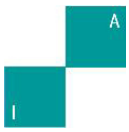
Inteligencia Artificial, 21(62) (2018), 67-74
doi: 10.4114/intartif.vol21iss62pp67-74



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

Characterizing and Computing All Delete-Relaxed Dead-ends

Christian Muise
IBM Research
christian.muise@ibm.com

Abstract Dead-end detection is a key challenge in automated planning, and it is rapidly growing in popularity. Effective dead-end detection techniques can have a large impact on the strength of a planner, and so the effective computation of dead-ends is central to many planning approaches. One of the better understood techniques for detecting dead-ends is to focus on the delete relaxation of a planning problem, where dead-end detection is a polynomial-time operation. In this work, we provide a logical characterization for not just a single dead-end, but for *every* delete-relaxed dead-end in a planning problem. With a logical representation in hand, one could compile the representation into a form amenable to effective reasoning. We lay the ground-work for this larger vision and provide a preliminary evaluation to this end.

Keywords: deadends, dead-ends, knowledge compilation, d-DNNF, BDD, SDD

1 Introduction

Learning conflicts and reasoning about them has long been recognized as an important component in many fields of artificial intelligence and optimization. Recently, a type of conflict found in planning that represents a state where the goal cannot be achieved, referred to as a *dead-end*, has received increased attention in the field. This interest culminated in the first ever International Planning Contest on Unsolvability in 2016 [15], which required the planners to detect whether or not the initial state of a planning problem was a dead-end. Effective techniques for dead-end detection are useful for determining problem unsolvability, as evidenced by the winning planners in the contest, as well as solving classical planning problems [7, 13], non-deterministic planning problems [17], and probabilistic planning problems [12, 3].

Perhaps one of the most well understood dead-end detection techniques is to search for a solution in the delete relaxation of a problem (i.e., assuming that actions in our model never delete what is true). Because solvability of the delete relaxation is a polynomial time operation, we can determine if a state is a dead-end in the delete relaxation in polynomial time as well. If this is the case, we can further conclude that the original state is a dead-end; if no plan exists in the delete relaxation, then no plan exists in the original problem.

In this work, we focus on characterizing what it means for a state to be a delete-relaxed dead-end, and we do so by introducing a novel logical encoding of the problem. From the encoding we can compile the theory into an equivalent representation that compactly represents all delete-relaxed dead-ends of a particular minimal form, while at the same time allowing efficient queries or modifications over the theory. For example, one could quantify the likelihood that a partially observable state is in

ISSN: 1137-3601 (print), 1988-3064 (on-line)

a dead-end, or manipulate the representation to compute all states that could lead to a delete-relaxed dead-end through the application of a particular action by using standard logical operations. The logical representation provides interesting insights into the structure of the delete-relaxed dead-end detection problem in relation to other planning encodings, and we elaborate on these connections throughout the paper.

Our contributions are 3-fold: (1) we provide a simple logical encoding to represent the set of delete-relaxed dead-end states in a STRIPS planning problem; (2) we present an extension that compiles away some of the naïve encoding to give us an equivalent encoding that is easier to reason with; and (3) we present a preliminary evaluation on a range of domains and knowledge compilers.

2 Preliminaries

Delete-Relaxed STRIPS Planning

In this work, we assume a STRIPS model of planning [6]. A problem is defined as a tuple, $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$, where $\mathcal{F}$ is the set of fluents in the problem, $\mathcal{I} \subseteq \mathcal{F}$ is the initial state (we assume all fluents not in $\mathcal{I}$ are false initially), $\mathcal{A}$ is the set of actions in the problem and $\mathcal{G} \subseteq \mathcal{F}$ is the goal that the planner must achieve. Every action $a \in \mathcal{A}$ is described by its precondition $\text{PRE}(a) \subseteq \mathcal{F}$ and add effects $\text{ADD}(a) \subseteq \mathcal{F}$. Traditionally, there is also a set of delete effects, $\text{DEL}(a) \subseteq \mathcal{F}$, but for this work we are only concerned with delete-relaxed planning: i.e., $\forall a \in \mathcal{A}, \text{DEL}(a) = \emptyset$. A complete state $s \subseteq \mathcal{F}$ describes what is true in the world, while every fluent in $\mathcal{F} \setminus s$ is presumed to be false. An action a is applicable in state s if and only if $\text{PRE}(a) \subseteq s$, and the result of applying a in s is defined to be $s \cup \text{ADD}(a)$. Note that because there are no delete effects, applying an action in a state can only add fluents to the state: i.e., make more of the fluents true. Further, because all of the preconditions are fluents as well, when an action is applicable it will remain applicable. We will use the following to refer to the actions that add a particular fluent:

