	-
bar14	14	**10.4**	22.8	37.7	70.3	109.9	117.1	32,398.7	30,821.4	27,411.1	21,713.3	14,215.2	**5,653.9**
chi14	18	448.8	369.7	230.9	122.8	**114.7**	123.9	367,694.7	291,537.8	180,355.6	96,305.4	82,250.7	**74,784.5**
ele11	15	**17.0**	30.8	60.1	109.1	239.5	514.2	45,027.3	43,549.5	42,218.9	42,054.8	42,103.2	**41,686.9**
flo11	4	86.7	87.1	**86.6**	119.1	154.0	235.8	173,597.0	116,613.3	67,609.5	44,016.0	26,286.3	**19,634.5**
flo14	1	**89.5**	128.2	179.1	197.6	175.0	230.1	210,121.0	178,847.0	126,276.0	58,064.0	38,774.0	**22,563.0**
ged14	20	**1.4**	2.1	3.7	5.2	10.4	21.8	2,110.9	2,110.9	2,110.9	2,110.9	2,110.9	**2,110.9**
nom11	16	**0.8**	0.9	1.0	1.1	1.5	2.2	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
ope11	20	**0.1**	**0.1**	**0.1**	0.2	0.3	0.5	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**
par11	3	**19.3**	31.9	51.0	83.9	146.9	273.3	**566.0**	**566.0**	566.7	567.7	569.3	570.3
par14	6	**24.2**	43.3	70.5	128.0	234.7	459.4	**475.8**	476.0	478.7	477.8	477.7	479.0
peg11	17	**14.4**	19.6	29.8	51.5	95.6	188.4	56,778.7	54,782.5	51,625.1	51,078.6	**51,078.6**	**51,078.6**
sca11	5	**28.4**	49.3	96.4	213.7	418.7	626.6	37,136.6	36,877.2	37,130.4	37,498.8	36,169.4	**31,590.2**
sok11	18	17.5	9.2	**7.2**	8.4	11.9	15.1	20,372.4	6,527.0	2,932.3	1,906.6	1,436.1	**1,008.1**
tra11	0	-	-	-	-	-	-	-	-	-	-	-	-
tra14	0	-	-	-	-	-	-	-	-	-	-	-	-
vis11	16	**0.5**	0.7	1.0	0.6	1.1	1.8	2,350.3	2,011.3	1,290.2	414.5	391.4	**343.6**
vis14	16	0.5	0.3	**0.2**	**0.2**	**0.2**	0.3	1,878.4	713.7	189.6	84.9	21.9	**12.9**
woo11	13	108.0	14.3	**9.6**	16.1	29.6	30.5	209,047.0	18,810.2	8,182.2	6,205.8	5,943.4	**3,829.8**
Average		55.8	45.9	**39.8**	44.4	71.3	118.1	64,365.0	42,306.2	29,470.8	20,436.1	18,122.5	**16,329.0**

References

Andersen, H. R., Hadzic, T., Hooker, J. N., & Tiedemann, P. (2007). A constraint store based on multivalued decision diagrams. In *Proceedings of the Principles and Practice of Constraint Programming*, pp. 118–132. Springer.

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., & Hooker, J. N. (2012). Variable ordering for the application of BDDs to the maximum independent set problem. In *Proceedings of the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 34–49. Springer.

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

Bergman, D., van Hoeve, W.-J., & Hooker, J. N. (2011). Manipulating MDD relaxations for combinatorial optimization. In *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pp. 20–35. Springer.

Betz, C., & Helmert, M. (2009). Planning with h^+ in theory and practice. In *Proceedings of the Annual German Conference on Artificial Intelligence*, pp. 9–16. Springer.

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1-2), 281–300.

Bonet, B., & Castillo, J. (2011). A complete algorithm for generating landmarks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 315–318.

Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, pp. 360–372. Springer.

Bonet, B., & Helmert, M. (2010). Strengthening landmark heuristics via hitting sets. In *Proceedings of the European Conference on Artificial Intelligence*, pp. 329–334.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *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., & Beck, J. C. (2018). Relaxed decision diagrams for cost-optimal classical planning. In *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning*, pp. 50–58.

