Journal of Artificial Intelligence Research 62 (2018) 373–431 Submitted 11/17; published 06/18

Extending Classical Planning with State Constraints: Heuristics and Search for Optimal Planning

Patrik Haslum PATRIK.HASLUM@ANU.EDU.AU
Franc Ivankovic FRANC.IVANKOVIC@DATA61.CSIRO.AU
The Australian National University, Canberra, Australia & Decision Sciences, CSIRO Data61

Miquel Ramírez MIGUEL.RAMIREZ@UNIMELB.EDU.AU
The University of Melbourne, Parkville, 3052, Australia

Dan Gordon DAN.GORDON@ANU.EDU.AU
Sylvie Thiébaux SYLVIE.THIEBAUX@ANU.EDU.AU
The Australian National University, Canberra, Australia

Vikas Shivashankar VIKSHIV@AMAZON.COM
Amazon Robotics
300 Riverpark Drive, North Reading, MA, USA

Dana S. Nau NAU@CS.UMD.EDU
Department of Computer Science & Institute for Systems Research, University of Maryland at College Park, College Park, MD, USA

Abstract

We present a principled way of extending a classical AI planning formalism with systems of state constraints, which relate – sometimes determine – the values of variables in each state traversed by the plan. This extension occupies an attractive middle ground between expressivity and complexity. It enables modelling a new range of problems, as well as formulating more efficient models of classical planning problems. An example of the former is planning-based control of networked physical systems – power networks, for example – in which a local, discrete control action can have global effects on continuous quantities, such as altering flows across the entire network. At the same time, our extension remains decidable as long as the satisfiability of sets of state constraints is decidable, including in the presence of numeric state variables, and we demonstrate that effective techniques for cost-optimal planning known in the classical setting – in particular, relaxation-based admissible heuristics – can be adapted to the extended formalism. In this paper, we apply our approach to constraints in the form of linear or non-linear equations over numeric state variables, but the approach is independent of the type of state constraints, as long as there exists a procedure that decides their consistency. The planner and the constraint solver interact through a well-defined, narrow interface, in which the solver requires no specialisation to the planning context. Furthermore, we present an admissible search algorithm – a variant of A* – that is able to make use of additional information provided by the search heuristic, in the form of preferred actions. Although preferred actions have been widely used in satisficing planning, we are not aware of any previous use of them in optimal planning.

1. Motivation

Proactive, informed control of infrastructure networks, such as power grids, transport systems or water networks, can optimise resource usage and increase reliability, with potentially significant economic and environmental benefits. Model-based AI technologies such as automated planning and diagnosis can play a key role in achieving this (Aylett, Soutter, Petley, & Chung, 1998; Thiébaux & Cordier, 2001; Piacentini, Alimisis, Fox, & Long, 2015).

However, modelling such interconnected physical systems requires an expressivity that classical planning formalisms do not provide. A key capability is expressing and reasoning with the global numeric constraints that govern network flows. A single (discrete) control action can, as a side effect, change the flow in all system components, and do so in a way that depends on the state of most components. For instance, closing a line switch in a power system can affect power flows in all the network lines, as well as the voltage and phase angle at all network buses, in a way that depends on the state of all other switches.

Although there is significant work on extending classical planning to hybrid discrete–continuous models, the most basic being the addition of numeric state variables (e.g., Koehler, 1998; Wolfman & Weld, 1999), the global scope of network flow constraints make them impractical to formalise as direct action preconditions and effects. Deciding if an action is applicable and determining its effects on the global state of the system requires solving a system of constraints potentially spanning the whole network. For instance, in AC power networks, we need to solve a system of non-linear equations linking real and reactive power flows to bus voltages to determine whether closing a switch would lead any line capacities to be exceeded.

Extending planning formalisms in a different direction, derived predicates and axioms (Thiébaux, Hoffmann, & Nebel, 2005; Hoffmann & Edelkamp, 2005) enable the compact encoding of a large class of state constraints, notably those involving the transitive closure of a relation, such as reachability. However, these are limited to expressing logical global relations derived from discrete atomic facts; there is, in prior work, no numerical counterpart to derived predicates. Moreover, support for derived predicates in current domain-independent planners remains sparse.[1]

We have proposed a mechanism for extending planning to formalisms with complex state constraints, and applied it to the specific cases of global numeric constraints[2] (Ivankovic, Haslum, Thiébaux, Shivashankar, & Nau, 2014) and PDDL axioms (Ivankovic & Haslum, 2015). The extended formalism remains classical in that we consider deterministic, fully known models and sequential plans, but allows for a richer class of planning models. In this paper, we present a variety of constraint-aware problem relaxations, from which we derive admissible heuristics, for the general extended formalism, i.e., in a way that is independent of the type of state constraints used. The relaxations are based on the well-known monotonic planning relaxation and on abstraction. We exemplify the general principle with problems that have numeric state constraints. We consider both linear and non-linear constraints, and

1. When we embarked on this investigation, there was, as far as we know, no work on *optimal* planning with derived predicates. We have since developed such a planner, based on the theory presented here (Ivankovic & Haslum, 2015).
2. We use the term "global" here to emphasise that the system of constraints may relate every part of a state. Some constraints are also *invariant*, meaning they must hold in every state, but this is not the case for all uses of state constraints in our extended formalism.

use a custom solver for the latter. We also present a new analysis of the expressivity of the extension. We show that independently of the type of constraints, problems in the extended formalism can be reduced to classical planning problems. However, we also show that extending classical planning with any sufficiently expressive constraint type is equivalent to allowing arbitrary formulas as action preconditions, which are known to require either an exponential increase in problem size or super-linear increase in plan length to compile away (Nebel, 2000).

Central to our approach is a partitioning of the planning model into a primary model – which is essentially classical – and a secondary model, in which the state constraints are captured. Such a division is made also in PDDL with axioms (Thiébaux et al., 2005) and in some other extended planning formalisms (Dornhege, Eyerich, Keller, Trüg, Brenner, & Nebel, 2009; Piacentini et al., 2015). Throughout the process of search and heuristic evaluation, the two models interact only through a well-defined, narrow interface. This means that responsibility for reasoning about the secondary model can be given over to a solver, suited to the task, with very weak requirements on the interface this solver must provide. In particular, and in difference to other recent proposals for integrating external reasoning components into planning (Dornhege et al., 2009; Gregory, Long, Fox, & Beck, 2012; Piacentini et al., 2015), computing relaxed plans and the associated heuristics does not require this solver to implement any form of relaxed reasoning about the constraints or their interaction with the primary model.

Motivated by the situation in which we have an accurate but expensive-to-compute admissible heuristic, we also proposed a new search strategy for cost-optimal planning, which makes use of preferred actions (Ivankovic et al., 2014). Preferred actions are an additional piece of information obtained as a by-product of the heuristic computation, and have been used extensively in non-optimal planning. Here, we present empirical evidence supporting our characterisation of the circumstances in which this strategy avoids generating many successor states and substantially reduces computation time.

Our motivation for focusing on optimal planning is primarily methodological: Advances in the optimal planning setting are simpler to evaluate, since the only dependent variable is efficiency. Developing admissible heuristics allows us to focus on the principles of the underlying relaxations and how to incorporate the state constraints of the extended formalism into them. Deriving non-admissible heuristics for satisficing planning from the relaxations following the same patterns as in classical planning is a straightforward exercise; we leave their evaluation to future work.

2. Extending a Classical Planning Formalism with State Constraints

In this section, we present our approach to extending classical planning with state constraints, in a way that is largely independent of the type of these constraints. However, for the examples and experiment benchmark problems in this paper, we will only use problems in which the state constraints are logical combinations of (linear or non-linear) equalities and inequalities over numeric variables. Section 3 presents detailed examples of how problems can be modelled in our framework. We address questions regarding the expressivity of the extended formalism in Section 4.

The two key concepts of classical planning world models are state variables and actions. In classical planning representations, such as propositional STRIPS or SAS+, states are assignments to a finite set of variables, each of which has a finite domain of values. Action preconditions (and the problem goal) are logical formulas over these variables, and each action's effect is to assign new values to some subset of variables; values of variables not reassigned persist. Our approach to incorporating state constraints is not dependent on the specific details of the classical planning formalism. For the purposes of presentation we will assume a formalism like SAS+.

Central to our approach is a partitioning of the state variables into two disjoint sets, called the *primary* and the *secondary* state variables, respectively. These sets of variables have different semantics: Primary state variables behave like classical planning state variables, in that they are assigned by actions' effects and persist from one state to the next when not reassigned. The values of secondary variables, on the other hand, are determined indirectly, by asserting the state constraints that they have to satisfy. These state constraints can involve both primary and secondary variables, and thus are the mechanism by which the two parts of the model interact. The constraints do not necessarily have a unique solution; thus, the secondary variables can also be characterised as "free" variables, which the planner can choose any value for in each state independently, as long as the chosen assignment satisfies the constraints. Because of this, we define a *state* as an assignment s of values to the primary state variables only:

Definition 1. *Let V_P be the set of primary state variables, and for each variable $v \in V_P$, let $D(v)$ be the domain of v, i.e., its set of possible values. A* state*, s, is a mapping from V_P such that $s(v) \in D(v)$ for all $v \in V_P$. For a formula φ over the primary variables, we also write $s(\varphi)$ for the value of φ in s.*

Moreover, the constraints do not necessarily have any solution for every valuation of the primary state variables; thus, they act as implicit preconditions, forbidding the plan from visiting certain states. PDDL 2.2's axioms (Thiébaux et al., 2005), as well as recent proposals for extending planning models via semantic attachement (Dornhege et al., 2009; Piacentini et al., 2015) also make a division between primary state variables that are essentially classical and secondary state variables that are derived from those; however, they require that the secondary variables are a deterministic function of the primary state[3] and thus do not admit either free variables or implicit preconditions as our formalism does.

A state (indeed, any partial assignment s to the primary state variables V_P) can also be specified by a set of constraints, $\{v = s(v) \mid v \in V_P\}$, or, equivalently, a conjunctive formula, $\bigwedge_{v \in V_P} v = s(v)$. We denote this constraint set by $C_P(s)$. In other words, $C_P(s)$ is a function that maps each state to a set of constraints satisfiable by that state only. We will make use of this to define satisfaction of state constraints below.

State constraints can appear in action preconditions, in the goal, and in a designated set C_{inv} of *invariant constraints*. The invariant constraints must be satisfied in every state visited by the plan. Thus, we say that a state is *valid* iff it is possible to satisfy the invariant constraints given the state's assignment to the primary variables.

3. This is how Dornhege et al. (2009) describe their integration of semantic attachements. See Section 8 for a longer discussion.

Definition 2. *A state s is* valid *if and only if $C_P(s) \cup C_{\mathsf{inv}}$ is satisfiable.*

Invariant constraints can serve two purposes: they define (sometimes uniquely, sometimes not) the values of secondary variables, contingent on the values of primary variables, and they impose constraints on the primary variables (sometimes indirectly by constraining the secondary variables) as required by the domain. For example, in power systems, the invariant constraints include both the equations that determine power flows, and operational constraints such as staying within line and generator capacity limits. There can be multiple power flow solutions for the same network configuration (cf. Section 3.2). Note that PDDL 2.2's axioms only serve the first of these two functions: stratification of the axioms ensures that in every state (that is, assignment of the primary variables, cf. Definition 1) there is an assignment of the derived predicates satisfying the axioms (Thiébaux et al., 2005); furthermore, this assignment is unique. Thus, axioms do not partition states into valid and invalid.

Action preconditions and the goal condition can be over both primary and secondary variables. We will express them using *partitioned conditions*, defined below. In principle, these conditions could be defined simply as formulas in the language of state constraints, which are also defined over both sets of variables. However, for algorithmic reasons, which will become apparent in Section 5, it is advantageous to distinguish the part that is a "simple" condition on the primary variables only.

Definition 3. *Let V_P and V_S be the set of primary and secondary state variables, respectively.*

A simple condition *is a conjunction of variable–value equalities or inequalities, without repeated variables.*

A partitioned condition *is a pair (φ_P, φ_S), where φ_P is a simple condition over the primary variables V_P and φ_S a set of constraints over $V_P \cup V_S$. The condition (φ_P, φ_S)* holds in state s *iff*

(i) $s(\varphi_P) = true$ *and*

(ii) $C_P(s) \cup \varphi_S \cup C_{\mathsf{inv}}$ *is satisfiable.*

By analogy with the partitioning of the state variables, we refer to the two parts of the partitioned action preconditions and goal as the primary and secondary preconditions and goal, respectively. Note, however, that the secondary condition can refer to both sets of variables, and that there is no inherent restriction on the form it takes. In fact, the formalism, as defined so far, even allows the secondary condition to be an arbitrary logical formula over primary variables only. (In the following subsection, we will introduce a specific, and somewhat restricted, form of secondary constraints; however, we will show in Section 4 that also this restricted form allows for polynomially encoding arbitrary formulas over the primary variables.) Several classical planning formalisms, including STRIPS, PSN, and SAS+, restrict action preconditions and goals to be simple conditions, or even more limited forms (Bäckström & Nebel, 1995; Helmert, 2009).

State constraints in the set $\varphi_S \cup C_{\mathsf{inv}}$ are formulas over variables in V_P and V_S. Recall that $C_P(s)$ is a set of constraints that restrict the values of variables in V_P to exactly the values they have in state s. Hence, taking the union of $C_P(s)$ and $\varphi_S \cup C_{\mathsf{inv}}$ is equivalent

to substituting the value $s(v)$ for each primary variable v in $\varphi_S \cup C_{\mathsf{inv}}$ and checking the satisfiability of the resulting formula. (The reason why item (*ii*) of Definition 3 is written the way it is is that this formulation extends naturally to relaxed states, in which state variables can have a disjunction of values. We define problem relaxations in Section 5.) An example illustrating Definition 3 can be found in Section 3.1 (page 381). It follows from Definition 3 that a partitioned condition can only hold in a valid state. In fact, a state s is valid if and only if the partitioned condition $(\mathit{true}, \emptyset)$ holds in s, because this is equivalent to requiring that $C_P(s) \cup C_{\mathsf{inv}}$ is satisfiable.

As usual, an action a is applicable in state s iff its precondition, $\mathrm{pre}(a) = (\mathrm{pre}_P(a), \mathrm{pre}_S(a))$ holds in s, according to Definition 3 above. An action's effects, $\mathrm{eff}(a)$, is a partial assignment of the primary variables, i.e., a set of atomic effects $v := e$, where $v \in V_P$ and e is an element in the domain of v. As usual, applying action a in state s results in a state s' such that $s'(v) = e$ if $v := e \in \mathrm{eff}(a)$ and $s'(v) = s(v)$ if v is not mentioned in the effects of a. We write $s' = \mathrm{apply}(a, s)$ for the state that results from applying action a in state s.

