Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Optimal Planning with Axioms

Franc Ivankovic and **P@trik Haslum**
Australian National University & NICTA Optimisation Research Group
firstname.lastname@anu.edu.au

Abstract

The use of expressive logical axioms to specify derived predicates often allows planning domains to be formulated more compactly and naturally. We consider axioms in the form of a logic program with recursively defined predicates and negation-as-failure, as in PDDL 2.2. We show that problem formulations with axioms are not only more elegant, but can also be easier to solve, because specifying indirect action effects via axioms removes unnecessary choices from the search space of the planner. Despite their potential, however, axioms are not widely supported, particularly by cost-optimal planners. We draw on the connection between planning axioms and answer set programming to derive a consistency-based relaxation, from which we obtain axiom-aware versions of several admissible planning heuristics, such as $h^{\max}$ and pattern database heuristics.

1 Introduction

In the classical planning model, state variables are modified by applying actions, and retain their value (persist) when not affected by an action. However, some properties of states are more naturally modelled not as direct effects of actions but instead as derived, in each state, from the basic state variables via a background theory, expressed in some logic.

Properties defined by a background theory were supported in several early planning systems [e.g., Manna and Waldinger, 1987; Barrett *et al.*, 1995; Bonet and Geffner, 2001], but fell out of use as the emphasis in planning shifted more towards efficiency than expressivity. Thiebaux et al. [2005] tried to promote this feature once more, by (re-)introducing it into PDDL and arguing that it allows many planning problems to be modelled in a more elegant and concise way. In their formulation, the theory that defines derived predicates is given by a logic program, with negation as failure semantics, whose rules are referred to as *axioms*. Nevertheless, support for derived predicates in recently developed planners is limited to a few systems (e.g., FF$_{\mathcal{X}}$, [Thiébaux *et al.*, 2005]; LPG [Gerevini *et al.*, 2005]; Marvin [Coles and Smith, 2007]; Fast Downward, [Helmert, 2006]; and LAMA [Richter and Westphal, 2010]), none of which are cost-optimal. In fact, few, and not very sophisticated, techniques for cost-optimal planning with axioms exist. (We discuss these, and their shortcomings, in Section 4.)

The contribution of this paper is twofold: First, we show, with examples both new and taken from existing literature, that axioms enable not only more elegant modelling but sometimes also more efficient solving of planning problems. This occurs because a model that uses axioms to derive indirect action effects removes unnecessary choices from the search space of the planner. Second, we apply our model of planning with global state constraints [Ivankovic *et al.*, 2014] to planning with axioms, exploiting the correspondance between the semantics of axioms and their logical foundation in answer set programming [Gelfond, 2008]. This enables us to derive problem relaxations that take the constraints on plans imposed by axioms into account. From the relaxations we can derive axiom-aware versions of admissible heuristics such as $h^{\max}$, LM-Cut and pattern databases.

2 Planning with axioms

We define axioms and derived state variables in the context of a finite-domain representation of planning problems (oft-called SAS$^+$), in a way that is consistent with the semantics of PDDL axioms [Thiébaux *et al.*, 2005] and the representation used in the Fast Downward planner [Helmert, 2009].

A planning problem consists of state variables, actions, axioms, an initial state and a goal condition. State variables are partitioned into *basic* variables, which are directly affected by actions and subject to persistence, and *derived* variables, whose values are computed via axioms. We will also refer to basic variables as *primary* and derived variables as *secondary* [Ivankovic *et al.*, 2014].

Each variable x has a finite domain $D(x)$ of values. For ease of presentation, we will assume that secondary variables are Boolean, i.e., their domain is $\{\mathrm{F}, \mathrm{T}\}$. A state is an assignment of values to all primary variables. The semantics of axioms (defined below) ensure that a complete valuation of the primary variables entails a unique consistent valuation of the secondary variables.

Conditions on states are built from atomic propositions of the form $x = v$, for $v \in D(x)$, and standard logic connectives. For Boolean variables, we abbreviate $x = \mathrm{T}$ as x and $x = \mathrm{F}$ as $\neg x$. Each action has a precondition ($\mathrm{pre}(a)$) and an

effect ($\text{eff}(a)$), which assigns new values to some subset of primary state variables.

An *axiom* is a rule $x \leftarrow \varphi$, where x is a secondary variable (called the *head* of the rule) and φ a state condition (called the *body*). The set of axioms with head x is said to *define* x. φ must be in negation normal form, meaning that negation is applied only to atomic propositions (primary or secondary). Intuitively, the meaning of an axiom is that when φ is true then the truth of x is inferred. The value of a secondary variable is false "by default", that is, unless implied by an axiom. Thus, the negation of secondary variables is defined by the "negation as failure" principle.