Castro, M. P., Piacentini, C., Cire, A. A., & Beck, J. C. (2019a). Relaxed BDDs: An admissible heuristic for delete-free planning based on a discrete relaxation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 77–85.

Castro, M. P., Cire, A. A., & Beck, J. C. (2019b). An MDD-based Lagrangian approach to the multi-commodity pickup-and-delivery TSP. *INFORMS Journal on Computing*, Forthcoming.

Cire, A. A., & van Hoeve, W.-J. (2013). Multivalued decision diagrams for sequencing problems. *Operations Research*, *61*(6), 1411–1428.

Corrêa, A. B., Pommerening, F., & Francès, G. (2018). Relaxed decision diagrams for delete-free planning. In *Proceedings of the Workshop on Constraints and AI Planning*, pp. 1–2.

Edelkamp, S., Kissmann, P., & Torralba, Á. (2012). Symbolic A* search with pattern databases and the merge-and-shrink abstraction. In *Proceedings of the European Conference on Artificial Intelligence*, pp. 306–311.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3-4), 189–208.

García-Olaya, Á., Jiménez, S., & Linares López, C. (2011). The 2011 International planning competition. Technical Report.

Gefen, A., & Brafman, R. I. (2011). The minimal seed set problem. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 319–322.

Geißer, F., Keller, T., & Mattmüller, R. (2016). Abstractions for planning with state-dependent action costs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 140–148.

Haslum, P., Slaney, J. K., & Thiébaux, S. (2012). Minimal landmarks for optimal delete-free planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 353–357.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: what's the difference anyway?. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 162–169.

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

Imai, T., & Fukunaga, A. (2014). A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In *Proceedings of the European Conference on Artificial Intelligence*, pp. 459–464.

Imai, T., & 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.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pp. 85–103. Springer.

Karpas, E., & Domshlak, C. (2012). Optimal search with inadmissible heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Keller, T., Pommerening, F., Seipp, J., Geißer, F., & Mattmüller, R. (2016). State-dependent cost partitionings for Cartesian abstractions in classical planning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pp. 3161–3169.

Keyder, E., Richter, S., & Helmert, M. (2010). Sound and complete landmarks for And/Or graphs. In *Proceedings of the European Conference on Artificial Intelligence*, Vol. 215, pp. 335–340.

Kinable, J., Cire, A. A., & van Hoeve, W.-J. (2017). Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research*, *259*(3), 887–897.

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, *3*(10), 497–520.

Piacentini, C., Castro, M. P., Cire, A. A., & Beck, J. C. (2018). Linear and integer programming-based heuristics for cost-optimal numeric planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 6254–6261.

Pommerening, F., & Helmert, M. (2012). Optimal planning for delete-free tasks with incremental LM-cut. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 363–367.

Pommerening, F., Röger, G., Helmert, M., & Bonet, B. (2014). LP-based heuristics for cost-optimal planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 226–234.

Pommerening, F., Röger, G., Helmert, M., Cambazard, H., Rousseau, L.-M., & Salvagnin, D. (2019). Lagrangian decomposition for optimal cost partitioning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 338–347.

Porteous, J., Sebastia, L., & Hoffmann, J. (2001). On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the European Conference on Planning*, pp. 174–182.

Scala, E., Haslum, P., Magazzeni, D., & Thiebaux, S. (2017). Landmarks for numeric planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 4384–4390.

Torralba, Á., Linares López, C., & Borrajo, D. (2013). Symbolic merge-and-shrink for cost-optimal planning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pp. 2394–2400.

Torralba, Á., Linares López, C., & Borrajo, D. (2016). Abstraction heuristics for symbolic bidirectional search. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pp. 3272–3278.

Zhu, L., & Givan, R. (2003). Landmark extraction via planning graph propagation. In *Proceedings of the International Conference on Automated Planning and Scheduling, Doctoral Consortium*, pp. 156–160.