$$adders(f) = \{a \mid a \in \mathcal{A} \text{ and } f \in \text{ADD}(a)\}$$

A planning problem, $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$, is *solvable* if a sequence of action applications transforms the state $\mathcal{I}$ into one where all of the fluents in $\mathcal{G}$ are true, and it is *unsolvable* otherwise. The state s is a delete-relaxed dead-end for problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ if and only if $\langle \mathcal{F}, s, \mathcal{A}, \mathcal{G} \rangle$ is unsolvable.

Satisfiability

We use the standard notion of Boolean logic to characterize the delete-relaxed dead-ends in a planning problem. Here we describe the basic concepts and notation, but the reader is referred to [1] for further details. The task of Satisfiability (or simply SAT) is to find a satisfying assignment to a set of Boolean variables given a logical formula. The standard representation for a logical formula, and the one we adopt, is Conjunctive Normal Form (CNF). A CNF formula is a conjunction of clauses, and each clause is a disjunction of literals; either a Boolean variable or its negation. For example, the following CNF formula,

$$(x \vee \neg y) \wedge (\neg x \vee y)$$

has two satisfying assignments: $(x = \top, y = \top)$ and $(x = \bot, y = \bot)$. While CNF is a natural form to express many problems, reasoning with it is a difficult task. Many other forms have been proposed in the literature that are more amenable to reasoning, and *knowledge compilation* is the task of converting from one form (typically CNF) to another in order to make reasoning easier [5]. We forgo describing the alternative forms here, but provide some empirical results in Section 4 on the difficulty of knowledge compilation from our proposed CNF encoding to a variety of target forms.

The final notion we will use is *projection*. The projection of a satisfying assignment onto a subset of the variables is simply the portion of the assignment that involves variables in the subset. We use the notion of projection to focus on only one aspect of the model, and it may be the case that many full assignments will map to the same projected assignment.

3 Encoding All Delete-Relaxed Dead-ends

To reason about all of the delete-relaxed dead-ends for a planning problem, we must construct a CNF where satisfying assignments correspond to delete-relaxed dead-ends. However, we do not want just any representation of delete-relaxed dead-ends, but instead one that captures the core reason that a state cannot achieve the goal. This can be useful for interpretability of deadends, but additionally (as we see later) allows for a far more compact encoding than the standard Planning-as-SAT encodings. To that end, we will only model delete-relaxed dead-ends that are in a particular fixed-point of delete-relaxed reachability.

Definition 1 (Fixed-Point State) *We say that a state s in a delete-relaxed planning problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ is a fixed-point state iff $\forall a \in \mathcal{A}, \text{PRE}(a) \not\subseteq s$ or $s \cup \text{ADD}(a) = s$.*

Intuitively, a state is a fixed-point state whenever applying additional actions will have no effect on the state (either an action is not applicable, or adds no new fluents to the state). Every delete-relaxed dead-end will have a corresponding fixed-point state s where $\mathcal{G} \not\subseteq s$, and it is the set of delete-relaxed dead-end fixed-point states that we model with our SAT encoding.

Not only are fixed-point delete-relaxed dead-end states more easy to work with when modeling, but they also represent the relevant core of what is causing a state to be a dead-end. For any state s, its corresponding fixed-point state s' will include everything achievable from s (and thus $s \subseteq s'$). With our encoding, we capture what does *not* hold in the fixed-point state (i.e., $\mathcal{F} \setminus s'$), and this will always be more general (i.e., fewer fluents) than what does not hold in s.

First, we present a simple but naïve encoding that achieves our objective of modeling the fluents not achievable in a fixed-point delete-relaxed dead-end. Then we prove the correctness of the encoding, and show an alternative encoding that captures the same set of solutions using fewer variables. The alternative encoding provides a better representation for knowledge compilers to work with due to the reduction in variables. We conclude with a discussion of some of the interesting properties that our encodings have, and their relation to existing notions in the literature.

3.1 Naive Encoding

The key insight that we use for our encoding is to focus on what *cannot* be achieved, rather than what *can* be achieved. This is in stark contrast with the vast majority of existing SAT encodings for planning-related problems. We should note that our aim is to model the space of all states that are a delete-relaxed dead-end for a particular problem, and not just identify if the initial state is a delete-relaxed dead-end.