The set of axioms in a planning problem is required to be *stratified*. A stratification maps each secondary variable x to a "level", $l(x) \in \{0, \ldots, m\}$, such that if y appears in the body of an axiom defining x then $l(y) \leq l(x)$ and if $\neg y$ appears in the body of an axiom defining x, then $l(y) < l(x)$. In other words, stratification allows recursive definitions, but not "recursion through negation". The level mapping induces a partitioning of the axioms into sets $A_0, \ldots, A_m$ such that A_i contains the axioms defining all variables whose level is i. Testing stratifiability and computing a level mapping can be done in polynomial time [Thiébaux *et al.*, 2005].

Primary variable facts in a state together with the axioms form a logic program, and values of the secondary variables are given by the *stable model* of this program [e.g., Gelfond, 2008]. Stratification ensures that this model is unique, and can be computed by a stratified fixpoint procedure [as shown by Apt *et al.*, 1988]: Secondary variables are initialised to F (the default). Axiom strata are then processed in sequence, $A_0, \ldots, A_m$. For each axiom $x \leftarrow \varphi$ in stratum A_i, x set to T if φ evaluates to true. This is repeated until no variable changes anymore. The values of the secondary variables defined in stratum i are then fixed, and computation proceeds to the next stratum.

Example 1 We illustrate the formalism with a simple, abstract example, which we will also use later in the paper. Consider an undirected graph, with a designated source node s and target node t. K "roadblocks" are located on the edges of the graph, and can move between adjacent edges. (More than one roadblock can occupy the same edge, and one can pass another on the edge.) The goal is to move the roadblocks so that there is no unblocked path from the source to the target node. In other words, a goal state identifies an s-t-cut of size $\leq K$. Hence, we call it the Min-Cut domain. Figure 1 shows a small instance, with two roadblocks A and B.

The primary state variables are the locations of the roadblocks, at_l, and their domains is the set of edges. The secondary variables and their defining axioms are:

$$
\begin{array}{lll}
\mathsf{blocked}(i,j) & \leftarrow \mathsf{at}_l = e_{ij} & l = 1, \ldots, K \\
\mathsf{reachable}(i) & \leftarrow (i = s) & \text{(source node)} \\
\mathsf{reachable}(i) & \leftarrow \mathsf{reachable}(j) \wedge \neg\mathsf{blocked}(i,j) & j \in N(i) \\
\mathsf{isolated}(i) & \leftarrow \neg\mathsf{reachable}(i), &
\end{array}
$$

where $N(i)$ is the set of neighbouring nodes of i. Variables that appear in the body of an axiom but not in the head are implicitly existentially quantified. Note that $\mathsf{reachable}(i)$ means that i is reachable from s by an unblocked path. The action, $\mathsf{move}(l, e_{ij}, e_{jk})$, changes at_l from e_{ij} to e_{jk} as expected. The goal is $\mathsf{isolated}(t)$.

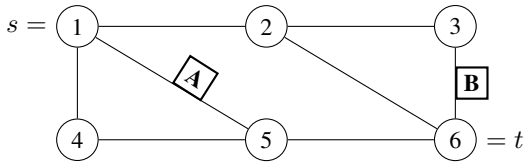


Figure 1: An example problem from the Min-Cut domain.

In the state in Figure 1, $\mathsf{at}_\mathbf{A} = e_{1,5}$ and $\mathsf{at}_\mathbf{B} = e_{3,6}$, so we can derive $\neg\mathsf{blocked}(1,2)$ and $\neg\mathsf{blocked}(2,6)$, and from this, $\mathsf{reachable}(1)$, $\mathsf{reachable}(2)$ and $\mathsf{reachable}(6)$. To achieve the goal, the two roadblocks must move to edges $e_{1,2}$ and $e_{5,6}$.

3 Modelling with axioms

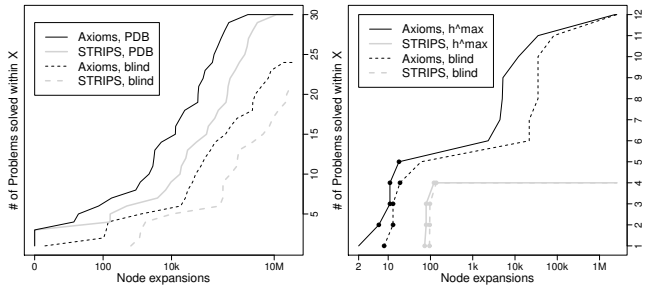
Thiébaux et al. [2005] argue that recursion makes axioms a natural way to express transitive closure properties, like the existence of paths (reachability) or flows. However, they can also express the negation of complex, including recursive, properties. Although planning problems with axioms can be reformulated without, the following examples show that their use improves both problem modelling and solving.

The Sokoban domain

The Sokoban puzzle is a well-known single-agent search benchmark, which has also been formulated as a planning problem and used in the IPC. It features a man who can move and push stones, one at a time, around a maze, with the goal of pushing all stones to designated goal squares. The objective (in this variant) is to minimise the number of stone pushes; moves of the man inbetween pushes do not count, as long as he is able to reach the square next to the stone to be pushed. (He cannot move through a square that contains a stone.) The classical planning formulation models the stepwise (non-pushing) moves of the man as actions with zero cost. This forces the planner to make an irrelevant choice of the exact path the man takes between each push action, increasing the size of the state space and plan length.