Note that we do not consider actions with conditional effects, or effects in which the value assigned to a variable is a function of the state that the action is applied in. Incorporating these does not present any conceptual difficulty, but several practical ones. Conditional effects can be defined by associating an effect condition, $\mathrm{cond}(v := e)$, with each atomic effect of an action, where $\mathrm{cond}(v := e)$ is a partitioned condition, and, as usual, the effect $v := e$ occurs iff $\mathrm{cond}(v := e)$ holds in the state where the action is applied. This, however, fails to consider consistency between secondary effect conditions. For example, the partitioned conditions $\varphi = (\mathit{true}, \{x < 0\})$ and $\psi = (\mathit{true}, \{x \geq 0\})$ may both hold, according to Definition 3, if neither $x < 0$ nor $x \geq 0$ is contradicted by the invariant constraints, but they cannot hold *simultaneously*. Recall that in our model, the planner is free to choose the values of secondary variables, subject to constraints. Hence, applying an action with conditional effects would require the planner to select a subset of effects to fire, such that the union of all selected effects' conditions and the negation of all non-selected effects' conditions is jointly satisfiable (together with $C_P(a) \cup C_{\mathsf{inv}}$). Since we have not implemented any support for conditional effects, we leave them out of the remainder of the paper.

2.1 Switched Constraints

Thus far, we have made *no* assumption about the syntactic form that constraints take, or even the domain of the secondary variables. We now introduce a particular form of state constraints.

The primary state variables are classical finite-domain variables, same as in the SAS+ formalism (Bäckström & Nebel, 1995), while the secondary variables are not so restricted. In this paper, we apply our framework to numeric (real- or rational-valued) secondary variables and constraints that are logical combinations of (linear or non-linear) equations or inequalities. However, secondary variables can be of any type, as long as we have a solver capable of reasoning about the consistency of constraints over variables of that type. As mentioned in the introduction, the derived predicates in PDDL version 2.2 (Thiébaux et al., 2005) may be viewed as an instance of this framework, where the secondary variables are propositional and the constraints are the axioms of a logic program. Secondary variables may even be complex objects, such as, for instance, sets (cf., e.g., Gregory et al., 2012). In

fact, nothing prevents us from having several, disjoint, sets of secondary state variables, of different types, with different types of constraints applied over each set.

This means that state constraints must combine both conditions on discrete finite-domain (primary) variables, and relations over variables of arbitrary types. Switched constraints (Ivankovic et al., 2014) are a syntactically restricted form of constraints which simplifies reasoning over this combination.

Definition 4. *Let V_P and V_S be the set of primary and secondary state variables, respectively. A* switched constraint *is a logical implication, $\varphi \rightarrow \gamma$, where φ is a simple formula (i.e., a conjunction of variable–value equalities or inequalities without repeated variables) over variables in V_P, and γ is a constraint over some subset of V_S.*

Examples of switched constraints can be found in the domain formulations in Section 3. There are, of course, many other syntactic forms that could be used to write constraints over the combined primary and secondary variables. The usefulness of switched constraints is that they allow us to separate reasoning about the primary condition and the satisfiability of the secondary condition. Thus, the latter can be given over to a solver that does not need to be capable of dealing with logical conditions over discrete variables.

Definition 5. *A switched constraint $\varphi \rightarrow \gamma$ is* active *in state s iff $s(\varphi) = true$. Given a set of switched constraints C, the* active set *of C in state s is*

$$\text{active}(C, s) = \{\gamma \mid \varphi \rightarrow \gamma \in C, s(\varphi) = \mathit{true}\}.$$

Proposition 6. *Let V_P and V_S be sets of primary and secondary state variables, respectively. Let C be a set of switched constraints, and s a state. Then there is an assignment to $V_P \cup V_S$ satisfying $C_P(s) \cup C$ if and only if there is an assignment to V_S satisfying* $\text{active}(C, s)$.

Proof. "if": Let σ be an assignment to V_S satisfying $\text{active}(C, s)$. Extend σ to an assignment σ' to $V_P \cup V_S$ by setting $\sigma'(v) = s(v)$ for all $v \in V_P$. Clearly σ' satisfies $C_P(s)$. Let $\varphi \rightarrow \gamma$ be a switched constraint in C. If $s(\varphi) = \mathit{true}$ then $\gamma \in \text{active}(C, s)$ so σ' satisfies γ (by assumption) and therefore σ' satisfies $\varphi \rightarrow \gamma$. If $s(\varphi) = \mathit{false}$ then φ is false also under σ' (by construction) so σ' satisfies $\varphi \rightarrow \gamma$.

"only if": If there is an assignment to $V_P \cup V_S$ satisfying $C_P(s) \cup C$, then by Definition 5 this assignment satisfies $\text{active}(C, s)$, and its restriction to V_s does as well since $\text{active}(C, s)$ only involves secondary variables. □

Corollary 7. *A partitioned condition (φ_P, φ_S) holds in state s if and only if $s(\varphi_P) = true$ and $\text{active}(\varphi_S \cup C_{\mathsf{inv}}, s)$ is satisfiable.*

Note that the active set contains only the consequent (right-hand side) of the implications whose triggering conditions are true. This means that switched constraints allow us to decide if a partitioned condition holds in a state by evaluating primary conditions and testing the satisfiability of a set of secondary conditions only. In some cases, the latter can be done in polynomial time. For the example and benchmark problems in this paper we instantiate our framework with numeric (real- or rational-valued) secondary variables and state constraints that are switched constraints in which the right-hand side is an equality

$g(x_1, \ldots, x_n) = 0$ or inequality $g(x_1, \ldots, x_n) \geq 0$, where $g(x_1, \ldots, x_n)$ is a function over the numeric variables. When this function is linear, we say these are *linear switched constraints*. In this case, all it takes to decide consistency is a linear equations solver.

2.2 The Planning Problem

We now have all the elements needed to define what a planning problem, and a solution plan, are in the extended formalism. Again, this definition is independent of the form of state constraints, and of the type of secondary variables.

Definition 8. *A planning problem P consists of:*

- *A set V_P of primary variables. Each variable $v \in V_P$ has an associated finite domain $D(v)$ of values.*
- *A set V_S of secondary variables.*
- *A set A of actions, each action a defined by:*
  - *a partitioned precondition* $\mathrm{pre}(a) = (\mathrm{pre}_P(a), \mathrm{pre}_S(a))$*, and*
  - *an effect* $\mathrm{eff}(a)$*, which is a set of assignments of values to primary state variables.*
- *A set C_{inv} of invariant constraints.*
- *An initial state s_0, assigning values to all variables in V_P, such that s_0 is valid.*
- *A partitioned goal condition $G = (G_P, G_S)$.*

An action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ induces a corresponding state sequence $\langle s_0, s_1, \ldots, s_n \rangle$, where $s_i = \mathrm{apply}(a_i, s_{i-1})$ is the result of applying action a_i in state s_{i-1}. π is a plan *iff each state s_i in the sequence is valid, each action a_i is applicable in s_{i-1}, and G holds in s_n.*

In this paper, we only consider the additive, state-independent action cost objective function. That is, each action $a \in A$ has an associated non-negative constant cost, $\mathrm{cost}(a)$, and the cost of a plan π is the sum of the costs of its actions. An optimal plan for a planning problem P is a plan for P whose cost is minimum among all plans for P. Previously, we explored also actions with state-dependent costs, meaning the cost of an action may be a non-constant function of the state that the action is applied in (Ivankovic et al., 2014). However, this makes the cost-optimal planning problem significantly harder, and our approach to handling state-dependent action costs was not very effective. Hence, we leave that extension out of this paper. Geißer et al. (2015, 2016) have since presented better approaches to creating both non-admissible and admissible heuristics for classical planning with state-dependent action costs. It is not clear whether their approach generalises from the classical case to our extended formalism. We leave this question for future investigation.

3. Domain Examples

Having defined an extended planning formalism, it is natural to ask whether this formalism is more expressive than classical planning. We address this question in Section 4. But first, we illustrate the power and limitations of the formalism by presenting four examples of

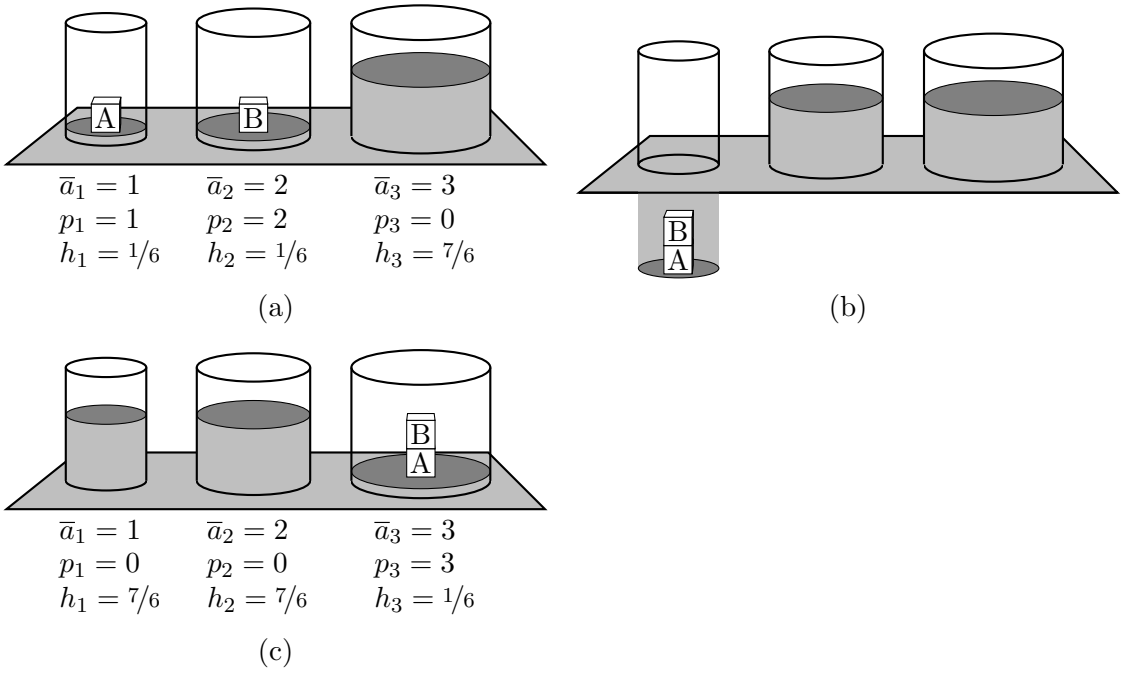


Figure 1: An example of the Hydraulic Blocks World domain. (a) A valid (initial) state: The weight of block A is 1 and the weight of block B, which sits on a twice as large area, is 2, causing pistons 1 and 2 to balance at the same height while the empty piston 3 rises higher. The total volume of fluid is $\bar{v} = 4$. (b) An invalid state: Placing B on A in cylinder 1, the combined weight causes the piston to fall through the bottom of the cylinder. (c) A valid goal state: Placing the total weight of A and B on the larger piston 3 makes it possible to counterbalance the weight with smaller fluid columns in the other cylinders.

domain models with numeric state constraints. As a notational convention throughout the paper, constants are distinguished from variables by an overline bar ($\bar{c}$ vs v).

3.1 Hydraulic Blocks World

Our first domain is intended mostly as a didactical example, illustrating how the primary and secondary parts of the model interact, and in particular how state constraints can restrict the set of valid plans. Consider a variant of Blocks World with a fixed number $\overline{m}$ of towers and $\overline{n}$ blocks: Each tower k sits on a piston inside a vertical cylinder with area $\overline{a}_k$, rising from a sealed reservoir of hydraulic fluid, as illustrated in Figure 1(a). Each block i has a weight $\overline{w}_i$. The height of each piston is determined by the total weight of the blocks in each cylinder and their areas, observing the law that the pressure that each column exerts on the reservoir must be equal and that the total volume of fluid remains constant. The goal is, as usual, to arrange some of the blocks into a given configuration. The main constraint,

in addition to the normal rules of the Blocks World domain, is that no piston may ever go above the top or below the bottom of its cylinder.

Actions are the usual pickup, putdown, unstack and stack, augmented to indicate what cylinder the block is moved to or from. For example, unstack(i, j, k) takes block i off block j in cylinder k. The primary state variables are pos_i, in_i, clear_i, and holding, where i is a block. Variable pos_i represents the position of block i, and its domain consists of the set of pistons, other blocks, and the constant in-hand. Variable in_i represents the cylinder block i is in, with domain $\{1, \ldots, \overline{m}\} \cup \{\mathsf{none}\}$. The Boolean variable clear_i represents whether block i is clear. Finally, the variable holding represents the block held, if any, and its domain consists of the set of blocks and none.

The preconditions and effects of actions on the primary variables are as expected. For example, unstack(i,j,k) requires $\mathsf{pos}_i = j$, $\mathsf{in}_i = k$, $\mathsf{clear}_i = \mathit{true}$ and holding $=$ none, and causes $\mathsf{pos}_i =$ in-hand, holding $= i$, $\mathsf{in}_i =$ none, $\mathsf{clear}_i = \mathit{false}$ and $\mathsf{clear}_j = \mathit{true}$. Actions have no secondary preconditions.

The key secondary variable is the height h_k of the fluid column in each cylinder k, and the main safety constraint is that this variable remains above 0 and below the height $\bar{l}_k$ of the cylinder:

$$0 \leq h_k \leq \bar{l}_k \tag{HBW.a}$$

for $k = 1, \ldots, \overline{m}$. (Note that this is actually a switched constraint, whose triggering condition is *true*. We omit the trigger for such constraints to simplify notation.) The total weight of the tower of blocks in cylinder k is represented by a secondary variable p_k. To compute p_k, we use secondary variables $p_{i,k}$, $i = 0, \ldots, \overline{n}$, $k = 1, \ldots, \overline{m}$, representing the contribution that block i makes to the total weight in cylinder k. $p_{i,k}$ is either 0, if block i is not in cylinder k, or the weight of the block, $\overline{w}_i$, if it is. This is enforced by the following switched constraints:

$$\mathsf{in}_i \neq k \rightarrow p_{i,k} = 0 \tag{HBW.b.i}$$

$$\mathsf{in}_i = k \rightarrow p_{i,k} = \bar{w}_i \tag{HBW.b.ii}$$

$$p_k = \sum_{i=1}^{\overline{n}} p_{i,k} \tag{HBW.b.iii}$$

$$0 \leq p_{i,k} \leq \bar{w}_i \qquad i = 1, \ldots, \overline{n}, k = 1, \ldots, \overline{m} \tag{HBW.b.iv}$$

Constraint (HBW.b.iv) is redundant, since it is implied by (HBW.b.i–HBW.b.ii). However, as we will see in the next section, adding redundant constraints to the secondary model can improve the inference power of relaxations. Now h_k can be determined via the following system of equations, which state that (HBW.c) the total amount of fluid $\bar{v}$ in the cylinders is constant, (HBW.d) the force f_k at the bottom of cylinder k is proportional to the weight p_k of the tower of blocks plus the weight of the fluid column in the cylinder (the fluid density $\bar{\rho}$ times the fluid's volume, where $\bar{a}_k$ is the cylinder's cross-sectional area) (HBW.d), and (HBW.e) the pressure (force per unit area) at the bottom of a cylinder is the same for each

cylinder:

$$\sum_{k=1}^{\overline{m}} \bar{a}_k h_k = \bar{v} \qquad k = 1, \ldots, \overline{m} \qquad \text{(HBW.c)}$$

$$f_k = p_k + \bar{\rho}\bar{a}_k h_k \qquad k = 1, \ldots, \overline{m} \qquad \text{(HBW.d)}$$

$$\frac{f_k}{\bar{a}_k} = \frac{f_{k+1}}{\bar{a}_{k+1}} \qquad k = 1, \ldots, \overline{m} - 1 \qquad \text{(HBW.e)}$$

A complete description of the sample problem shown in Figure 1(a) is given in Figure 2.