Because we are interested in the states from which the goal cannot be achieved, our encoding focuses on those aspects of the problem that are *unachievable*. A fluent f is unachievable from state s whenever the problem $\langle \mathcal{F}, s, \mathcal{A}, \{f\} \rangle$ is unsolvable. Similarly, an action a is unachievable from state s whenever the problem $\langle \mathcal{F}, s, \mathcal{A}, \text{PRE}(a) \rangle$ is unsolvable.

Viewing the task of representing all delete-relaxed dead-ends in terms of what cannot be achieved leads us to a simple CNF encoding where the variables are defined as:

- $x_{\bar{f}}$: For every fluent $f \in \mathcal{F}, x_{\bar{f}}$ represents the fact that f is unachievable.
- $x_{\bar{a}}$: For every action $a \in \mathcal{A}, x_{\bar{a}}$ represents the fact that a is unachievable.

The clauses that we use to characterize all fixed-point delete-relaxed dead-ends are as follows:

$$\bigvee_{f \in \mathcal{G}} x_{\bar{f}} \tag{1}$$

$$x_{\bar{a}} \rightarrow \bigvee_{f \in \text{PRE}(a)} x_{\bar{f}} \qquad \forall a \in \mathcal{A} \tag{2}$$

$$x_{\bar{f}} \rightarrow x_{\bar{a}} \qquad \forall f \in \mathcal{F}, \; \forall a \in adders(f) \tag{3}$$

The intuition behind each of the clause types is as follows: (1) some aspect of the goal must be unachievable; (2) if an action is unachievable, then some aspect of its precondition must be unachievable; and (3) if a fluent is unachievable, then every action that could add it must be unachievable.

The encoding is relatively small, and dominated by clauses of type 2 and 3. For type 2, each clause is only as large as the precondition of the action in question, and there are only $|\mathcal{A}|$ clauses of this type. For type 3, each clause is binary, and there will be $\sum_{f \in \mathcal{F}} |adders(f)|$ such clauses (a small number in typical planning problems). We will use $CNF(P)$ to refer to the encoding of planning problem P.

There are two special cases worth noting. First, if a fluent f has no action that can add it (i.e., $adders(f) = \emptyset$), then there will be no clauses of type 3: if f is false in the state of the world, it will remain unachievable. Second, if an action has no precondition then it is trivially *not* unachievable (i.e., it can always be executed, and its effects achieved).

A satisfying assignment will stipulate which actions and fluents are deemed unachievable. The *corresponding state* of a satisfying assignment is the state s defined from the fluent variable as $\{f \mid x_{\bar{f}} = \bot\}$. We can now establish the theoretical connection between the set of fixed-point delete-relaxed dead-ends for a problem P and $CNF(P)$:

Theorem 2 *The set of satisfying assignments for $CNF(P)$, projected to the fluent variables, corresponds one-to-one with the set of fixed-point delete-relaxed dead-ends of P.*

Proof Sketch. We first establish that the projection of any satisfying assignment onto the fluent variables corresponds to a fixed-point delete-relaxed dead-end. Formulae 2 and 3 together ensure that the assignment is a fixed-point state: if it was not, then some action a must be applicable in the corresponding state with $f \in \text{ADD}(a)$ and f marked as being unachievable: i.e., $x_{\bar{a}}$ and $x_{\bar{f}}$ hold in the assignment while a is applicable in the corresponding state. Note, however, that this leads to a contradiction: the contrapositive of formula 2 stipulates that if every precondition fluent of a is not unachievable, then a is not unachievable as well. Finally, the assignment must correspond to a delete-relaxed dead-end, as it is a fixed-point state and some aspect of the goal is unachievable (following formula 1).