Reachability, given the current arrangement of stones, is straightforward to express as a recursively derived property. Thus, with axioms, the problem can be formulated with pushing actions only, allowing the man to "jump" from one stone to the next as long as a path between them exists.

Figure 2(a) shows the effect that this has on A* search. The graph shows the number of node expansions needed to prove optimality, i.e., for search to reach the $f^\star$ value. (This is a function of the size of the search space and the informedness of the heuristic only, not subject to tie-breaking variations.) We compare the IPC 2008 STRIPS formulation with one using axioms. The pattern database (PDB) heuristic is the canonical additive combination of several PDBs [Haslum *et al.*, 2007]. The axiom-aware PDB heuristic is described in Section 4. For the formulation with axioms, we use one PDB per stone location variable. In the STRIPS formulation, the location and "at-goal" status of each stone is split over two



(a) (b)

Figure 2: Node expansions required to prove optimality on equivalent problem formulations with and without axioms: (a) Sokoban problems; (b) controller verification problems due to Ghosh et al. [2015]. The door controller problems are marked with a dot (•) in (b).

variables, so we include both in each PDB to get an equivalent heuristic. CPU time is limited to 1h and memory to 3Gb per problem. Blind search on the STRIPS formulation expands, on average, 17 times more nodes than on the formulation with axioms. However, node expansion with axioms is, on average, an order of magnitude slower, so runtimes end up within a factor 2.7 of each other. The precomputation time for the axiom-aware PDB heuristic is also several orders of magnitude larger.

Pseudo-adversarial domains

By adversarial planning we usually mean game-like planning problems, where the planning agent faces an intelligent adversary who actively opposes or disrupts the plan. In some situations, however, the actions of the adversary are determined by a complex, but known and deterministic, procedure. We may call such problems "pseudo-adversarial".

Ghosh et al. [2015] encode verification of functional requirements of distributed automotive control systems as such a pseudo-adversarial planning problem. In their setting, the planning agent is the "environment" and the adversary is the control system, acting according to its specification (though the environment may exploit any non-determinism in the controller, by chosing the order of applicable control actions). The environment may only take actions when the controller is in a "stable" state, meaning no control action is applicable. This reflects an assumption that controller reaction times are faster than the pace of events in the environment.

Ghosh et al. term this "planning with action prioritization", and propose a compilation to classical planning. They add a Boolean variable, en_a, for each control action a, which indicates that a may be applicable, and add $\bigwedge_{a \in A_{ctl}} \neg \mathsf{en}_a$ to the precondition of every environment action and to the goal. A "disabling" action d_l for each literal l that appears in the precondition of some control action, with $\mathrm{pre}(d_l) = \neg l$ and $\mathrm{eff}(d_l) = \{\mathsf{en}_a{:=}\mathsf{F} \mid l \in \mathrm{pre}(a)\}$, is used to mark control actions inapplicable. Each (environment and control) action that potentially contributes to making $\mathrm{pre}(a)$ true sets $\mathsf{en}_a{:=}\mathsf{T}$, so the plan must include disabling actions before each environment action to verify its applicability. Because the compiled problems are hard, for the planners they tried, they also propose an incremental, partial compilation coupled with a plan repair approach. However, the requirement that no control action is applicable is easily formulated with axioms:

$$\begin{array}{lcl} \mathsf{stable} & \leftarrow & \bigwedge_{a \in A_{ctl}} \neg \mathsf{en}_a \\ \mathsf{en}_a & \leftarrow & \mathrm{pre}(a) \end{array}$$

Although these rules mirror almost exactly the actions in the compilation by Ghosh et al., making them axioms instead removes from the planner the choice of when and which disabling actions to apply, resulting in a smaller state space and shorter plans.

This is clearly seen in Figure 2(b). We compare the two compilations (plain STRIPS and with axioms) on two sets of verification problems provided by Ghosh et al. [2015], testing several safety properties of two control systems in a car: a door lock system and an adaptive cruise controller (ACC). Some of the problems have no plan: we count these as "solved" when search is able to prove plan non-existence. The door lock example is very simple, with only 6 control and 8 environment actions. The ACC example is much more complex, with 34 control and 691 environment actions. Neither blind nor heuristic search was able to solve any of the ACC problems in the STRIPS formulation, even with up to 4h CPU time and 60Gb of memory. Using the formulation with axioms, in contrast, all problems can be solved by blind search, the most difficult taking less than a minute.

The trapping game

As an example of a pseudo-adversarial domain with a more complex opponent strategy, we consider a game of two players, called the "blocker" and the "cat". The cat moves from node to node on a graph (G), always moving to a node on a shortest unblocked path to one of a set of designated exit nodes. (To ensure deterministic moves, ties are broken by an arbitrary ordering of the nodes.) In between each move of the cat, the blocker can permanently block a node in the graph (though not the node the cat currently occupies). The cat wins if it reaches an exit, and the blocker wins if the cat is trapped (i.e., no longer able to reach an exit).