The initial state is as shown in Figure 1(a). The set of active invariant constraints (right-hand sides only) in the initial state are $p_{\mathsf{A},1} = 1$, $p_{\mathsf{A},2} = 0$, $p_{\mathsf{A},3} = 0$, $p_{\mathsf{B},1} = 0$, $p_{\mathsf{B},2} = 2$, $p_{\mathsf{B},3} = 0$, and constraints (HBW.a), (HBW.b.iii–HBW.e) whose triggering conditions are constantly true. This set is satisfiable – a solution is shown in Figure 1(a) – so the initial state is valid.

The goal is to place block B on block A. Considering only the primary (classical) part of the model, this can be achieved by picking up B and stacking it on A. However, the resulting state – illustrated in Figure 1(b) – is not valid, because the combined weight of A and B placed on the small area in cylinder 1 exerts too much pressure; to counterbalance it, the columns in cylinders 2 and 3 would need more fluid than the total volume, $\bar{v} = 4$. The active constraints with non-constant triggers are $p_{\mathsf{A},1} = 1$, $p_{\mathsf{A},2} = 0$, $p_{\mathsf{A},3} = 0$, $p_{\mathsf{B},1} = 2$, $p_{\mathsf{B},2} = 0$, $p_{\mathsf{B},3} = 0$, which yields $p_1 = 3$, $p_2 = 0$, $p_3 = 0$ (from HBW.b.iii) which with (HBW.d–HBW.e) yields $(3+1h_1)/1 = (0+2h_2)/2 = (0+3h_3)/3$, which together with (HBW.c) is only satisfiable when $h_1 < 0$, contradicting (HBW.a).

A valid goal state is shown in Figure 1(c). Here, the weight of the tower is placed in cylinder 3, which has a larger area $\bar{a}_3 = 3$, making it possible to counterbalance the weight with lower columns in cylinders 1 and 2. This state is reachable by moving A from piston 1 to piston 3, then moving B onto A. All intermediate states in this plan are valid.

3.2 Switching Problems in Power Networks

Our second domain exemplifies the kind of useful problem that our approach enables planning to address. The problem we consider is to reconfigure a power network, by opening or closing line switches. This can be for several purposes: an example known in the planning literature is power supply restoration (PSR), where the goal is to isolate known faulty parts and resupply disconnected loads. For another example, we may want to isolate a particular line or generator, that is being phased out for servicing, while maintaining supply to all loads at every intermediate state of the plan. Line switching is also used to reconfigure a network to minimise line losses or balance load. The PSR problem appeared as a benchmark of the 2004 International Planning Competition (Hoffmann, Edelkamp, Thiébaux, Englert, dos S. Liporace, & Trüg, 2006). In contrast to the IPC benchmark, our formalism allows us to model numeric nonlinear power flows as well as capacity and voltage constraints, which is an essential requirement to make the model realistic.

The network is a graph $\langle \mathcal{B}, \mathcal{L} \rangle$ whose nodes are buses $i \in \mathcal{B}$ and edges $(i, j) \in \mathcal{L}, i < j$ are power lines equipped with switches that, when open, disable the edge. A subset of buses ($\mathcal{G}$) are generators, which supply the network with power. A bus is *fed* iff there is a path

State variables:		Domain	
(primary)	pos_i	$\{\mathsf{A}, \mathsf{B}, 1, 2, 3, \mathsf{in\text{-}hand}\}$	for $i \in \{\mathsf{A}, \mathsf{B}\}$
(primary)	in_i	$\{1, 2, 3, \mathsf{none}\}$	for $i \in \{\mathsf{A}, \mathsf{B}\}$
(primary)	clear_i	$\{\mathit{true}, \mathit{false}\}$	for $i \in \{\mathsf{A}, \mathsf{B}, 1, 2, 3\}$
(primary)	holding	$\{\mathsf{A}, \mathsf{B}, \mathsf{none}\}$	
(secondary)	$p_{i,k}$	$\mathbb{R}$	for $i \in \{\mathsf{A}, \mathsf{B}\}$, $k \in \{1, 2, 3\}$
(secondary)	p_k, h_k, f_k	$\mathbb{R}$	for $k \in \{1, 2, 3\}$

Actions:

$\mathsf{pickup}(i, k)$ — for $i \in \{\mathsf{A}, \mathsf{B}\}$, $k \in \{1, 2, 3\}$
- pre: $(\mathsf{pos}_i = k \wedge \mathsf{in}_i = k \wedge \mathsf{clear}_i = \mathit{true} \wedge \mathsf{holding} = \mathsf{none},\ \emptyset)$
- eff: $\mathsf{pos}_i := \mathsf{in\text{-}hand}, \mathsf{in}_i := \mathsf{none}, \mathsf{clear}_i := \mathit{false}, \mathsf{clear}_k := \mathit{true}, \mathsf{holding} = i$

$\mathsf{putdown}(i, k)$ — for $i \in \{\mathsf{A}, \mathsf{B}\}$, $k \in \{1, 2, 3\}$
- pre: $(\mathsf{pos}_i = \mathsf{in\text{-}hand} \wedge \mathsf{in}_i = \mathsf{none} \wedge \mathsf{clear}_k = \mathit{true} \wedge \mathsf{holding} = i,\ \emptyset)$
- eff: $\mathsf{pos}_i := k, \mathsf{in}_i := k, \mathsf{clear}_i := \mathit{true}, \mathsf{clear}_k := \mathit{false}, \mathsf{holding} := \mathsf{none}$

$\mathsf{unstack}(i, j, k)$ — for $i, j \in \{\mathsf{A}, \mathsf{B}\}$, $i \neq j$, $k \in \{1, 2, 3\}$
- pre: $(\mathsf{pos}_i = j \wedge \mathsf{in}_i = k \wedge \mathsf{clear}_i = \mathit{true} \wedge \mathsf{holding} = \mathsf{none},\ \emptyset)$
- eff: $\mathsf{pos}_i := \mathsf{in\text{-}hand}, \mathsf{in}_i := \mathsf{none}, \mathsf{clear}_i := \mathit{false}, \mathsf{clear}_j := \mathit{true}, \mathsf{holding} := i$

$\mathsf{stack}(i, j, k)$ — for $i, j \in \{\mathsf{A}, \mathsf{B}\}$, $i \neq j$, $k \in \{1, 2, 3\}$
- pre: $(\mathsf{pos}_i = \mathsf{in\text{-}hand} \wedge \mathsf{in}_i = \mathsf{none} \wedge \mathsf{in}_j = k \wedge \mathsf{clear}_j = \mathit{true} \wedge \mathsf{holding} = i,\ \emptyset)$
- eff: $\mathsf{pos}_i := j, \mathsf{in}_i := k, \mathsf{clear}_i := \mathit{true}, \mathsf{clear}_j := \mathit{false}, \mathsf{holding} := \mathsf{none}$

Invariant constraints:

$0 \le h_1 \le 2$		$0 \le h_2 \le 2$	$0 \le h_3 \le 2$	(a)
$\mathsf{in}_\mathsf{A} \neq 1 \rightarrow p_{\mathsf{A},1} = 0$		$\mathsf{in}_\mathsf{A} \neq 2 \rightarrow p_{\mathsf{A},2} = 0$	$\mathsf{in}_\mathsf{A} \neq 3 \rightarrow p_{\mathsf{A},3} = 0$	(b.i)
$\mathsf{in}_\mathsf{B} \neq 1 \rightarrow p_{\mathsf{B},1} = 0$		$\mathsf{in}_\mathsf{B} \neq 2 \rightarrow p_{\mathsf{B},2} = 0$	$\mathsf{in}_\mathsf{B} \neq 3 \rightarrow p_{\mathsf{B},3} = 0$	
$\mathsf{in}_\mathsf{A} = 1 \rightarrow p_{\mathsf{A},1} = 1$		$\mathsf{in}_\mathsf{A} = 2 \rightarrow p_{\mathsf{A},2} = 1$	$\mathsf{in}_\mathsf{A} = 3 \rightarrow p_{\mathsf{A},3} = 1$	(b.ii)
$\mathsf{in}_\mathsf{B} = 1 \rightarrow p_{\mathsf{B},1} = 2$		$\mathsf{in}_\mathsf{B} = 2 \rightarrow p_{\mathsf{B},2} = 2$	$\mathsf{in}_\mathsf{B} = 3 \rightarrow p_{\mathsf{B},3} = 2$	
$p_1 = p_{\mathsf{A},1} + p_{\mathsf{B},1}$		$p_2 = p_{\mathsf{A},2} + p_{\mathsf{B},2}$	$p_3 = p_{\mathsf{A},3} + p_{\mathsf{B},3}$	(b.ii)
$0 \le p_{\mathsf{A},1} \le 1$		$0 \le p_{\mathsf{A},2} \le 1$	$0 \le p_{\mathsf{A},3} \le 1$	(b.iv)
$0 \le p_{\mathsf{B},1} \le 2$		$0 \le p_{\mathsf{B},2} \le 2$	$0 \le p_{\mathsf{B},3} \le 2$	
$1h_1 + 2h_2 + 3h_3 = 4$				(c)
$f_1 = p_1 + 1h_1$		$f_2 = p_2 + 2h_2$	$f_3 = p_3 + 3h_3$	(d)
$f_1/1 = f_2/2 = f_3/3$				(e)

Initial state: $\mathsf{pos}_\mathsf{A} = 1, \mathsf{pos}_\mathsf{B} = 2, \mathsf{in}_\mathsf{A} = 1, \mathsf{in}_\mathsf{B} = 2, \mathsf{clear}_\mathsf{A} = \mathit{true}, \mathsf{clear}_\mathsf{B} = \mathit{true}, \mathsf{clear}_1 = \mathit{false}, \mathsf{clear}_2 = \mathit{false}, \mathsf{clear}_3 = \mathit{true}, \mathsf{holding} = \mathsf{none}$

Goal: $(\mathsf{pos}_\mathsf{B} = \mathsf{A},\ \emptyset)$

Figure 2: Formal description of the Hydraulic Blocksworld problem shown in Figure 1. Note that in this domain, the secondary part of all actions' preconditions and the goal is empty. Parameters are $\overline{w}_\mathsf{A} = 1$, $\overline{w}_\mathsf{B} = 2$, $\overline{a}_1 = 1$, $\overline{a}_2 = 2$, $\overline{a}_3 = 3$, $\overline{l}_1 = 2$, $\overline{l}_2 = 2$, $\overline{l}_3 = 2$, $\overline{v} = 4$ and $\rho = 1$.

(of lines with closed switches) to it from a generator bus. In this case, its entire load must be supplied. Being fed is modelled by a secondary variable $f_i \in \{0, 1\}$ for each bus $i \in \mathcal{B}$.

We model an alternating current (AC) network. This allows for a more realistic model, but at the price of having non-linear equations governing the power flow. The simpler direct current (DC) approximation yields a linear set of equations. In the AC model, electrical quantities (power, voltage, etc) are complex, so we have to distinguish a real and imaginary component of each variable. The imaginary component of power is known as reactive power. Each bus supports constant real and reactive loads $\bar{p}_{Li}$ and $\bar{q}_{Li}$, respectively. Each bus has a variable voltage whose real and imaginary components are v_{Ri} and v_{Ii}. The square of the voltage magnitude, $v_{Ri}^2 + v_{Ii}^2$, at each bus is constrained to remain in an, often quite narrow, interval around a nominal value. Each generator bus $i \in \mathcal{G}$ supplies a variable amount of real and reactive power, p_{Gi} and q_{Gi}, to the network. The generators have capacity constraints that limit maximum real and reactive power output. In the PSR problem, we also distinguish a subset $\mathcal{F}$ of faulty buses.

Each line (i, j) is characterised by a constant admittance, whose real and imaginary components are the conductance $\bar{g}_{ij}$ and susceptance $\bar{b}_{ij}$, respectively. The real and reactive power flows on each line need to be modelled in both directions, leading to 4 variables: p_{Fij} and q_{Fij} for the real and reactive power flows from i to j, and p_{Tij} and q_{Tij} for the flows in the reverse direction. This is because, due to line losses, $p_{Fij} \neq -p_{Tij}$ and $q_{Fij} \neq -q_{Tij}$. Lines have thermal limits that constrain the apparent power flow in both directions; this means that both $p_{Fij}^2 + q_{Fij}^2$ and $p_{Tij}^2 + q_{Tij}^2$ are bounded above.

The only primary variables are the line switch positions y_{ij}. Opening/closing a switch toggles y_{ij} between *false* (open) and *true* (closed). The planner also controls each generator's power output. Rather than using explicit actions, we model this using the freedom our formalism gives the planner to assign the underconstrained secondary variables p_{Gi} and q_{Gi}, which represent the active and reactive power produced at generator bus $i \in \mathcal{G}$. There can be multiple solutions to the active state constraints.