For the other direction, consider a candidate delete-relaxed dead-end fixed-point state s. We construct a corresponding satisfying assignment by setting the following variables to true and all others false:

$$\{x_{\bar{f}} \mid f \notin s\} \cup \{x_{\bar{a}} \mid \text{PRE}(a) \not\subseteq s\}$$

Formulae 2 and 3 naturally follow from the properties of s being a fixed-point state, and 1 holds from the fact that s cannot contain the complete goal (as it is a dead-end). □

3.2 Fluent-based Encoding

While the above encoding is simple and intuitive, it unnecessarily contains information in the form of variables for action unachievement. Here, we derive an alternative encoding that removes the variables corresponding to the actions. We do so by using the transitive property of the implications in formulae 3 and 2 above; by combining them into a single formula, we obtain the following:

$$x_{\bar{f}} \rightarrow \bigvee_{f' \in \text{PRE}(a)} x_{\bar{f}'} \qquad \begin{array}{c} \forall f \in \mathcal{F}, \\ \forall a \in adders(f) \end{array} \tag{4}$$

Formulae 1 and 4 combined capture the precise set of delete-relaxed dead-ends for a problem. The number of variables is reduced to just the number of fluents in the problem, but the number of clauses is increased as a result: one clause of size $|\mathcal{G}|$ for the goal, and another $\sum_{f \in \mathcal{F}} |adders(f)|$ clauses of size $|\text{PRE}(a)|$. Just as $adders(f)$ is typically small, so is the size of $\text{PRE}(a)$. We found this increase to be negligible. Because the correctness of the fluent-based encoding follows directly from the logical combination of formulae 2 and 3 to produce formula 4, we forgo a formal proof here.

3.3 Discussion

A number of SAT encodings have been proposed for modeling planning problems, but to the best of our knowledge they all focus on modeling the existence of plans in some form. The idea of planning-as-SAT was pioneered in the early 1990's as a method for solving planning problems [9]. Recent advances

Domain	bddmin		c2d		cnf2bdd		DSHARP		minic2d		sharpSAT	
	act	flu	act	flu	act	flu	act	flu	act	flu	act	flu
airport (50)	0	2	7	**12**	2	6	3	4	7	10	4	5
floortile (20)	0	0	10	10	0	7	2	2	4	**14**	5	6
mystery (30)	0	4	7	8	2	6	6	6	6	6	7	**9**
parcprinter (20)	0	0	14	**16**	0	0	0	0	7	8	0	0
pegsol (20)	0	0	**20**	**20**	0	17	0	**20**	0	14	**20**	**20**
sokoban (20)	0	0	9	**10**	0	0	0	0	2	5	3	3
trucks (30)	0	1	10	**11**	0	6	1	3	5	5	2	4
woodworking (20)	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
ALL (210)	0	8	78	**88**	5	43	13	36	32	63	42	48

Table 1: # of Problems Compiled. (Brackets): all encoded problems. **Bold**: greatest number of problems compiled.

have drastically improved the performance of planning-as-SAT [19], focused on optimal planning [20], computation of heuristics [2], planning with multi-valued variable representations [8], and computing maximally flexible plans [14]. All of these approaches, however, share a common thread: a satisfying assignment corresponds to a plan.

In contrast, our encodings capture (fixed-point) states of the delete-relaxed problem where no plan exists. Many planning-as-SAT encodings use a layered approach and must fix the number of actions in a plan without knowing in advance what depth would suffice. For those that do not use a layered approach (e.g., [20, 14]), a common bottleneck is the number of clauses required for preventing causal self support: e.g., action a_1 adds f_1, which enables a_2, which adds f_2, which enables a_1, etc. To rule out any such causal loops in a plan, a cubic number of clauses are required to model the transitive closure of "support". Our encodings do not need to pay this high cost in encoding complexity. Intuitively, this is because the fundamental property we are modeling (i.e., if a fluent is unachievable) is universally quantified: a fluent is unachievable if and only if *every* action that adds it is unachievable. This is in contrast with the fundamental *existential* property that non-layered planning-as-SAT encodings aim to capture: a fluent is achievable when *at least one* action (or the initial state) is able to achieve it. [1]

The final connection of interest is prime implicants and minimal delete-relaxed dead-ends. Prime implicants describe only those variable settings required for a Boolean assignment to be satisfying, and minimal dead-ends describe only those unachievable fluents required to be a dead-end, so there is reason to believe they would coincide. However, note that relaxing one Boolean variable in the encodings above amounts to saying that we have a fixed-point delete-relaxed dead-end *regardless* of whether or not a selected fluent or action is unachievable. Changing the setting to just one Boolean variable can easily cause the assignment to no longer be satisfying, and as a result the prime implicants do not correspond directly to the minimal delete-relaxed dead-ends. The final note of interest is with respect to minimal delete-relaxed dead-ends: i.e., a delete-relaxed dead-end with the fewest number of unachievable fluents specified. Every minimal delete-relaxed dead-end will correspond directly to some satisfying assignment of the proposed encoding. This follows from the fact that the set of satisfying assignments correspond to every fixed-point delete-relaxed dead-end, which necessarily includes the minimal delete-relaxed dead-ends, and it also underscores the subtle nature of the $x_{\bar{f}}$ variables: when set to false, many of the constraints become satisfied due to the structure of (3) or (4).