To formulate the cat's strategy we need to determine which of two neighbouring nodes is closer to an exit:

$$\begin{array}{lll} \mathsf{closer}(n,n') \leftarrow \mathsf{dte}(n,i) \wedge \neg \mathsf{dte}(n',i) & & 0 \le i \le |G| \\ \mathsf{dte}(n,0) \leftarrow \mathsf{is\text{-}exit}(n) \wedge \neg \mathsf{blocked}(n) & & \\ \mathsf{dte}(n,i) \leftarrow \neg \mathsf{blocked}(n) \wedge \mathsf{dte}(n',i-1) & & n' \in N(n) \\ \mathsf{dte}(n,i) \leftarrow \mathsf{dte}(n,i-1) & & \end{array}$$

The secondary variable $\mathsf{dte}(n,i)$ ("distance-to-exit") is true if the shortest distance, along an unblocked path, to an exit from node n is i steps *or less*. Hence, node n is closer to an exit than n' iff $\mathsf{dte}(n,i)$ is true and $\mathsf{dte}(n',i)$ is false, for some i. The two nodes are at the same distance iff neither $\mathsf{closer}(n,n')$ nor $\mathsf{closer}(n',n)$ holds. Note that shortest distances are bounded by the size of the graph ($|G|$). The cat's preference can be expressed as

$$\begin{array}{lcl} \mathsf{prefer}(n,n') & \leftarrow & \mathsf{closer}(n,n') \\ \mathsf{prefer}(n,n') & \leftarrow & \mathsf{same}(n,n') \wedge n \le_{lex} n' \end{array}$$

(where $\le_{lex}$ is the arbitrary lexical ordering of the nodes). Let cat-pos be the primary variable representing the cat's current node: the precondition of action $\mathsf{cat\text{-}move\text{-}to}(n)$ is $n \in$

$N(\mathsf{cat\text{-}pos}) \wedge \bigwedge_{n' \in N(\mathsf{cat\text{-}pos}) \setminus \{n\}} \mathsf{blocked}(n') \vee \mathsf{prefer}(n, n')$. Since $\mathsf{dte}(n, i)$ is false for all i when there is no unblocked path from n to an exit, the blocker's goal that the cat is trapped can be written $\mathsf{trapped} \leftarrow \bigwedge_{0 \leq i \leq |G|} \neg \mathsf{dte}(\mathsf{cat\text{-}pos}, i)$.

Social and multi-agent planning

Interaction in a multi-agent setting is a more varied, and not always adversarial, game, in which agents may influence others to achive their own goals, drawing on their beliefs about others' behaviour in a given situation. Chang and Soo [2008] term this "social planning", and propose its use for automated narrative generation. Taking as their example Shakespear's play Othello, they observe that much of the story can be viewed as the execution of a plan by the villain of the play, Iago, who manipulates the other characters into carrying out his aims. In Chang and Soo's formulation, agents plan with actions of their own as well as actions performed by other characters, but each action taken by another character requires as a precondition that the character has a motive for achieving the action's effect. They propose that reasoning about characters' motives can be formalised with axioms. For instance, the axioms

$$\mathsf{motive\text{-}has}(c, c, i) \leftarrow \mathsf{greedy}(c) \wedge \mathsf{precious}(i) \wedge \mathsf{sees}(c, i)$$
$$\mathsf{motive\text{-}has}(c, c, i) \leftarrow \mathsf{motive\text{-}has}(c, c', i)$$

allow inferring that character c can be motivated to take action to achieve $\mathsf{has}(c, i)$ if c is greedy and i is a precious item that c has layed eyes on, or if c desires some other character c' to possess i.

Kominis and Geffner [2015] discuss planning in a multi-agent setting with partial observability and nested beliefs, meaning agents reason not only with what they know about the world state, but also with what they know about what other agents know. Agent's actions may change the world state, provide information about it (sensing) or provide information about another agent's knowledge (communication). Kominis and Geffner propose a compilation of sequential planning for this kind of problem to classical planning, which uses axioms.

Chang and Soo's and Kominis and Geffner's problems are still classical in the sense that the planning agent decides which actions are taken. But they enable better modelling of the conditions on when actions may, or must, be taken.

4 Relaxation of planning with axioms

A very simple approach to planning with axioms is to treat each axiom as a zero-cost action, whose precondition is the axiom body and effect the head. Initially assigning all derived variables the default value (F), any admissible classical planning heuristic computed on the relaxed problem yields an admissible estimate for the problem with axioms. We call this the *naive relaxation*. Because the naive relaxation does not force axioms to be applied, the resulting heuristic is blind to the difficulty of achieving the negation of a derived proposition.[1] Consider the Min-Cut domain from Example 1: in any state of the relaxed problem, the goal is achievable at no cost by *electing not to derive* $\mathsf{reachable}(t)$ and applying the axiom $\mathsf{isolated}(t) \leftarrow \neg\mathsf{reachable}(t)$.