There are three main types of invariant constraints (Thiébaux, Coffrin, Hijazi, & Slaney, 2013). The first define the line power flows, as a function of the voltages and admittances. Of course open lines have no flow, hence the following switched constraints for all $(i, j) \in \mathcal{L}$:

$$y_{ij} = \mathit{true} \rightarrow p_{Fij} = \bar{g}_{ij}(v_{Ri}^2 + v_{Ii}^2) - \bar{g}_{ij}(v_{Ri}v_{Rj} + v_{Ii}v_{Ij}) - \bar{b}_{ij}(v_{Ii}v_{Rj} - v_{Ri}v_{Ij}) \quad \text{(PSR.a.i)}$$

$$y_{ij} = \mathit{true} \rightarrow p_{Tij} = \bar{g}_{ij}(v_{Rj}^2 + v_{Ij}^2) - \bar{g}_{ij}(v_{Rj}v_{Ri} + v_{Ij}v_{Ii}) - \bar{b}_{ij}(v_{Ij}v_{Ri} - v_{Rj}v_{Ii}) \quad \text{(PSR.a.ii)}$$

$$y_{ij} = \mathit{true} \rightarrow q_{Fij} = \bar{b}_{ij}(v_{Ri}^2 + v_{Ii}^2) + \bar{b}_{ij}(v_{Ri}v_{Rj} + v_{Ii}v_{Ij}) - \bar{g}_{ij}(v_{Ii}v_{Rj} - v_{Ri}v_{Ij}) \quad \text{(PSR.a.iii)}$$

$$y_{ij} = \mathit{true} \rightarrow q_{Tij} = \bar{b}_{ij}(v_{Rj}^2 + v_{Ij}^2) + \bar{b}_{ij}(v_{Rj}v_{Ri} + v_{Ij}v_{Ii}) - \bar{g}_{ij}(v_{Ij}v_{Ri} - v_{Rj}v_{Ii}) \quad \text{(PSR.a.iv)}$$

$$y_{ij} = \mathit{false} \rightarrow p_{Fij} = q_{Fij} = p_{Tij} = q_{Tij} = 0 \quad \text{(PSR.a.v)}$$

Constraints of the second type encode the flow propagation through the network using Kirchhoff's Law (flow conservation at the buses (PSR.b.i-PSR.b.ii)) whilst enforcing that no faulty bus ($i \in \mathcal{F}$) is fed (PSR.c.i), that non-faulty generator buses ($i \in \mathcal{G} \setminus \mathcal{F}$) are fed

(psr.c.ii), and that connected buses have the same fed status (psr.c.iii).

$$p_{Gi} - f_i\bar{p}_{Li} - \sum_{j:(i,j)\in\mathcal{L}} p_{Fij} + \sum_{j:(j,i)\in\mathcal{L}} p_{Tji} = 0 \qquad i \in \mathcal{B} \qquad \text{(psr.b.i)}$$

$$q_{Gi} - f_i\bar{q}_{Li} - \sum_{j:(i,j)\in\mathcal{L}} q_{Fij} + \sum_{j:(j,i)\in\mathcal{L}} q_{Tji} = 0 \qquad i \in \mathcal{B} \qquad \text{(psr.b.ii)}$$

$$f_i = 0 \qquad i \in \mathcal{F} \qquad \text{(psr.c.i)}$$

$$f_i = 1 \qquad i \in \mathcal{G} \setminus \mathcal{F} \qquad \text{(psr.c.ii)}$$

$$y_{ij} = \mathit{true} \rightarrow f_i = f_j \qquad (i,j) \in \mathcal{L} \qquad \text{(psr.c.iii)}$$

Finally, the following constraints encode limits on generation (psr.d.i-psr.d.ii), voltage magnitude (psr.e), and apparent power (psr.f.i-psr.f.ii).

$$0 \leq p_{Gi} \leq \bar{P}_{Gi} \qquad i \in \mathcal{G} \qquad \text{(psr.d.i)}$$

$$\bar{q}_{Gi} \leq q_{Gi} \leq \bar{Q}_{Gi} \qquad i \in \mathcal{G} \qquad \text{(psr.d.ii)}$$

$$\bar{v}_i^2 \leq v_{Ri}^2 + v_{Ii}^2 \leq \bar{V}_i^2 \qquad i \in \mathcal{G} \qquad \text{(psr.e)}$$

$$p_{Fij}^2 + q_{Fij}^2 \leq \bar{s}_{ij}^2 \qquad (i,j) \in \mathcal{L} \qquad \text{(psr.f.i)}$$

$$p_{Tij}^2 + q_{Tij}^2 \leq \bar{s}_{ij}^2 \qquad (i,j) \in \mathcal{L} \qquad \text{(psr.f.ii)}$$

A restoration plan is a sequence of switching operations, each changing the value of one variable y_{ij}. One objective is to resupply as much load as possible as fast as possible: if we plot the power supplied as a function of time (plan steps), the objective function that we wish to maximise is the area under this curve (Thiébaux et al., 2013). Unlike typical planning objectives (e.g., cost or makespan), this value can be strongly affected by reordering independent actions. Another objective is to minimise the deviation from the standard (pre-fault) network configuration. Both objectives can be expressed as a sum of state-dependent action costs. However, minimising plan length is a reasonable proxy, at least for the latter objective, and is much easier for planners to do. In this paper, we consider only this variant.

3.3 Multi-commodity Linehaul Transportation

Logistics problems have long been a staple planning benchmark. Our third example domain models a real-world multi-commodity transportation problem (Kilby, Abio, Guimarans, Harabor, Haslum, Mayer-Eichberger, Siddiqui, Thiébaux, & Urli, 2015).

Goods, of different types, need to be transported from a depot to customer locations, $1, \ldots, \bar{m}$. As a convention, we label the depot location 0. D_{ij}, $i, j \in \{0, \ldots, \bar{m}\}$ is the distance, along the road network, between locations i and j. Each customer i has a demand $\bar{q}_i^g$ for good type g. Transportation is done with a fleet of $\bar{n}$ trucks. Each truck k has a set $\bar{\mathcal{G}}_k$ of goods types that it can carry, a capacity $\bar{p}_k$, and a per-kilometer cost $\bar{c}_k$. In the problems we encounter, there are usually several trucks of the same type, i.e., with identical parameters. Also, there are only two goods types: ambient and chilled. Refrigerated trucks can carry both types, while non-refrigerated trucks can only carry ambient temperature goods. All trucks start at the depot and must return to the depot at the end of the plan, as

well as meet all customer demands. Because of time constraints, each truck can only make one tour (from depot to customers and back) in a plan.

In our model of this problem, all reasoning about goods delivery is done in the secondary model. Primary state variables are loc_k for each truck k, with domain $\{0, 0^\star, \ldots, \bar{m}\}$, representing the current location of the truck (0 means the truck has not yet left the depot, while $0^\star$ means it has returned to the depot). In addition, a Boolean variable $\mathsf{visited}_{k,i}$ keeps track of whether truck k has visited location i. The only action is $\mathsf{drive}(k, i, j)$, with precondition $\mathsf{loc}_k = i$, effect $\mathsf{loc}_k = j$ and $\mathsf{visited}_{k,j} = \mathit{true}$, and cost $\bar{c}_k \cdot D_{i,j}$.

For each truck k, customer location i, and goods type $g \in \{\mathrm{am}, \mathrm{ch}\}$, a secondary variable $d^g_{k,i}$ represents the amount of goods type g that truck k delivers to customer i. The following constraints ensure that trucks deliver only to locations that they visit, and that type and capacity restrictions are met:

$$\mathsf{visit}_{k,i} = \mathit{false} \rightarrow d^g_{k,i} = 0 \qquad k = 1, \ldots, \bar{n}, i = 1, \ldots, \bar{m}, \; g \in \{\mathrm{am}, \mathrm{ch}\} \qquad \text{(LH.a)}$$

$$\left(\sum_{i=1,\ldots,\bar{m}, g \in \{\mathrm{am},\mathrm{ch}\}} d^g_{k,i} \right) \leq \bar{p}_k \qquad k = 1, \ldots, \bar{n} \qquad \text{(LH.b)}$$

The goal of meeting customer demands is expressed by a secondary goal constraint:

$$\left(\sum_{k : g \in \bar{\mathcal{G}}_k} d^g_{k,i} \right) = \overline{q}^g_i \qquad i = 1, \ldots, \bar{m}, g \in \{\mathrm{am}, \mathrm{ch}\} \qquad \text{(LH.c)}$$

Demand and capacity values are integer, and for every problem instance there is a finite maximum. Hence, this problem can also be modelled as a classical planning problem, using only finite-domain variables. We explore the relative performance of our formulation and a purely classical formulation in Section 7.6.

3.4 The Counters Domain

The counters domain was invented by Francès and Geffner (2015), as a simple example to illustrate one of the flaws of heuristics based on delete relaxation (also called *monotonic relaxation*, cf. Section 5). They introduce state constraints into the construction of the relaxation to overcome it. We show how this domain can be modelled in our formalism, and later, in Sections 5.5 and 7.6, that this model also leads to a relaxation that is as powerful as that proposed by Francès and Geffner.

The domain features $\bar{n}$ counters, $X_1, \ldots, X_{\bar{n}}$, each ranging over integers $0, \ldots, \bar{m}$. Actions $\mathsf{inc}(i)$ and $\mathsf{dec}(i)$ increment and decrement, respectively, counter i by 1. Initial values of the counters can be all zero, all maximum, or random. The goal is $X_1 < X_2 \wedge X_2 < X_3 \wedge \ldots \wedge X_{\bar{n}-1} < X_{\bar{n}}$.

The failure of delete relaxation that this problem demonstrates is that it evaluates each goal conjunct in isolation. For example, if, in a relaxed state, the value of each of counters X_1, X_2 and X_3 is in $\{0, 1\}$, subgoals $X_1 < X_2$ and $X_2 < X_3$ are both satisfiable, but the conjunction of them is not.

In our model, primary state variables are $\bar{m}$ propositional variables, $p_{i,j}$, for each counter i and $j = 1, \ldots, \bar{m}$. The model represents $X_i = k$ with $p_{i,j} = \mathit{true}$ for $j = 1, \ldots, k$ and $p_{i,j} = \mathit{false}$ for $j = k+1, \ldots, \bar{m}$. Formulating the actions to maintain this representation is straightforward. Because we do not use conditional effects we need a separate action for each counter value. For example, action $\mathsf{inc}(i, j)$ (with $j > 0$), which increments counter i from $j-1$ to j, has primary precondition $p_{i,j-1} = \mathit{true}$ (except if $j = 1$) and $p_{i,j} = \mathit{false}$, and effect $p_{i,j} = \mathit{true}$. Each counter is also represented by a secondary variable, x_i. The following invariant constraints ensure that the primary and secondary representations agree:

$$p_{i,j} = \mathit{true} \rightarrow x_i \geq j \qquad i = 1, \ldots, \bar{n}, j = 1, \ldots, \bar{m} \qquad \text{(COUNTERS.i)}$$

$$p_{i,j} = \mathit{false} \rightarrow x_i \leq j - 1 \qquad \text{(COUNTERS.ii)}$$

$$0 \leq x_i \leq \bar{m} \qquad i = 1, \ldots, \bar{n} \qquad \text{(COUNTERS.iii)}$$

For example, if $p_{i,1} = \mathit{true}$ and $p_{i,2} = \mathit{false}$, (COUNTERS.i) and (COUNTERS.ii) force $1 \leq x_i \leq 2 - 1$, i.e., $x_i = 1$. Constraint (COUNTERS.iii) ensures that $x_i = 0$ when $p_{i,1} = \mathit{false}$ and that $x_i = \bar{m}$ when $p_{i,\bar{m}} = \mathit{true}$. The goal is expressed on the secondary variables. To account for the fact that counter values are integer, we write the subgoals as $x_i + 1 \leq x_{i+1}$.

4. Expressivity

We now return to the question whether the formalism with state constraints is more expressive than classical planning, and the complexity of solving problems expressed in it.

Our first observation is that the secondary part of any partitioned condition can be compiled into the primary part, albeit not necessarily into a simple formula over the primary variables.

Proposition 9. *Let (φ_P, φ_S) be a partitioned condition. There is a formula $\mathrm{F}(\varphi_S)$ over V_P such that for every state s, $s(\varphi_P \wedge \mathrm{F}(\varphi_S)) = \mathit{true}$ if and only if (φ_P, φ_S) holds in s.*

Proof. Let $\mathit{Models}(\varphi_S) = \{s \mid C_P(s) \cup \varphi_S \cup C_{\mathsf{inv}} \text{ is satisfiable } \}$. (φ_P, φ_S) holds in state s if and only if $s(\varphi_P) = \mathit{true}$ and $s \in \mathit{Models}(\varphi_S)$ (by Definition 3). Since states are assignments of values to the primary state variables, each of which has a finite domain of values (Definition 1), the set of possible states is finite (though exponentially large) and hence so is $\mathit{Models}(\varphi_S)$. Thus, there exists a finite-sized formula, $\mathrm{F}(\varphi_S)$, over V_P, that is true exactly in the states $\mathit{Models}(\varphi_S)$. This formula may be written simply as a disjunction of conjunctions of variable–value equalities each defining a complete state in $\mathit{Models}(\varphi_S)$, but more compact forms may also exist. Then $\varphi_P \wedge \mathrm{F}(\varphi_S)$ characterises exactly the states in which (φ_P, φ_S) holds. □

Thus, secondary conditions can be eliminated from action preconditions and the goal. This may introduce disjunctive (primary) conditions, but these can be compiled away following standard procedures (Nebel, 2000). But what of the invariant constraints? A formula $\mathrm{F}(C_{\mathsf{inv}})$ that characterises valid states in terms of the primary variables only can be constructed as in the proof of Proposition 9. Adding $\mathrm{F}(C_{\mathsf{inv}})$ to all action preconditions and to the goal ensures that a plan visits only valid states: each state except the last must be valid for the next action to be applicable, and the final state must be valid to satisfy the

goal. Hence, our formalism can be reduced to classical planning. The compilation given in the proof of Proposition 9 increases problem size exponentially. The proposition does not, however, prove that a more space-efficient compilation is not possible. Proposition 10 below, answers this question.

Recall that although in this paper we focus on switched constraints over real-valued secondary variables, our approach to extending classical planning with state constraints, as defined in Section 2, is independent of the constraint language. Proposition 9 holds no matter what kinds of secondary variables and constraints over them appear in the problem. This also implies that, independently of the size of the problem that results from compiling away state constraints, we cannot upper-bound the time complexity of performing the compilation, since it depends on the complexity of checking if a set of constraints are satisfiable in a given state. Depending on the constraint language, this may be tractable, NP-hard or worse, or not even be decidable. (Proposition 9 states that the formula $\mathrm{F}(\varphi_S)$ *exists*, not that it is computable.)

Our second observation is that we can encode complex conditions over the primary variables into secondary constraints, provided the constraint language is sufficiently expressive. In particular, within the formalism of linear switched constraints over real-valued secondary variables we can formulate action preconditions and goals that are equivalent to general formulas over the primary variables. Thus, the restriction that primary conditions are simple (conjunctions of variable–value equalities or inequalities) is not a true restriction on the expressivity of the formalism.

Proposition 10. *Let φ be any formula over the primary variables V_P. There exists a set of linear switched constraints C such that for every state s, $C_P(s) \cup C$ is satisfiable if and only if $s(\varphi) = true$. Moreover, the size of C is polynomial in the size of φ.*

Proof. Without loss of generality we can assume φ to be in negation normal form, since translation to this form does not increase the size of the formula more than polynomially.