[1] The ability for the initial state to support a fluent is typically captured through an introduced "initial state action".

4 Evaluation

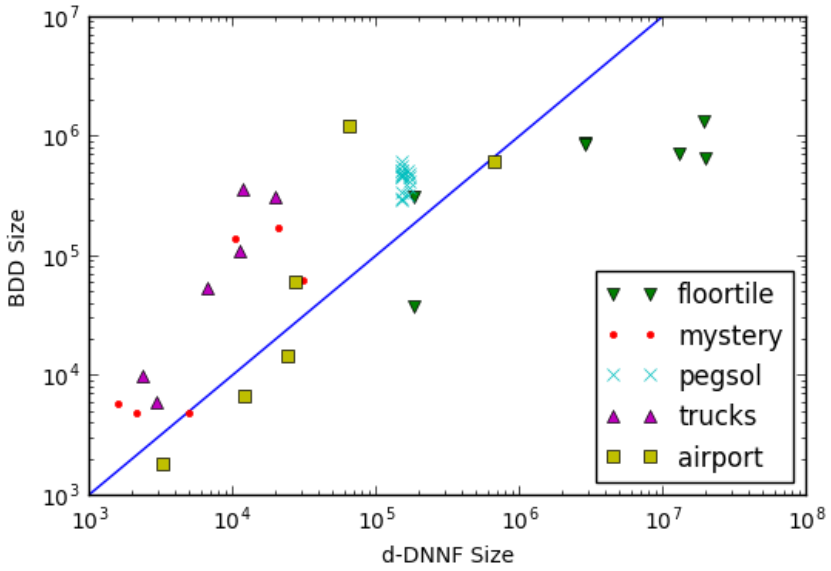


Figure 1: Size of the d-DNNF generated by c2d compared with the size of the BDD generated with cnf2bdd.

There are many possibilities for using the logical characterization of delete-relaxed dead-ends that we presented in Section 3. For our preliminary investigation, we evaluate the potential of compiling the theory into many of the popular target languages for knowledge compilation: (1) Deterministic Decomposable Negation Normal Form or d-DNNF [5]; (2) Sentential Decision Diagrams or SDD [18]; and (3) Binary Decision Diagrams or BDD [1]. We evaluate both encodings using the range of compilers that are available. For d-DNNF this includes Dsharp [16], c2d [4], and sharpSAT [23]. While sharpSAT is not a compiler, the Dsharp compiler is built on top of the sharpSAT code-base and so it provides a natural bound on performance for Dsharp. For SDD, we use the canonical compiler miniC2D [18]. Finally, for BDD, we use both BDD-MiniSAT [24] and the CUDD software package [22]; referred to here as cnf2bdd.

We implemented software to convert STRIPS planning problems (specified in PDDL) into CNF corresponding to the two encodings (the converter, benchmarks, and data will be released along with publication). All experiments were conducted on a 3.4Ghz Linux Desktop, and every trial was limited to 30min and 4Gb of time and memory. We used a selection of common benchmarks known to have dead-ends. In Table 1 we show the coverage for each of the tested compilers (in their best configuration) on each of the encodings. While the comparison across compilers is not direct, as they compile the encoding into different target languages, we can make some general observations.

The first aspect to note is that the fluent-based encoding strictly dominates the action-based one. Despite the increase in number of clauses, the search space becomes much more manageable. Investigating the data further, we found that roughly two thirds of all failures were due to memory violations, which indicates that storing the compiled form is the bottleneck in the action-based encoding.

The next key insight is that d-DNNF (via c2d) is the target language most easily compiled. This is a direct result of the more compact form the target language yields: Figure 1 shows the size comparison for the fluent-based problems both c2d and cnf2bdd could compile. Generally, the d-DNNF is far more compact with the notable exception of the floortile domain. While efficient reasoning can be done using d-DNNF, common symbolic planning operations such as progression or regression remain difficult. BDD's, on the other hand, are the target language of choice for many symbolic planning approaches, and so it is of particular interest to focus on cnf2bdd. We observed the biggest jump in coverage between encodings for a compiler occurred with cnf2bdd, indicating that there is far more structure in the fluent-based

encoding that the BDD can capture succinctly.