We introduce two kinds of axiom-aware relaxations: one is based on the monotonic (or "value accumulating") relaxation that generalises the delete relaxation to non-propositional state variables [Gregory *et al.*, 2012; Domshlak and Nazarenko, 2013], and the other on abstraction, by projecting on a subset of primary variables. In both we have *relaxed states*, which can be viewed as representing sets of actual assignments to the primary state variables. The key question is how to evaluate conditions on secondary variables in a relaxed state. Our approach [Ivankovic *et al.*, 2014] treats this as a question of consistency, which is delegated to an appropriate external solver: In the case of planning with axioms, that is an answer set programming (ASP) solver. Therefore, we first briefly recap answer set programming.

Answer sets

Answer sets (also known as stable models) provide a declarative semantics for logic programs with negation as failure[2] and epistemic disjunction [e.g., Gelfond, 2008]. A (ground) answer set program is a set of rules of the form $H \leftarrow B$, where H, the head, is a disjunction of propositions and B, the body, a conjunction of literals. The intuitive meaning of the rule is that when all literals in B hold, then so does at least one proposition in H. A rule with an empty body is called a fact, and asserts that at least one proposition in the head holds. A rule with an empty head is called a constraint, and asserts that the literals in the body cannot all hold.

A model of a logic program is a set of true propositions that satisfies all rules of the program (taking propositions not in the set to be false). A stable model (answer set) is a model that is minimal w.r.t. set inclusion. Hence, both negation (in the body) and disjunction (in the head) are interpreted as a form of defaults: p is false, and $\neg p$ therefore true, unless p implied by the set of rules. A disjunction, $p \vee q$, is not exclusive, but as long as there is no other justification to infer that both p and q are true, the disjunction implies at most one of them.

4.1 Consistency-based monotonic relaxation

In the monotonic planning relaxation, primary state variables accumulate, rather than change, values [Domshlak and Nazarenko, 2013]. (This specialises to the well-known delete relaxation when all variables are Boolean; cf. Gregory et al. [2012].) Thus, in a relaxed state s^+ each primary variable x_i has a set of possible values, $s^+(x_i) \subseteq D(x_i)$, and s^+ itself represents the set of states obtainable by assigning each variable x_i one value from its set $s^+(x_i)$. That is,

$$\mathrm{states}(s^+) = \{\{x_1 = v_1, \ldots, x_n = v_n\} \mid \forall i : v_i \in s^+(x_i)\}$$

[1] The Fast Downward planner [Helmert, 2006] implements a mechanism that tries to address this problem: For each derived variable, y, that occurs negatively in some condition, a set of axioms defining $\neg y$ are added to the problem. If $A_y = \{y \leftarrow \varphi_i \mid i = 1, \ldots, k\}$ is the set of axioms that define y, $A_{\neg y} = \{\neg y \leftarrow \bigwedge_{i=1,\ldots,k} \neg\varphi_i\}$. Admissibility is preserved if the derived variable default value is taken to be neither true nor false. However, as the planner rewrites all axioms to DNF, this transformation often results in an exponential blow-up of problem size.

[2] Answer set programs may use both negation as failure and classical logical negation. However, since we use only the former, we omit the latter from discussion.

Relaxed application of an action effect x_i:=v adds v to the set of possible values of x_i.

This is an overapproximation of the set of reachable states, since some conjunctions of variable values may be unreachable even when each value is separately achievable [Haslum, 2012; Francès and Geffner, 2015]. A formula φ is true in a relaxed state s^+ if φ is true in some state in the set $\text{states}(s^+)$. Note that if φ is a conjunction of atomic propositions, $x_i = v$, over primary variables, this is equivalent to checking if $v \in s^+(x_i)$ for each proposition (if φ does not require any variable to have two values).

The set of all primary variable facts, $(x = s(x)) \leftarrow$, that hold in state s together with the axioms forms an answer set program, which we denote $\Pi(s)$. Because $\Pi(s)$ is stratified, it is consistent and has a unique answer set [Apt *et al.*, 1988]; the planning formalism defines the values of secondary variables in s as their values in this model. Hence, a secondary literal y is true in s iff $\Pi(s) \cup \{\leftarrow \neg y\}$ is consistent.

When viewed this way, the generalisation to evaluation in a relaxed state s^+ is immediate: For each primary variable, i, we have instead a disjunctive fact, $(\bigvee_{v \in s^+(x)} x = v) \leftarrow$, and a set of mutual exclusion constraints, $\leftarrow x = v, x = v'$ for all $v, v' \in s^+(x)$ such that $v \neq v'$, to ensure the variable has only one value. Again, we denote the answer set program that is the union of these facts, constraints, and the axioms of the planning problem by $\Pi(s^+)$.

Proposition 1 *Let s^+ be a relaxed state. Let $y_1, \ldots, y_n$ be secondary literals. $\Pi(s^+) \cup \{\leftarrow \neg y_1, \ldots, \leftarrow \neg y_n\}$ is consistent if and only if $y_1 \wedge \ldots \wedge y_n$ is true in some state in* $\text{states}(s^+)$.