We will introduce a secondary variable $0 \leq p_\psi \leq 1$ for every subformula ψ of φ, along with a set of constraints C' such that

$$\{p_\psi > 0\} \cup C' \cup C_P(s) \text{ is satisfiable if only if } s(\psi) = \mathit{true}. \qquad (\star)$$

The constraint set C claimed by the proposition is then given by $C = \{p_\varphi > 0\} \cup C'$. The construction of C' is as follows:

- For each atomic subformula of the form $v = e$, C' contains the two switched constraints $v = e \rightarrow p_{v=e} = 1$ and $v \neq e \rightarrow p_{v=e} = 0$.
- For each atomic subformula of the form $v \neq e$, C' contains the two switched constraints $v \neq e \rightarrow p_{v\neq e} = 1$ and $v = e \rightarrow p_{v\neq e} = 0$.
- For each conjunctive subformula $\psi = \chi_1 \wedge \ldots \wedge \chi_k$, C' contains the constraints $p_\psi \leq p_{\chi_1}, \ldots, p_\psi \leq p_{\chi_k}$.
- For each disjunctive subformula $\psi = \chi_1 \vee \ldots \vee \chi_k$, C' contains the constraint $p_\psi \leq p_{\chi_1} + \ldots + p_{\chi_k}$.

(The construction has some resemblance with well-known encodings of propositional logic into integer linear programs. Note, however, that the secondary variables here are real, *not* integer.) Both the number of constraints in C' and the number of terms in any expression that appears in one of them is bounded by a constant times the number of subformulas of φ, so the size of C is polynomial in that of φ. It remains to show that C' has property $(\star)$.

For the "if" part, let s be an arbitrary state and extend s to an assignment σ over primary variables and the secondary variables mentioned in C' by setting $\sigma(p_\psi) = 1$ if $s(\psi) = \mathit{true}$ and $\sigma(p_\psi) = 0$ if $s(\psi) = \mathit{false}$ for each subformula ψ. We will show that σ satisfies every constraint in C'. Thus, this assignment is a witness to the fact that $\{\sigma(p_\psi) > 0\} \cup C' \cup C_P(s)$ is satisfiable if $s(\psi) = \mathit{true}$.

If $\sigma(v = e) = \mathit{true}$ the constraint $v = e \rightarrow p_{v=e} = 1$ is satisfied because $\sigma(p_{v=e}) = 1$ (by construction) and the constraint $v \neq e \rightarrow p_{v=e} = 0$ is satisfied because $\sigma(v \neq e) = \mathit{false}$; if $\sigma(v = e) = \mathit{false}$ it is the other way around. Constraints $v \neq e \rightarrow p_{v\neq e} = 1$ and $v = e \rightarrow p_{v\neq e} = 0$ are analogous. The constraints $p_\psi \leq p_{\chi_1}, \ldots, p_\psi \leq p_{\chi_k}$ created for a conjunction $\psi = \chi_1 \wedge \ldots \wedge \chi_k$ are satisfied because $\sigma(p_\psi) = 1$ only if $\sigma(\psi) = \mathit{true}$ only if $\sigma(\chi_i) = \mathit{true}$ for each conjunct χ_i, in which case $\sigma(p_{\chi_i}) = 1$; when $\sigma(p_\psi) = 0$ the constraints are satisfied because all the indicator variables are bounded and must be greater than or equal to zero. Similarly, the constraint $p_\psi \leq p_{\chi_1} + \ldots + p_{\chi_k}$ created for a disjunction is satisfied because $\sigma(p_\psi) = 1$ only if $\sigma(\psi) = \mathit{true}$ only if $\sigma(\chi_i) = \mathit{true}$ for at least one disjunct χ_i, in which case $\sigma(p_{\chi_i}) = 1$ which makes also the sum at least 1, and if $\sigma(p_\psi) = 0$ because zero also lower-bounds the sum.

For the "only if" part, we proceed by a structural induction. As the first base case, consider an atomic subformula of the form $v = e$, and a state s such that $s(v = e) = \mathit{false}$. Then $\{\sigma(p_{v=e}) > 0\} \cup C' \cup C_P(s)$ contains $\{v = e', v \neq e \rightarrow p_{v=e} = 0, p_{v=e} > 0\}$, for some $e' \neq e$, which is clearly not satisfiable. The second base case, an atomic subformula of the form $v \neq e$, is analogous.

Consider a conjunctive formula, $\psi = \chi_1 \wedge \ldots \wedge \chi_k$. If $s(\psi) = \mathit{false}$ then $s(\chi_i) = \mathit{false}$ for at least one conjunct χ_i. By inductive assumption, this implies $\{p_{\chi_i} > 0\} \cup C' \cup C_P(s)$ is unsatisfiable. Since C' contains $p_\psi \leq p_{\chi_i}$, $p_\psi > 0$ implies $p_{\chi_i} > 0$ in any model for C', which means that $\{p_\psi > 0\} \cup C' \cup C_P(s)$ is also unsatisfiable.

Finally, consider a disjunctive formula, $\psi = \chi_1 \vee \ldots \vee \chi_k$. If $s(\psi) = \mathit{false}$ then $s(\chi_i) = \mathit{false}$ for every disjunct χ_i. By inductive assumption, this implies $\{p_{\chi_i} > 0\} \cup C' \cup C_P(s)$ is unsatisfiable. Thus, the sum $p_{\chi_1} + \ldots + p_{\chi_k}$ also cannot be greater than zero (since that would imply one of its parts is), and thus $\{p_\psi > 0\} \cup C' \cup C_P(s)$ is also unsatisfiable. □

Nebel (2000) analysed the complexity of compilations between classical planning formalisms spanning from (propositional) ADL to (propositional) STRIPS. Two of the implications of his results are that compiling away general Boolean formulas (in action preconditions and the goal) requires either a worst-case exponential increase in the size of the problem, or a super-linear (but still polynomial) increase in plan length. This, together with Proposition 10, implies that extending classical planning with state constraints increases expressivity, in the following sense: If the constraint language is sufficiently expressive to compactly encode arbitrary action preconditions and goals over the primary variables – and linear switched constraints are, as shown by Proposition 10 – then compiling away those constraints must

require at least the same worst-case complexity as compiling away those pre- and goal conditions.

5. Relaxations of Planning with State Constraints

To derive admissible heuristics to guide search for optimal plans, we draw on the well-known idea of optimal relaxed planning. We explore two kinds of relaxations: one is based on the monotone (also known as "value accumulating") relaxation that generalises the delete relaxation to non-propositional state variables (Gregory et al., 2012; Domshlak & Nazarenko, 2013), and the other on a form of abstraction, namely projection. In both we have *relaxed states*, which can be viewed as representing sets of assignments to the primary state variables. The key question is how to evaluate conditions on secondary variables in a relaxed state. Our approach is to treat this as a question of consistency, which can be delegated to an appropriate external solver.

From the monotone relaxation, we obtain analogues of the classical $h^{\max}$ and h^+ heuristics for our extended planning formalism. To compute h^+, we use the landmark-based algorithm of Haslum, Slaney and Thiébaux (2012). By changing the algorithm slightly, we can also get an analogue of the LM-Cut heuristic (Helmert & Domshlak, 2009), although it is not likely to achieve the same advantageous trade-off between computational cost and accuracy that LM-Cut does compared to h^+ in the classical case (cf. Section 5.3.2). From abstraction we obtain pattern database (PDB) heuristics (Edelkamp, 2001). PDBs are a building block that can be used in additive ensembles (Haslum, Bonet, & Geffner, 2005) or on-line optimisation (Pommerening, Röger, & Helmert, 2013) to obtain better admissible heuristics. Where it is necessary to distinguish these from the classical versions, we refer to these heuristics as *constraint-aware*, because they are based on relaxations that take the state constraints into account. However, when we say $h^{\max}$, h^+, etc, in the context of a domain formulation with state constraints, it is implied that we mean the constraint-aware version.

5.1 The Monotone Relaxation of Classical Planning

Several researchers (e.g., Gregory et al., 2012; Domshlak & Nazarenko, 2013) have noted that the delete relaxation of propositional planning can also be characterised as planning with a *value accumulating* interpretation of action effects, instead of the usual value assignment semantics: each state variable, v, in a relaxed state has a *set* of values instead of just one value, and applying an action effect $v := e$ adds the new value, e, to the set, without removing any existing value.

Definition 11. *Let V_P be the set of primary state variables, and for each variable $v \in V_P$, $D(v)$ the domain of v, i.e., its set of possible values. A* relaxed state*, s^+, is a mapping from V_P to sets of values such that $s^+(v) \subseteq D(v)$ for all $v \in V_P$.*

The relaxed application of an action a to s^+ results in a state $t^+ = \text{apply}^+(a, s^+)$ such that $t^+(v) = s^+(v) \cup \{e\}$ if $v := e \in \text{eff}(a)$ and $t^+(v) = s^+(v)$ otherwise.

A relaxed state s^+ represents a set of states, namely those obtainable by assigning each variable v_i one value from its value set $s^+(v_i)$:

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

The value-set semantics extends to logical formulas. Intuitively, given a formula φ, $s^+(\varphi)$ denotes the set of values that φ can take in relaxed state s^+.

Definition 12. $s^+(\varphi) = \{s(\varphi) \mid s \in \text{states}(s^+)\}$

In other words, $\mathit{true} \in s^+(\varphi)$ if and only there exists a state $s \in \text{states}(s^+)$ such that $s(\varphi) = \mathit{true}$ (and analogously for false).

Determining the truth value of a general formula in a relaxed state according to Definition 12 is equivalent to deciding satisfiability of the formula conjoined with the restriction on variable values imposed by the relaxed state, and hence potentially intractable. In one special case, however, efficient evaluation is possible: Recall that a simple formula is a conjunction of variable–value equalities or inequalities, without repeated variables. Because each conjunct in a simple formula mentions only one state variable, and is the only part of the formula to mention that variable, their satisfiability in the relaxed state can be evaluated independently (Francès & Geffner, 2015). However, as also observed by Francès and Geffner (2015), this is not true in general: evaluating conjuncts of a non-simple formula independently results in an overapproximation of the set of possible truth values.

The relaxed planning problem is defined by replacing $s(\varphi) = \mathit{true}$ with $\mathit{true} \in s^+(\varphi)$, i.e., actions' preconditions and the goal only need to be *possibly* true, and replacing the normal (value-assigning) definition of action effects with the relaxed (value-accumulating) one. Any plan for the original problem is also a plan under the relaxed semantics; hence the minimum relaxed plan cost is a lower bound on minimum real plan cost. Furthermore, it is never required to apply any action more than once in a relaxed plan, which bounds the length of optimal relaxed plans by the number of actions, and, when pre- and goal conditions are simple, makes relaxed plan existence decidable in polynomial time. Computing the optimal relaxed plan cost, however, is NP-hard (Bylander, 1994).

5.2 Monotone Relaxation of Planning with State Constraints

Adapting the monotone relaxation to our extended formalism requires only resolving how to determine whether a secondary condition (action precondition or goal) holds in a relaxed state, and whether a relaxed state is valid. Our solution to this question is a natural extension of our handling of state constraints in non-relaxed planning, and of the relaxed evaluation of primary conditions in relaxed states: a set of state constraints C holds in a relaxed state s^+ if it is consistent with the restrictions on the primary variable assignment imposed by s^+. However, since state constraints can express non-simple conditions, solving this consistency question exactly is NP-hard in general. In Section 5.5, we define two successively weaker relaxations, which are solvable in polynomial time (assuming that consistency of secondary constraints is tractable). Because of this, we will also refer to the relaxation defined here as "strong".

Recall that $C_P(s)$ denotes a set of constraints (variable–value equalities) that specify the state s exactly, i.e., such that $C_P(s)$ is satisfied in s and not in any other state. In the same way, a set of constraints can specify a relaxed state. The only difference is that where a variable has several possible values, these constraints are disjunctions. That is, the characteristic constraint set of a relaxed state s^+ is $\{\bigvee_{x \in s^+(v)} v = x \mid v \in V_P\}$, (or, equivalently, $\{v \in s^+(v) \mid v \in V_P\}$). We denote this as well with $C_P(s^+)$. As we mentioned

in Section 2, this leads to a natural generalisation of when a partitioned condition holds in a state (Definition 3) to when it holds in a relaxed state:

Definition 13. *Let s^+ be a relaxed state. A partitioned condition (φ_P, φ_S) holds in s^+ according to the strong relaxation iff $\{\varphi_P\} \cup C_P(s^+) \cup \varphi_S \cup C_{\mathsf{inv}}$ is satisfiable.*

Note that while item (*ii*) of Definition 3 required only satisfiability of $C_P(s) \cup \varphi_S \cup C_{\mathsf{inv}}$, the above definition requires that this is jointly satisfiable with the primary part of the condition, φ_P. In a relaxed state s^+ there can be a choice of values for state variables; thus, requiring that this choice satisfies φ_P can constrain the values, which can affect the satisfiability of $C_P(s^+) \cup \varphi_S \cup C_{\mathsf{inv}}$. In a non-relaxed state, on the other hand, which assigns a single value to each variable, the simple formula φ_P is satisfiable if and only if it evaluates to true, so requiring that it is true cannot constrain the state. This is also why $\mathit{true} \in s^+(\varphi_P)$, the equivalent of item (*i*) in Definition 3 is omitted from the definition above. It is implied by the requirement that $\{\varphi_P\} \cup C_P(s^+)$ is satisfiable.

We define the *(strong) monotone relaxation of a planning problem with state constraints* by replacing states and action application with their relaxed counterparts (Definition 11) and condition evaluation with relaxed evaluation (Definition 13).

Analogously to the classical planning relaxation, a set of constraints that is satisfiable in a relaxed state s^+ remains satisfiable in any relaxed state reachable from s^+ by relaxed application of action effects (i.e., "true conditions remain true"). Hence, this relaxation keeps the properties that a plan for the original problem is also a plan for the relaxation (thus optimal relaxed plan cost is a lower bound on optimal real plan cost) and that no action needs to be applied more than once in a relaxed plan.

5.3 Deriving Heuristics from the Monotone Relaxation

The monotonicity property of the relaxation means that we can build a relaxed planning graph, following the same procedure as in classical planning (Hoffmann, 2000). Fact layers are relaxed states. Each action layer includes all actions that are allowed in the preceding relaxed state and that have not appeared in any previous action layer. An action a is allowed in a relaxed state s^+ iff (i) $(\mathrm{pre}(a)_P, \mathrm{pre}(a)_S)$ holds in s^+ and (ii) $(\mathrm{eff}(a), \emptyset)$ holds in $\mathrm{apply}^+(a, s^+)$. The second part ensures that the action's effects, considered by themselves, do not lead to an invalid state. Note that just as in the classical relaxed planning graph, we make an independence assumption in that the allowedness of each action is tested separately from other actions in the same layer. The next fact layer is the relaxed state that results from applying the effects of all actions in the current layer. (There is no need for explicit no-ops, since previously achieved values remain under the value accumulating semantics.) Graph construction stops when the goal holds in the last relaxed state, or when two consecutive relaxed states are the same, indicating that all reachable variable values have been achieved.