While only preliminary, the results paint a broad picture of the capabilities existing compilers have for dealing with the characterization of all delete-relaxed dead-ends in a domain. There remains plenty of room for improvement for the key compilers such as cnf2bdd in the form of planning-specific variable ordering [11] and approximate compilation [21].

5 Summary

Dead-end detection for classical planning is central to many state-of-the-art planners, and also a key component for approaches that solve more expressive planning formalisms. In this work, we take the first steps towards characterizing the space of all delete-relaxed dead-ends. We achieve this using a pair of Boolean logic encodings that are elegant in their simplicity, and remarkably, are not susceptible to the same level of complexity that similar related planning encodings face. We also performed a preliminary evaluation to test the range of knowledge compilation techniques that can process the Boolean encodings, and demonstrated the strengths and weaknesses that they have on a suite of benchmarks known to have dead-ends. Our work lays the foundation for incorporating a compact representation of delete-relaxed dead-ends in a larger system, and moving forward we hope to (1) leverage our encodings to improve planner performance; and (2) incorporate notions such as the Π-compilation for stronger dead-end detection by combining fluents [10].

References

[1] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. *Handbook of satisfiability, frontiers in artificial intelligence and applications.* IOS Press, 2009.

[2] Blai Bonet and Hector Geffner. Heuristics for planning with penalties and rewards using compiled knowledge. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 452–462, 2006.

[3] Alberto Camacho, Christian Muise, and Sheila A. McIlraith. From fond to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *The 26th International Conference on Automated Planning and Scheduling*, 2016.

[4] A. Darwiche. New advances in compiling CNF to decomposable negation normal form. In *Proceedings of European Conference on Artificial Intelligence*, pages 328–332, 2004.

[5] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[6] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice.* Elsevier, 2004.

[7] Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In *21st European Conference on Artificial Intelligence (ECAI 2014)*, pages 441–446, 2014.

[8] R. Huang, Y. Chen, and W. Zhang. SAS+ planning as satisfiability. *Journal of Artificial Intelligence Research (JAIR)*, 43:293–328, 2012.

[9] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *European Conference on Artificial Intelligence*, pages 359–363, 1992.

[10] Emil Ragip Keyder, Jörg Hoffmann, and Patrik Haslum. Semi-relaxed plan heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, 2012.

[11] Peter Kissmann and Jörg Hoffmann. Bdd ordering heuristics for classical planning. *Journal of Artificial Intelligence Research*, 51:779–804, 2014.

[12] Andrey Kolobov, Mausam, and Daniel S. Weld. A theory of goal-oriented mdps with dead ends. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 438–447, 2012.

[13] Nir Lipovetzky, Christian Muise, and Hector Geffner. Traps, invariants, and dead-ends. In *The 26th International Conference on Automated Planning and Scheduling*, 2016.

[14] Christian Muise, J. Christopher Beck, and Sheila A. McIlraith. Optimal partial-order plan relaxation via maxsat. *Journal of Artificial Intelligence Research*, 2016.

[15] Christian Muise and Nir Lipovetzky. Unsolvability international planning competition. `http://unsolve-ipc.eng.unimelb.edu.au`, 2016. Accessed: 2018-03-19.

[16] Christian Muise, Sheila A. Mcilraith, J. Christopher Beck, and Eric Hsu. DSHARP: Fast d-DNNF Compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*, 2012.

[17] Christian J Muise, Sheila A McIlraith, and J Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 2012.

[18] Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 3141–3148, 2015.

[19] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[20] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. Partial weighted maxsat for optimal planning. In *11th Pacific Rim International Conference on Artificial Intelligence*, pages 231–243, 2010.

[21] Mathias Soeken, Daniel Gro, Arun Chandrasekharan, Rolf Drechsler, et al. Bdd minimization for approximate computing. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 474–479. IEEE, 2016.

[22] Fabio Somenzi. Cudd: Cu decision diagram package release. `https://github.com/ivmai/cudd`, 2015. Accessed: 2018-03-19.

[23] M. Thurley. sharpSAT — counting models with advanced component caching and implicit BCP. In *Ninth International Conference on Theory and Applications of Satisfiability*, pages 424–429, 2006.

[24] Takahisa Toda and Takehide Soh. Implementing efficient all solutions SAT solvers. *CoRR*, abs/1510.00523, 2015.