Example 2 Consider the Min-Cut domain from Example 1 and the problem instance in Figure 1. With actions $\mathsf{move}(\mathbf{A}, e_{15}, e_{12})$ and $\mathsf{move}(\mathbf{A}, e_{15}, e_{56})$, we reach a relaxed state such that $s^+(\mathsf{at}_\mathbf{A}) = \{e_{12}, e_{15}, e_{56}\}$ and $s^+(\mathsf{at}_\mathbf{B}) = \{e_{36}\}$. The program $\Pi(s^+)$ has three stable models (one for each value of $\mathsf{at}_\mathbf{A}$) and $\mathsf{reachable}(6)$ is true in all, since no matter where roadblock **A** is placed the target node 6 is reachable as long as **B** does not move. Hence, $\Pi(s^+) \cup \{\leftarrow \neg\mathsf{isolated}(6)\}$ is not consistent.

We may add to $\Pi(s^+)$ further constraints, such as mutual exclusions between primary variable assignments. This excludes from $\text{states}(s^+)$ certain states that are not reachable in the original planning problem, thus strengthening the relaxation. It is the same idea as *constrained abstraction*, used to improve pattern database heuristics [Haslum *et al.*, 2005].

From relaxation to heuristics

Since relaxed action application can only add values, the set of states represented by the resulting relaxed state can only grow. This ensures monotonicity, in the sense that relaxed action application can never make false a condition that was previously true, and thus that the relaxation has the properties of the classical delete relaxation: Every plan for the original problem is also valid for the relaxed problem, which implies that optimal relaxed plan cost is an admissible heuristic. Applying an action more than once is redundant in the relaxed problem, which means that relaxed plan length is linearly bounded and that the optimal relaxed plan can be constructed as a minimum-cost hitting set over a collection of disjunctive action landmarks [Bonet and Helmert, 2010].

As we have previously shown [Ivankovic *et al.*, 2014], this means we can use standard techniques, such as building a relaxed planning graph, to compute relaxed reachability, invoking the consistency test each time we need to determine the truth of a secondary condition in relaxed state (i.e., RPG layer). Thus, we can compute the $h^{\max}$ heuristic.

We can also use the iterative landmark algorithm [Haslum *et al.*, 2012] to compute the optimal relaxed plan heuristic, h^+, or a weaker but less expensive heuristic, equivalent to the LM-Cut heuristic when all actions have unit cost [Bonet and Helmert, 2010]. However, our implementations of the axiom-aware versions of these heuristics are too slow to be useful.

4.2 Consistency-based abstraction

A projection is a problem abstraction in which a subset of (primary) state variables are treated normally, while remaining variables are ignored. Projections underlie pattern database heuristics (PDBs), which precompute optimal plan costs for all abstract states so that during search the heuristic value of a state can be found by looking up in a table the value of the corresponding abstract state [Culberson and Schaeffer, 1998; Edelkamp, 2001].

If A is the subset of primary variables projected on (the *pattern*), we can view an abstract state s^A as analogous to a relaxed state in which variables in A have only a single value and variables not in A have all possible values in their domain. That is,

$$\text{states}(s^A) = \{\{x_1 = v_1, \ldots, x_n = v_n\} \mid v_i = s^A(x_i) \text{ if } x_i \in A; \text{else } v_i \in D(x_i)\}.$$

Hence, we can formulate an answer set program $\Pi(s^A)$ in the same way as described above, and use it to test the consistency of secondary conditions in an abstract state.

Example 3 Consider again the Min-Cut problem in Figure 1, and a projection on the single-variable set $A = \{\mathsf{at}_\mathbf{A}\}$. The abstract state has $s^A(\mathsf{at}_\mathbf{A}) = \{e_{15}\}$ and $s^A(\mathsf{at}_\mathbf{B}) = \{e_{12}, e_{14}, e_{15}, e_{23}, e_{26}, e_{36}, e_{45}, e_{56}\}$, i.e., the set of all edges. Again, the program $\Pi(s^A) \cup \{\leftarrow \neg\mathsf{isolated}(6)\}$ is not consistent since node 6 remains reachable no matter where **B** is as long as **A** stays on edge e_{15}.

A standard PDB is efficiently constructed by an exhaustive reverse exploration from the (abstract) goal states, but this strategy is not easily adapted to problems with axioms. We use a two-stage PDB computation, in which the first stage builds an explicit graph of the reachable abstract space by forward exploration, and the second computes optimal costs over this graph. Hence, building an axiom-aware PDB takes more time than standard PDB construction. An advantage of PDB heuristics in this setting, however, is that the overhead is limited to the precomputation phase only; state evaluation is done by a table lookup, and takes no more time than in a standard PDB heuristic.