The relaxed planning graph construction provides a basis for computing several admissible heuristics. First, it provides a yes/no decision procedure for relaxed plan existence: If it ends with a final relaxed state in which the goal does not hold, the goal is not relaxed reachable. Equipped with this test, we can compute the optimal relaxed plan heuristic, i.e., constraint-aware h^+, using the iterative landmark algorithm (described below). Second, the number of action layers added before the goal condition holds is a lower bound on optimal

plan length. A lower bound on plan cost can be obtained by indexing fact layers of the relaxed planning graph by accumulated cost, rather than depth. We define this lower bound to be the constraint-aware $h^{\max}$ heuristic. (In the classical setting, the $h^{\max}$ heuristic is defined as the greatest fixpoint of the relaxation's Bellman equation, which can be shown to be equivalent to the heuristic computed by the (cost-sensitive) relaxed plan graph construction (Haslum, 2009). However, we have not formulated an equational definition of the $h^{\max}$ heuristic for the formalism with state constraints.)

Although the size of the graph is polynomial (in the size of the planning problem), its construction is not necessarily tractable since deciding if a partitioned condition holds in relaxed state, according to Definition 13, may require solving an NP-hard constraint satisfaction problem. (This is true even if all state constraints are linear switched constraints, because Definition 13 asks for an assignment to the discrete primary variables that are under-constrained by the relaxed state.) In Section 5.5 we define two weaker relaxations, by replacing the exact satisfiability test of Definition 13 with weaker, i.e., overapproximating, conditions, which are decidable in polynomial time (assuming that consistency of secondary constraints is tractable). This makes the relaxed planning graph construction tractable.

5.3.1 Computing h^+

A disjunctive action landmark ("landmark" for short; Karpas & Domshlak, 2009) is a set of actions at least one of which must appear in every relaxed plan; hence, a relaxed plan is a hitting set over any collection of landmarks. The iterative landmark algorithm (Haslum et al., 2012) exploits this by formulating relaxed plan computation as an incremental hitting set problem. Given a set of landmarks, L, it finds a minimum-cost hitting set H of actions. If H is a relaxed plan, it is optimal (because every relaxed plan must hit all landmarks in L, and none can do this at less cost than H). If not, the algorithm generates a new minimal landmark l' disjoint from H, adds l' to the collection L and repeats the process. Generating the new landmark l' is done by extending H to an inclusion-maximal set of actions H' that is not sufficient to make the goal relaxed-reachable, then taking $l' = (A - H')$. Since the actions in H' alone are not sufficient to make the goal relaxed-reachable, any relaxed plan must include at least one action not in H'; thus $(A - H')$ is a landmark. Computing H' is done by iteratively adding actions to the set and testing if it has become a relaxed plan; if so, the last action added is removed again (it will be in the landmark).

Because this algorithm interfaces with the planning formalism only through a relaxed reachability test (is the goal relaxed-reachable from the initial state using a given subset of actions?), which we can perform as explained above, and because our relaxation, like the classical delete-relaxation, does not require any action more than once in an optimal relaxed plan, we can apply this algorithm to compute h^+ also for problems with state constraints. Haslum, Slaney and Thiébaux (2012) propose several modifications to the basic algorithm, which serve mainly to reduce the number of optimal hitting set problems that need to be solved, at the expense of increasing the number of relaxed reachability tests. Although these modifications are applicable also in our setting, they do not lead to faster heuristic computation because the runtime for relaxed reachability testing relative to that of the

hitting set solver is much higher. Also unlike Haslum, Slaney and Thiébaux we use an integer programming solver (Gurobi Optimization Inc., 2016) to find optimal hitting sets.

When computing h^+ for each state, we can use information from the parent state to speed up the iterative landmark algorithm. Let s be a state, $L(s)$ the set of landmarks computed for s, and $s' = \text{apply}(a, s)$ the state resulting from applying action a in s. Then each element of $\{l \in L(s) \mid a \notin l\}$ is also a landmark for s'. Thus, we can start the algorithm with this collection of landmarks, instead of an empty set. This reduces the number of iterations, and hence the number of relaxed reachability tests substantially. A similar technique was used by Pommerening and Helmert (2012) for the LM-Cut heuristic.

5.3.2 Computing LM-Cut

There is a close relationship between the iterative landmark algorithm and the LM-Cut heuristic (Helmert & Domshlak, 2009), as observed already by Bonet and Helmert (2010). The LM-Cut heuristic also iteratively computes disjunctive action landmarks, and the heuristic value is a lower bound on the cost of hitting every landmark in the computed set. However, the landmarks computed by LM-Cut are disjoint. This makes the optimal hitting set problem trivial, since it reduces to finding the least cost element in each set. (The heuristic deals with non-unit action costs by subtracting the least action cost in the landmark from the cost of the others, in a sense "partially hitting" the more costly actions.) As a consequence, LM-Cut is dominated (i.e., upper-bounded) by h^+. On the other hand, in the classical setting, LM-Cut is computable in polynomial time, and in practice strikes a compromise between computation time and heuristic accuracy that is for the vast majority of problems more effective than using h^+.

The iterated landmark algorithm can be restricted to finding a set of disjoint landmarks by initialising the set H' with the union of all landmarks found so far, rather than a hitting set over them. This emulates one aspect of the behaviour of LM-Cut. However, the LM-Cut heuristic, as originally proposed by Helmert and Domshlak (2009), uses a different procedure to compute each landmark. This procedure exploits the explicit causal structure of a propositional planning problem, in the form of a so-called "justification graph", and has a time complexity that is linear in the number of actions, whereas the time complexity of the landmark generation procedure used by Haslum, Slaney and Thiébaux (2012) is quadratic in the number of actions.

It is not obvious how, or even if, an equivalent of the justification graph-based procedure can be constructed for planning problems with state constraints, since it requires pinpointing which action is responsible for adding each proposition (at least cost). Thus, while we can compute an admissible constraint-aware heuristic equivalent to LM-Cut, it is not clear if it will have the same computation time–accuracy trade-off, relative to the optimal monotone relaxation heuristic h^+, as in the classical setting.

5.4 Abstraction of Planning with State Constraints

An abstraction maps the states of a planning problem into a smaller, abstract state space, such that the existence of a path between two states in the original problem implies the existence of a path of lower or equal cost between the corresponding abstract states. Thus, optimal plan cost in the abstract space is a lower bound on optimal real plan cost. Ab-

straction is the basis of planning heuristics such as merge-and-shrink (Helmert, Haslum, Hoffmann, & Nissim, 2014). A *projection* is an abstraction in which all but a designated subset of primary state variables are ignored. Thus, the abstract state space is defined as the set of value assignments over the variables in this set. The set of kept variables is called the *pattern*, and the heuristic obtained from this kind of abstraction is known as a *pattern database*, or PDB (Culberson & Schaeffer, 1998; Edelkamp, 2001).

We define the projection of a planning problem with state constraints onto a subset of primary state variables, B. Action preconditions and effects on primary state variables, and the primary goal condition, in the abstract problem are obtained by ignoring any variable not in B, just as in the classical case. For the state constraints, note that we can view an abstract state s^B as a relaxed state in which primary variables in B have a single value and primary variables not in B have all possible values in their domain. That is,

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

Thus, the truth value of a partitioned condition in an abstract state is determined in exactly the same way as in any other relaxed state, according to Definition 13. That is, a partitioned condition (φ_P, φ_S) holds in s^B iff $\{\varphi_P\} \cup C_P(s^B) \cup \varphi_S \cup C_{\mathsf{inv}}$ is satisfiable. As mentioned in the context of the monotone relaxation above, this means that evaluating a partitioned condition in an abstract state may require solving an NP-hard constraint satisfaction problem. Either of the weaker but tractable satisfaction conditions described in Section 5.5 below can be used in place of the condition of Definition 13 to make the abstract evaluation tractable.

Note that all secondary variables are, in a sense, present in the abstraction. However, heuristic values from abstractions onto effect-disjoint sets of primary variables can still be admissibly added (Edelkamp, 2001). This is a consequence of the fact that the values of the secondary variables are free, and are independently chosen in each abstraction to result in a minimal cost abstract plan for that particular abstraction.

5.4.1 Computing PDB Heuristics

A PDB heuristic precomputes the optimal cost to reach the goal from every abstract state, 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 & Schaeffer, 1998; Edelkamp, 2001). This means that there is no computational overhead for handling state constraints in the heuristic evaluation of search states, since the PDB abstracts primary states. (There is, of course, an overhead in the precomputation phase, where the state constraints must be considered.) This is one of the features that make PDB heuristics attractive in our setting.

In the classical setting, a PDB can be efficiently constructed by an exhaustive reverse exploration from the (abstract) goal states, but this strategy is not easily adapted to problems with state constraints. For example, in many of the example domains presented in Section 3, the goal condition is defined on the secondary model only. This means to even identify the abstract goal states we need to enumerate the models of the set of constraints over primary and secondary variables.

Instead, we adopt a two-stage PDB computation (Ivankovic & Haslum, 2015). The first stage builds an explicit graph of the reachable abstract space, by forward exploration from

the initial state; it also identifies which of the reachable abstract states are goal states. The second stage computes the optimal cost-to-goal for each abstract state present in this graph. This is done by a reverse exploration, starting from the goal states, like in the classical case. This two-stage procedure is more expensive than the classical PDB construction, but still practical for projections that induce a small enough abstract state space. As already mentioned, however, the overhead of handling state constraints is limited to the PDB construction phase: once the PDB is built, state evaluation is done by a table lookup and takes no more time than in a classical PDB heuristic. As we will see in the experimental analysis (Section 7.5), this means that reducing the number of state evaluations has less impact on total runtime when using constraint-aware PDB heuristics than when using the h^+ heuristic, which carries the overhead every time it is computed.

5.5 Tractable Relaxed Evaluation with State Constraints

As shown in Section 4, state constraints in any sufficiently expressive language can encode non-simple formulas over the primary state variables. This, as we have seen above, implies that deciding the truth of a partitioned condition in a relaxed or abstract state, where primary state variables are only partially constrained, may require solving an NP-hard constraint satisfaction problem.

There are several ways to deal with this problem: (1) We can simply accept it, and invoke a complete constraint solver. Although it may take exponential time in the worst case, the worst case does not always occur; depending on the specific form of the constraints and the techniques employed in the solver, it can often be fast enough. We have previously demonstrated in the context of planning with logical axioms that this can be effective for some classes of problems (Ivankovic & Haslum, 2015). (2) We can apply a tractable, sound, but incomplete, inference algorithm to the constraints. The relaxation is still sound as long as we treat a constraint set as satisfiable unless it is proven inconsistent by the inference algorithm. This is the approach taken by Francès and Geffner (2015) to dealing with constraints over the primary variables in the monotone relaxation. (3) In a similar vein, we can define a weaker, but tractable, condition for the relaxed satisfaction of constraints, and thus a weakened relaxation. This is the approach we will take here: in the following two subsections, we define two successively weaker but computationally cheaper conditions for the satisfaction of a partitioned condition in a relaxed or abstract state. In principle, this can also be viewed as using an incomplete inference method.

5.5.1 1ST Weaker Relaxation

Once more, it is the syntactic form of switched constraints that allows us to separate reasoning about the primary and secondary parts of the model. In the relaxed case, however, this separated satisfaction condition is not equivalent to the full consistency test (Definition 13) but induces a further relaxation. Before we can define this relaxation, two technical definitions are needed.

Definition 14. *Let s^+ be a relaxed state and C a set of switched constraints. The relaxed active set of C in s^+ is*

$$\text{active}^+(C, s^+) = \{\gamma \mid \varphi \to \gamma \in C,\ \mathit{false} \notin s^+(\varphi)\}.$$

In other words, a switched constraint $\varphi \to \gamma$ is active in a relaxed state only if *false* is not a possible value for the triggering condition φ in s^+; i.e., φ *must* be true.

Definition 15. *Let s^+ be a relaxed state and φ a simple formula (i.e., a conjunction of variable–value equalities or inequalities without repeated variables) over the primary variables V_P such that true $\in s^+(\varphi)$. $s^+|\varphi$, called s^+* conditioned on *φ, is the relaxed state defined by*

$$
\begin{aligned}
&(s^+|\varphi)(v) = \{e\} && \text{if } v = e \text{ in } \varphi \\
&(s^+|\varphi)(v) = s^+(v) \setminus \{e\} && \text{if } v \neq e \text{ in } \varphi \\
&(s^+|\varphi)(v) = s^+(v) && \text{otherwise}
\end{aligned}
$$

We define the 1st weaker relaxation by replacing the satisfaction condition for partitioned conditions (Definition 13) with the following weaker condition:

Definition 16. *Let s^+ be a relaxed/abstract state. A partitioned condition (φ_P, φ_S) holds in s^+ according to the 1st weaker relaxation iff (i) true $\in s^+(\varphi_P)$ and (ii)* $\text{active}^+(\varphi_S \cup C_{\mathsf{inv}}, s^+|\varphi_P)$ *is satisfiable.*

If the consistency of secondary constraints can be decided in polynomial time, so can this condition. This is the case when, for example, secondary constraints are linear inequalities. Note that we can apply the weaker satisfaction condition of Definition 16 in both the monotone relaxation and abstractions. In both cases, it induces a further relaxation; that is, any plan for the strong monotone relaxation (abstraction) of a planning problem with state constraints is also a plan for the 1st weaker monotone relaxation (abstraction). Thus, optimal plan cost in the weaker relaxations is also a lower bound on real optimal plan cost.

The reason why it is weaker is that the truth of the triggering conditions of switched constraints are evaluated in independently. Consider, for example, constraints (HBW.b.i–HBW.b.ii) from the Hydraulic Blocks World domain described in Section 3.1:

$$\mathsf{in}_i \neq k \to p_{i,k} = 0 \qquad \text{(HBW.b.i)}$$

$$\mathsf{in}_i = k \to p_{i,k} = \bar{w}_i \qquad \text{(HBW.b.ii)}$$

In the state depicted in Figure 1(a), $\mathsf{in}_\mathsf{B} = 2$. Applying the relaxation of action $\mathsf{pickup}(\mathsf{B}, 2)$ results in a relaxed state with $s^+(\mathsf{in}_\mathsf{B}) = \{2, \mathsf{none}\}$. The set of constraints $\{\mathsf{in}_\mathsf{B} \neq 2 \to p_{\mathsf{B},2} = 0, \mathsf{in}_\mathsf{B} = 2 \to p_{\mathsf{B},2} = 2, \mathsf{in}_\mathsf{B} \in \{2, \mathsf{none}\}, p_{\mathsf{B},2} = 1\}$ is not satisfiable, since s^+ encodes a discrete disjunction between $\mathsf{in}_\mathsf{B} = 2 \wedge p_{\mathsf{B},2} = 2$ and $\mathsf{in}_\mathsf{B} = \mathsf{none} \wedge p_{\mathsf{B},2} = 0$. However, because *false* $\in s^+(\mathsf{in}_\mathsf{B} = 2)$ and *false* $\in s^+(\mathsf{in}_\mathsf{B} \neq 2)$, neither of the two switched constraints is active in s^+, leaving only constraint (HBW.b.iv): $0 \leq p_{\mathsf{B},2} \leq 2$, which is consistent with $p_{\mathsf{B},2} = 1$. (This also demonstrates why constraint (HBW.b.iv), which is redundant in the non-relaxed problem, is useful in the relaxation: without it, $p_{\mathsf{B},2}$ would be free to take any value in s^+.)