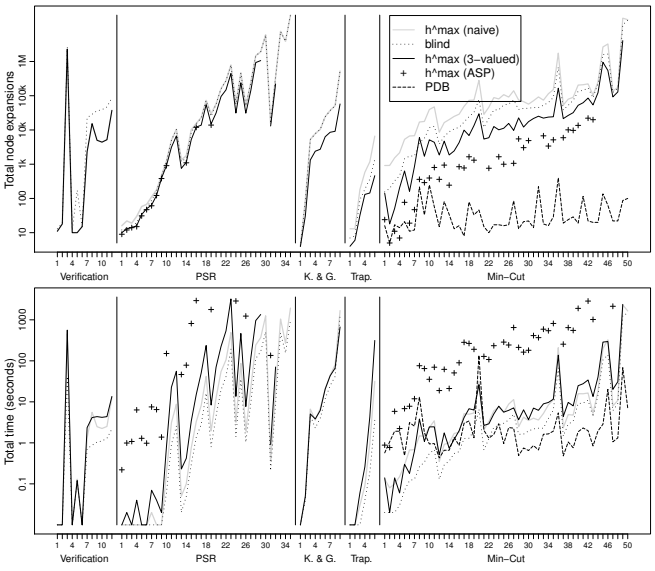


Figure 3: Total nodes expanded (above) and total planning time (below) with different heuristics. Instances in each set are sorted by increasing shortest plan length.

4.3 Exploring weaker relaxations

Invoking an answer set solver each time we need to test the truth of a secondary condition is time consuming. Although the heuristics we obtain are more informed, the time required to compute them means that in most cases they are not effective. However, we can also make other trade-offs between heuristic power and computational cost within our framework. As long as the consistency test is *sound* and we treat a secondary condition as false only when it is *proven* inconsistent with the relaxed state, our relaxation, and the heuristics we derive from it, remain admissible. (Francès and Geffner [2015] demonstrate the same point in classical planning.)

We have tried one instantiation of this idea, treating atomic propositions $x = v$ in a relaxed state s^+ as false if $v \notin s^+(x)$, true if $s^+(x) = \{v\}$, i.e., v is the only value in the set, and *unknown* (U) otherwise. Compound formulas are evaluated with the usual three-valued semantics (e.g., $\neg \mathrm{U} = \mathrm{U}$, $\mathrm{F} \wedge \mathrm{U} = \mathrm{F}$, $\mathrm{T} \wedge \mathrm{U} = \mathrm{U}$, etc). We then apply the stratified fixpoint evaluation procedure to compute a value in $\{\mathrm{F}, \mathrm{T}, \mathrm{U}\}$ for each secondary variable. A secondary literal y is considered true if it evaluates to T or U. This is sound, in that y evaluates to F only if y is false under every interpretation of the unknown propositions, and thus only if $\Pi(s^+) \cup \{\leftarrow \neg y\}$ is inconsistent. It is, however, not complete, making the relaxation weaker.

Example 4 Consider again the Min-Cut problem in Figure 1, and the relaxed state $s^+(\mathsf{at}_\mathbf{A}) = \{e_{12}, e_{15}, e_{56}\}$ and $s^+(\mathsf{at}_\mathbf{B}) = \{e_{36}\}$ from Example 2. Since $\mathsf{at}_\mathbf{A} = e_{12}$ and $\mathsf{at}_\mathbf{A} = e_{56}$ both evaluate to U, we can derive that $\mathsf{isolated}(6)$ is also U. Thus, the relaxation fails to prove that moving roadblock **A** alone is insufficient to reach the goal, unlike the stronger relaxations shown in Examples 2 and 3.

The impact of the different relaxations on search is shown in Figure 3. We use five sets of problems: the verification problems from Ghosh et al. [2015], the PSR domain (middle-size set from IPC 2004), compiled multi-agent planning problems due to Kominis and Geffner [2015], instances of the trapping game (cf. Section 3) played on graphs with 4–47 nodes, and random instances of the Min-Cut domain, with graphs of size 12–20 and 3–4 roadblocks. Each planner was run with up to 1h CPU time and 3 Gb memory per problem.[3]

If there are no negative secondary literals, the naive version of $h^{\max}$ coincides with 3-valued $h^{\max}$ (here, this happens only in the verification problems). For all other problems, the naive relaxation is as uninformed as blind search. The ASP-based relaxation is more accurate than the 3-valued relaxation only when some secondary facts can be derived from the disjunctions in a relaxed state. In our test set, this occurs only in the PSR and Min-Cut domains. (Results for the ASP-based $h^{\max}$ heuristic are omitted from the other domains, in which it expands exactly the same number of nodes but takes far more time to do so.) Although all heuristics except the naive reduce the amount of search, blind search is still often the fastest. (This has also been demonstrated in STRIPS planning, e.g., in the results of recent IPCs.) The 3-valued $h^{\max}$ heuristic is faster than blind search on the hardest of Kominis and Geffner's problems, and results with the axiom-aware PDB heuristic on the Min-Cut domain show that even the ASP-based relaxation can be sufficiently informative to compensate for the overhead of computing it. We did not try the PDB heuristic on other problems since we do now know which are good abstractions, if indeed any exist.