Conditioning the relaxed state on the primary part of an action's precondition, or its effects when testing if the resulting relaxed state is valid, is a way to partially rectify this, by asserting the variable–value equalities and inequalities that are known to be necessarily true while evaluating the triggering conditions of switched constraints. For example, to determine if the action $\mathsf{stack}(\mathsf{B},\mathsf{A},1)$ is allowed in the relaxed state s^+ in the example above, we test the satisfiability of the invariant constraints that are active in

the resulting state conditioned on this action's effects. The action has, i.a., the effect $\mathsf{in_B} := 1$, so in the resulting state $t^+ = \text{apply}^+(\mathsf{stack(B,A,1)}, s^+)$, $t^+(\mathsf{in_B}) = \{1, 2, \mathsf{none}\}$. However, $(t^+|\text{eff}(\mathsf{stack(B,A,1)}))(\mathsf{in_B}) = 1$, activating constraints $\mathsf{in_B} = 1 \rightarrow p_{\mathsf{B},1} = 2$, $\mathsf{in_B} \neq 2 \rightarrow p_{\mathsf{B},2} = 0$, etc., such that the resulting set of active secondary constraints is unsatisfiable, proving that the plan $\mathsf{pickup(B,2)}$, $\mathsf{stack(B,A,1)}$ is not valid even in the relaxation of the problem.

5.5.2 2ND Weaker Relaxation

Although tractable, the 1st weaker relaxation can be still be too computationally expensive to be the basis of a cost-effective search heuristic. It needs two secondary constraint consistency checks to determine the applicability of an action in a relaxed state (one for the action's preconditions and one to determine if the successor relaxed state is allowed). Each consistency check involves a call to an external solver, which is substantially more time-consuming than evaluating a simple formula over the primary variables in a relaxed state. A relaxation that can be decided more efficiently, but which is also weaker, is obtained by not conditioning the relaxed state in which active constraints are determined. This is the approach we adopted in our earlier work (Ivankovic et al., 2014).

We define the 2nd weaker relaxation by replacing the satisfaction condition for partitioned conditions with the following weaker condition:

Definition 17. *Let s^+ be a relaxed/abstract state. A partitioned condition (φ_P, φ_S) holds in s^+ according to the 2nd weaker relaxation iff true $\in s^+(\varphi_P)$ and* $\text{active}^+(\varphi_S \cup C_{\mathsf{inv}}, s^+)$ *is satisfiable.*

Again, we can apply this weaker condition in the monotone relaxation and abstractions, and again this results in further relaxation of the original problem. The 2nd weaker monotone relaxation has the property that invariant constraints are always satisfiable in any relaxed state that is relaxed reachable starting from a valid state. Thus, in this relaxation, if an action's precondition holds in a relaxed state, the action is also allowed in the state. (Note, however, that this is not necessarily true of 2nd weaker abstractions.) This eliminates one of the two consistency tests needed for each action in each layer of the relaxed planning graph. If the action has no secondary preconditions, no consistency test is needed; it suffices to check if the primary precondition holds in the relaxed state.[4]

As demonstrated by the Hydraulic Blocks World example above, removing the conditioning of the relaxed state can lead to a weaker relaxation. However, this can only happen if the invariant constraints of the problem are unsatisfiable in some reachable states. In two of the example domains presented in Section 3, namely the Linehaul and Counters domains, no reachable state is invalid. In these domains, the secondary model determines only whether the goal has been achieved (and there is no primary goal condition). In domains of this kind, the 2nd weaker relaxation is as good as the 1st.

Consider, for example, the problem with three (integer) counters, X_1, X_2 and X_3, initially all at zero, and the goal $\{X_1 < X_2, X_2 < X_3\}$, described in Section 3.4. In our

4. If the action's precondition does have non-empty primary and secondary parts, a consistency test of the secondary precondition is neeeded anyway, and the relaxation can be strengthened, without significant computational overhead, by conditioning the relaxed state on the primary part of the condition.

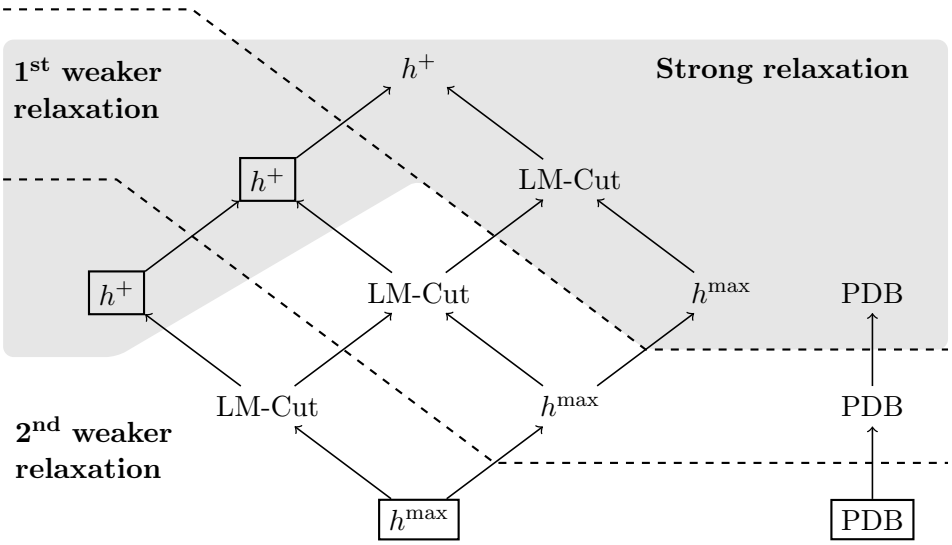


Figure 3: Dominance relations among constraint-aware heuristics. Heuristics in the shaded area are intractable. Heuristics marked with a box are experimentally compared in Section 7.

formulation of the problem, relaxed application of the actions inc(2) and inc(3) leads to a relaxed state s^+ where $s^+(p_{1,0}) = s^+(p_{2,0}) = s^+(p_{3,0}) = \{true\}$, $s^+(p_{1,1}) = s^+(p_{2,2}) = s^+(p_{3,2}) = \{false\}$, and $s^+(p_{2,1}) = s^+(p_{3,1}) = \{true, false\}$. Hence, we have the active constraints $0 \leq x_1 \leq 0$, $0 \leq x_2 \leq 1$ and $0 \leq x_3 \leq 1$. (Recall that the secondary variables x_i here are real-valued, while the counters X_i are integer-valued.) It is clear that their conjunction with the secondary goal condition $\{x_1 + 1 \leq x_2, x_2 + 1 \leq x_3\}$ is unsatisfiable. Thus, although both the individual goal conjuncts are relaxed achievable by the plan inc(2), inc(3), their conjunction is not, even in the 2nd weaker monotone relaxation. Francès and Geffner (2015) show that the same conclusion can be reached by enforcing arc consistency on the constraints $X_1 < X_2$ and $X_2 < X_3$, over integer-valued variables for each counter.

Figure 3 summarises the constraint-aware heuristics defined in this section, and dominance relations between them. (An arrow $h' \to h$ means $h'(s) \leq h(s)$ for all s; this is not the same as the per-state existential dominance relation used by Helmert & Domshlak, 2009.) PDB heuristics are incomparable with all monotone relaxation heuristics. The shaded area of the figure indicates which heuristics are inherently intractable, while those in the area below are computable in polynomial time, under the assumption that checking consistency of secondary constraints is. Section 7 presents an experimental comparison of a subset of heuristics across the four example domains described in Section 3. The heuristics compared are marked with a box in Figure 3.

6. Preferred Actions in A* Search

In the preceding sections, we have shown how to extend a classical planning formalism with state constraints, and how to derive constraint-aware relaxations of the extended planning problem and heuristics from the relaxations. To solve the relaxed problem, and therefore to compute the heuristics, may involve many consistency tests over the secondary model, each of which requires a call to an external solver. This is typically much more time-consuming than solving the corresponding classical planning relaxation. Hence, it is in our interest to extract and make use of as much information as we can from the solution to the relaxed problem, beyond only the heuristic value.

We have combined two ideas — preferred actions and Partial Expansion A* — to create a novel search algorithm that can achieve significant runtime savings when the heuristic is computationally expensive but quite accurate, and states have many successors (Ivankovic et al., 2014). This situation is characteristic of many planning heuristics and problems, including the Power Supply Restoration problem that we tackle. Before presenting the algorithm, we briefly review the two ideas it builds on.

6.1 Preferred Actions in Planning

Preferred actions (also known as "helpful" actions) can be obtained as a side-effect of any heuristic that computes a plan from the current state in some relaxation of the problem. This is the case with, for example, the h^+ heuristic, which computes a plan for the monotonic relaxation, and with PDB heuristics, which are based on the cost of an abstract plan in each projection. (The PDB must be modified to store the first action in the abstract plan for each abstract state.) Even the so-called critical path heuristics, which include $h^{\max}$, can be seen as providing a relaxed plan, consisting of the actions on one (arbitrary) critical path. The preferred actions in a state s are actions that appear in the relaxed plan for s that are also applicable in s. The intuition behind their use is that if the relaxed plan is similar to a real plan, then taking an action that is part of it is more likely to be a step towards the goal, and therefore giving preference to the successor states generated by such actions can lead to a goal state more quickly. This has been shown highly useful in greedy and hill-climbing search (Richter & Helmert, 2009; Hoffmann, 2000), but as far as we are aware there has been no use of this source of information in optimal search.

6.2 Partial Expansion A*

An optimal search algorithm such as A* must expand any state that could possibly lie on a cheaper path to the goal, i.e., any state whose f-value is less than the optimal plan cost, $f^\star$. The Partial Expansion A* (PEA*) algorithm (Yoshizumi, Miura, & Ishida, 2000) tries to avoid placing unpromising states on the open list, by expanding states only partially and re-inserting them on the open list for later consideration. If the f-value of the expanded node n is $f(n)$, only successors with an f-value less than or equal to $f(n) + \Delta$ are placed on the open list (Δ is an algorithm parameter), and the expanded node is re-inserted with its f-value set to the smallest f-value among the discarded successors. However, this means that vanilla PEA* still evaluates all successors to determine which are promising. Felner et al. (2012) noticed that using a problem- and heuristic-specific procedure, it can sometimes

```
1: procedure PrefPEA*
2:   Set open = {(s_0, 0, h(s_0), pref(s_0))}, closed = ∅.
3:   while open ≠ ∅ do
4:     Select n = min_≺ open,
       where (n ≺ n') ≡   (f(n) < f(n'))
                        ∨ (f(n) = f(n') ∧ h(n) < h(n'))
                        ∨ (f(n) = f(n') ∧ h(n) = h(n') ∧ pref(n) ≠ ∅ ∧ pref(n') = ∅))
5:     if n is a goal state then return n.
6:     if pref(n) ≠ ∅ then
7:       Select a ∈ pref(n), remove a from pref(n).
8:       Generate s' from n through a.
9:       NewState(s', g(n) + cost(a))
10:    else
11:      for each non-preferred successor (a', s') of n do
12:        NewState(s', g(n) + cost(a'))
13:      Move n to closed.
14:  return null.

15: procedure NewState(s, g)
16:  if ∄n' ∈ open ∪ closed with state s then
17:    Add (s, g, h(s), pref(s)) to open.
18:  else if g < g(n') then
19:    Set g(n') = g and update parent pointer.
20:    if n' ∈ closed then Move n' back to open.
```

Figure 4: Partial Expansion A* with Preferred Actions (PrefPEA*). The NewState subroutine handles path cost updates and node re-opening, as in standard A*.

be possible to determine the partial successor set without generating and evaluting all successors.

6.3 The PrefPEA* Algorithm

The PrefPEA* search algorithm adopts the idea of partial expansion from PEA*, but stages node expansion by the preferredness of successors instead of their f-value. Pseudo-code for the algorithm is shown in Figure 4. When a state is generated, its set of preferred actions are extracted from the heuristic computation, and stored with the node. When the node is selected for expansion, we generate (and evaluate) only one preferred successor, using one of the actions in its preferred set. This action is then removed from the preferred set, and the parent node kept in the open list. Only when an expanded node has no remaining preferred actions are all its non-preferred successors generated, and the node moved to the closed list. The algorithm prioritises expansion of nodes that have still unexplored preferred successors. This is done using non-emptiness of the preferred action set as an additional tie-breaking criterion, after the standard tie-breaking in favour of lower h-value. That is, if

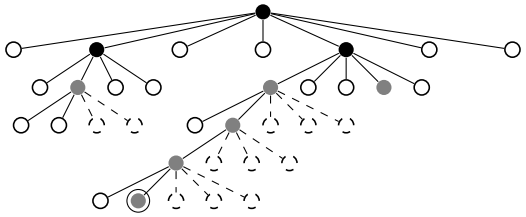


Figure 5: Illustration of PrefPEA* (Ivankovic et al., 2014). Black nodes have $f(n) < f^\star$; these must be fully expanded. Gray nodes have $f(n) = f^\star$; some of these will be expanded, and may be partially expanded. White nodes have $f(n) > f^\star$. The dashed part represents non-preferred successor nodes that are never generated or evaluated. Once the search has reached the $f^\star$ layer and hit a node on an optimal path, tie-breaking on h will keep it on this path (assuming no zero-cost actions). From this point, only preferred successors are generated.

two nodes n and n' have equal f- and h-values, but $\mathsf{pref}(n) \neq \emptyset$ and $\mathsf{pref}(n') = \emptyset$, then n is chosen for expansion before n'.

Proposition 18. *Given an admissible heuristic, the* PrefPEA* *algorithm returns an optimal-cost path.*

Proof. PrefPEA* expands nodes in order of non-decreasing f-values (cf. definition of $\prec$ in line 4 of the algorithm). A node is removed from the open list only when all its successors have been generated (line 13); until then, it remains in the open list with its original f-value, or a lower value if it is updated (cf. line 19). Thus, when the first goal node is selected for expansion and the algorithm returns it, every node reachable by a path of lower f-values has been expanded, and found not to be a goal. Admissibility then implies the goal is not reachable at lower cost, in the same was as it does for A*. □

Because PrefPEA* modifies the order of node expansion in a way that is not guaranteed to be congruent with the heuristic, it may need to reopen closed nodes even if the heuristic is consistent. Like A*, the algorithm must expand every node whose f-value is strictly smaller than the optimal plan cost ($f^\star$). The benefit of PrefPEA* lies in avoiding the generation, and heuristic evaluation, of some siblings of nodes expanded in the final f-layer. This is illustrated in Figure 5. If a large fraction of generated states lie in the final f-layer, if states have, on average, many successors but few preferred ones, and if heuristic computation accounts for a large portion of runtime, these savings can be substantial. If, on the other hand, most node's estimates are below the optimal cost, the branching factor is small, or the heuristic is cheap to compute, then the savings can be expected to be minor. Section 7.5 presents an experimental comparison between A* and PrefPEA*, using the constraint-aware h^+ and PDB heuristics, on problems in the AC-PSR, Hydraulic Blocksworld and Linehaul domains (cf. Section 3). Results of the experiment are mostly in line with predictions based on the characterisation above, though there is one case in which PrefPEA* performs much better that expected.