5 Conclusion

We have shown that the use of logical axioms to specify derived variables allows formulations of planning domains that are not only more compact and natural, but also more efficiently searchable. Although equivalent problems can be formulated without axioms, using them removes unnecessary choices from the search space of the planner.

Since axioms are essentially state constraints, our approach to defining planning relaxations by checking consistency, and using an external solver to perform the check [Ivankovic *et al.*, 2014], can be applied also to planning with axioms. This provides some validation of the generality of this idea. The relaxation defined here is potentially better since all information in the relaxed state is used for the consistency check, instead of selecting a subset of constraints through relaxed evaluation of conditions on primary variables.

Although the admissible heuristics we derive in this way are often informative, repeated consistency checking makes them often too slow. As suggested by Francès and Geffner [2015], sound but incomplete consistency tests based on propagation may present a way to balance inference power and computation time. The approach can work also with abstractions other than projection, such as the merge-and-shrink heuristics [Helmert *et al.*, 2007].

[3] Our implementation is built on the Fast Downward planner (fast-downward.org). The ASP solver is Clasp v2.1.3 (potassco.sourceforge.net).

Acknowledgements

This work was supported by ARC project DP140104219, "Robust AI Planning for Hybrid Systems". NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

[Apt *et al.*, 1988] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[Barrett *et al.*, 1995] A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, Y. Sun, and D. Weld. UCPOP users manual. Technical Report 93-09-06d, University of Washington, CS Dept., 1995.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. GPT: a tool for planning with uncertainty and partial information. In *IJCAI Workshop on Planning under Uncertainty and Incomplete Information*, pages 82–87, 2001.

[Bonet and Helmert, 2010] B. Bonet and M. Helmert. Strengthening landmark heuristics via hitting sets. In *Proc. 19th European Conference on Artificial Intelligence (ECAI)*, pages 329–334, 2010.

[Chang and Soo, 2008] Hsueh-Min Chang and Von-Wun Soo. Simulation-based story generation with a theory of mind. In *Proc. Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pages 16–21, 2008.

[Coles and Smith, 2007] Andrew Coles and Amanda Smith. Marvin: A heuristic search planner with online macro-action learning. *Journal of AI Research*, 28:119–156, 2007.

[Culberson and Schaeffer, 1998] J.C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[Domshlak and Nazarenko, 2013] C. Domshlak and A. Nazarenko. The complexity of optimal monotonic planning: The bad, the good, and the causal graph. *Journal of AI Research*, 48:783–812, 2013.

[Edelkamp, 2001] S. Edelkamp. Planning with pattern databases. In *Proc. 6th European Conference on Planning (ECP)*, pages 13–24, 2001.

[Francès and Geffner, 2015] G. Francès and H. Geffner. Modelling and computation in planning: Better heuristics from more expressive languages. In *Proc. 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 70–78, 2015.

[Gelfond, 2008] Michael Gelfond. Answer sets. In *Handbook of Knowledge Representation*, pages 285–316. Elsevier, 2008.

[Gerevini *et al.*, 2005] Alfonso Gerevini, Alessandro Saetti, Ivan Serina, and Paolo Toninelli. Fast planning in domains with derived predicates: An approach based on rule-action graphs and local search. In *Proc. AAAI Conference*, pages 1157–1162, 2005.

[Ghosh *et al.*, 2015] Kamalesh Ghosh, Pallab Dasgupta, and S. Ramesh. Automated planning as an early verification tool for distributed control. *Journal of Automated Reasoning*, 54:31–68, 2015.

[Gregory *et al.*, 2012] P. Gregory, D. Long, M. Fox, and C. Beck. Planning modulo theories: Extending the planning paradigm. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 65–73, 2012.

[Haslum *et al.*, 2005] P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for domain-independent planning. In *Proc. AAAI Conference*, pages 1163–1168, 2005.

[Haslum *et al.*, 2007] P. Haslum, M. Helmert, B. Bonet, A. Botea, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI Conference*, pages 1007 – 1012, 2007.

[Haslum *et al.*, 2012] P. Haslum, J. Slaney, and S. Thiébaux. Minimal landmarks for optimal delete-free planning. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 353–357, 2012.

[Haslum, 2012] P. Haslum. Incremental lower bounds for additive cost planning problems. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 74–82, 2012.

[Helmert *et al.*, 2007] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proc. 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 176 – 183, 2007.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of AI Research*, 26:191–246, 2006.

[Helmert, 2009] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5–6):503–535, 2009.

[Ivankovic *et al.*, 2014] Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar, and Dana S. Nau. Optimal planning with global numerical state constraints. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

[Kominis and Geffner, 2015] F. Kominis and H. Geffner. Beliefs in multiagent planning: From one agent to many. In *Proc. 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 147–155, 2015.

[Manna and Waldinger, 1987] Z. Manna and R. Waldinger. How to clear a block: A theory of plans. *Journal of Automated Reasoning*, 3:343–377, 1987.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of AI Research*, 39:127–177, 2010.

[Thiébaux *et al.*, 2005] Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005.