7. Results

In this section, we present experimental and some analytical results regarding the effectiveness of our approach to planning with state constraints. The questions that we study are: (1) the informativeness, and computational cost, of the proposed heuristics, including the relative strength and cost of the 1st and 2nd weaker monotone relaxation (Sections 7.3–7.4); (2) the benefit of PrefPEA* over plain A* (Section 7.5); and (3) the possible benefits of modelling problems in our formalism with (linear or non-linear switched) state constraints compared to modelling domains in other formalisms (including classical, or classic numerical, planning), where such a comparison is possible (Section 7.6). Sections 7.1 and 7.2 describe the planner implementation and the problem sets, respectively.

7.1 Implementation

We have implemented a framework for planning with state constraints, and in this framework the PrefPEA* and A* search algorithms and several heuristics based on the relaxations described in this paper.[5] Checking the consistency of linear constraints is done by calling an off-the-shelf LP solver; we have used Gurobi, version 6.5.2 (Gurobi Optimization Inc., 2016). For the AC Power Supply Restoration domain, which has non-linear constraints, consistency is tested by a custom solver, built on the PowerTools and SmartGridToolbox libraries.[6] PowerTools in turn uses the Bonmin (Bonami, Biegler, Cohn, Cournéjouls, Grossmann, Laird, Lee, Lodi, Margot, Sawaya, & Wächter, 2008), IPOPT (Wächter & Biegler, 2006) and Gurobi solvers.

The heuristics we have tested are constraint-aware $h^{\max}$ and h^+ based on the two tractable monotone relaxations, and an additive constraint-aware PDB heuristic. Our implementation of the 2nd weaker monotone relaxation uses the strengthening described in Section 5, by conditioning the relaxed state when evaluating a partitioned condition that has non-empty primary and secondary parts. The implementation of h^+ uses caching of parent node landmarks, as described in Section 5.

The PDB heuristic is based on the tractable 2nd weaker abstraction. The additive pattern collection used by the heuristic is found by a local search in the pattern space, following the procedure described by Haslum, Helmert, Bonet, Botea, and Koenig (2007). For domains where the goal condition is on the primary variables, the initial pattern collection has one projection onto each single primary goal variable. (In our test set, only the Hydraulic Blocksworld problems have primary goals.) For domains where the goal is expressed only in terms of secondary constraints, the initial collection is an arbitrary partitioning of the primary state variables that generate small abstractions.

7.2 Problem Sets

We used 1292 problem instances across the four domains presented in Section 3. Details of the instances of the Counters domain are provided in Section 7.6.4.

5. The planner is implemented in Python. Source code and benchmarks are available from `https://github.com/patrikhaslum/gscplanner/`
6. See `http://github.com/hhijazi/PowerTools` and `http://nicta.github.io/SmartGridToolbox/`.

7.2.1 Hydraulic Blocksworld (HBW)

We created 75 problem instances, with 4–7 blocks, using Slaney and Thiébaux's (2001) Blocksworld state generator. Because we did not control the number of towers in the initial or goal states, the number of cylinders for each problem was set to the maximum of the number of towers in the initial and goal states, with a minimum of 3. This resulted in problems with 3–5 cylinders. Block weights and cylinder areas and heights were held constant across all instances (weights: $\overline{w}_1 = 5$, $\overline{w}_2 = 9$, $\overline{w}_3 = 7$, $\overline{w}_4 = 2$, $\overline{w}_5 = 4$, $\overline{w}_6 = 7$, $\overline{w}_7 = 1$; areas: $\overline{a}_1 = 2$, $\overline{a}_2 = 2$, $\overline{a}_3 = 1$, $\overline{a}_4 = 4$, $\overline{a}_5 = 1$). We further vary the constrainedness of the problem instances by varying the total volume of fluid in the cylinders and reservoir, between $\overline{v} = 2$ and $\overline{v} = 10$ in steps of 1. Cylinder heights are large enough that at these volumes no piston ever overflows; only the lower limit $0 \leq h_k$ imposes a constraint on plans. Thus, problems with lower $\overline{v}$ are more constrained than those with higher $\overline{v}$. This can be seen in Figure 6, which shows the distribution of the ratio of optimal plan lengths for solvable instances of the domain compared to the optimal plan length for the corresponding instances ignoring the secondary constraints. At $\overline{v} = 2$ or $\overline{v} = 3$, most problems are unsolvable, and the ones that have solutions are trivial (have plans of zero or a few steps). At $\overline{v} = 10$, the secondary constraint plays essentially no role: all instances are solvable in the same number of steps as when no constraints are imposed. However, at intermediate levels ($\overline{v} = 5$, $\overline{v} = 6$) most instances are solvable but require plans that are significantly longer than when cylinder height constraints are not enforced. Experimental comparison of planner configurations in this domain is done with $\overline{v} = 4, \ldots, 10$, yielding a total of 525 problem instances, 341 solvable and 184 unsolvable. Every instance was decided by some planner configuration.

7.2.2 AC Power Supply Restoration (AC-PSR)

The power network switching problem we consider is a variant of supply restoration planning (PSR). In the initial state, a number of switches are open to protect the network from overloads caused by faults, isolating some loads from any source of power. The goal is to resupply a given set of loads while at all times ensuring that no faulty bus is connected to a live power feed, and that the AC power flow equations and the generator capacity, voltage, and line thermal limit constraints are satisfied. Power networks were drawn from the NESTA benchmark collection (Coffrin, Gordon, & Scott, 2014). We used networks with up to 30 buses. The NESTA benchmarks provide realistic values for all network parameters (loads, generator and line capacities, voltage bounds, etc), which would otherwise be difficult to obtain. The disadvantage of using this benchmark set is that we do not obtain a smooth spread of network sizes. We generated PSR instances as follows. For each bus, we created one instance in which this bus is faulty. The initial state for this instance is obtained, starting with all switches being closed, by opening the switches adjacent to a minimal set of generators sufficiently large to ensure that the faulty bus is not fed. This set of switches to open is determined by depth-first search from the faulty bus. The goal is to supply a subset of buses, selected so as to maximise their sum of active loads $\bar{p}_{L_i}$ whilst satisfying constraints (psr.a.i-psr.f.ii) (Jabr, 2006; Hijazi & Thiébaux, 2015). Note that even though the goal condition is feasible, it is not guaranteed that a goal state is actually reachable because there may not be a transition sequence from the initial state that passes only

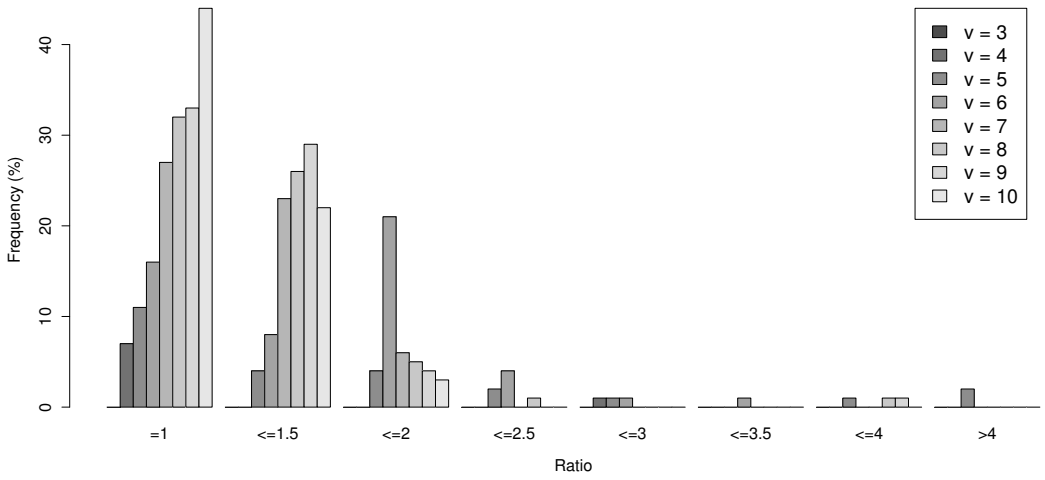


Figure 6: Distribution of the ratio of plan lengths in solvable instances of the hydraulic blocksworld domain to plan length of the corresponding unconstrained problem. Instances are grouped by reservoir volume. Problems with lower volumes are more tightly constrained, so fewer of them are solvable; at intermediate levels ($\bar{v} = 5$, $\bar{v} = 6$) more instances are solvable but require longer plans than when cylinder height constraints are not considered.

through valid states. We encountered a small number of instances (all derived from NESTA network 6-ww) that were unsolvable for precisely this reason.

7.2.3 Linehaul Transportation

Our data set comprises requests for 364 daily deliveries to 6 customer locations. The distance between the depot and each customer is much greater than the distance between customers (which is why it is called a "linehaul" problem). Therefore, it is generally cheaper to use only the largest available vehicles, which have the lowest cost per unit of capacity. We created problem instances by varying the number of refrigerated (R) and ambient-temperature (A) trucks, each between 2 and 4. For a given fleet of trucks, a set of deliveries is feasible if and only if the total quantity of chilled goods requested is at most the total capacity of the refrigerated trucks ($\sum_i \bar{q}_i^{\mathrm{ch}} \leq \sum_{k:\mathrm{ch}\in\bar{\mathcal{G}}_k} \bar{p}_k$) and the total quantity requested is at most the total capacity of all trucks ($\sum_i \left(\bar{q}_i^{\mathrm{ch}} + \bar{q}_i^{\mathrm{am}}\right) \leq \sum_k \bar{p}_k$), since ambient goods can be transported also in refrigerated trucks. We created only solvable instances for each fleet size.

Because our approach to optimally solving planning problems with numeric state constraints is based on admissible heuristic search, it is affected by problem symmetries, such as those caused by identical trucks in the linehaul transportation problem. We can eliminate some of these symmetries (though by no means all) by a small reformulation of the primary model. drive actions are formulated so that trucks can not leave the sink location ($0^\star$) or move back to the source (0) once they have left it. To permit the plan to not use a particular

truck, a zero-cost action $\mathsf{drive}(k, 0, 0^\star)$ is added for each truck. Since movements of different trucks are completely independent, they can be ordered in any way. To avoid the factorial number of equivalent plans that differ only by reordering of independent actions, we force trucks to move in sequence: Truck $k+1$ can only leave the source (depot) after truck k has reached the sink (depot). Furthermore, if trucks $k+1$ and k are of the same type (i.e., $\bar{\mathcal{G}}_k = \bar{\mathcal{G}}_{k+1}$, $\bar{p}_k = \bar{p}_{k+1}$ and $\bar{c}_k = \bar{c}_{k+1}$), then truck $k+1$ cannot be used (i.e. visit at least one customer) unless k has. This avoids the exponential number of equivalent plans that differ only by which subset of identical trucks is used.

7.3 Summary of Results

Coverage (number of problems solved or proven unsolvable) is summarised in Table 1. (The column headed "Alternative formulation" shows the best result achieved by planners tested on alternative domain formulations without state constraints; see Section 7.6 for details.)

Figure 7 shows the distribution of runtimes and search efforts (as measured by nodes evaluated or expanded) for the Hydraulic Blocksworld, Power Supply Restoration and Linehaul Transportation domains. As can be seen, these three domains present very different positions along the trade-off axis between the computational cost and value of heuristic information. In the Hydraulic Blocksworld domain, blind search is nearly the most effective. A* search with the $h^{\max}$ heuristic performs fewer node evaluations, but that gain is matched by an almost equal increase in time per node (for computing the heuristic) such that the runtime of this configuration ends up matching that of the blind search very closely. With the h^+ heuristic, although it is even better informed and evaluates fewer nodes, the net effect is negative as the increased time-per-node results in higher runtimes overall. In the AC Power Supply Restoration domain, on the other hand, informed search – even using the expensive h^+ heuristic, together with preferred action information – solves more problems than blind search, and, for the harder problems, in less time. The Linehaul Transportation domain positions itself in the middle, in that A* search with the computationally cheaper but less informative $h^{\max}$ heuristic solves far more problems than either of using the more informed but expensive h^+ heuristic or no heuristic at all.

The accuracy of PDB heuristics depends on how good the chosen abstractions (patterns) are. In the HBW domain, goals are defined on the primary variables, leading to a good pattern selection and a heuristic that is almost as good as h^+, and roughly an order of magnitude more informed than blind search. In the other two domains, the goal is defined only in terms of secondary constraints, and consequently the pattern selection is essentially arbitrary (as explained in Section 7.1 above). In AC-PSR, the resulting heuristic is still good, while in the Linehaul domain it is almost the same as blind search. (The latter is not surprising, since it is known that projections, which PDBs are based on, are often not capable of capturing useful abstractions of transportation domains; cf. Helmert et al., 2014.)

7.4 Comparing the 1st and 2nd Weaker Monotone Relaxations

The results for $h^{\max}$ and h^+ shown in Table 1 and Figure 7 are for the heuristics based on the 2nd weaker monotone relaxation. We have compared the h^+ heuristic also with its counterpart based on the 1st weaker monotone relaxation. Although the 1st weaker relaxation dominates the 2nd, recall that it is only in domains that have reachable invalid

Problems	#	A*				PrefPEA*		Alternative
		Blind	$h^{\max}$	h^+	PDB	h^+	PDB	formulation
Hydraulic Blocksworld (HBW)								
solvable								
4 blocks	110	110	110	110	110	110	110	110
5 blocks	84	84	84	84	84	84	84	84
6 blocks	80	80	80	52	80	77	80	80
7 blocks	67	64	65	23	67	23	67	66
Σ	341	338	339	269	341	294	341	340[(a)]
unsolvable								
4 blocks	23	23	23	23	23	23	23	23
5 blocks	42	42	42	27	42	37	42	42
6 blocks	53	53	53	50	53	50	53	53
7 blocks	66	66	63	51	66	54	66	66
Σ	184	184	181	151	184	164	184	184
AC Power Supply Restoration (AC-PSR)								
solvable								
4-bus	1	1	1	1	1	1	1	1
6-bus	5	5	5	5	5	5	5	5
9-bus	6	6	6	6	6	6	6	6
14-bus	11	9	9	6	10	8	11	1
24-bus	12[(b)]	0	2	2	2	2	2	2
29-bus	2[(b)]	0	0	0	0	1	1	0
30-bus	47[(b)]	5	5	5	5	5	6	5
Σ	84	26	28	25	29	28	32	20
unsolvable								
6-bus	6	6	6	6	6	6	6	2
Linehaul Transportation								
2R/2A trucks	6	6	6	6	6	6	6	6
3R/2A trucks	19	9	19	7	9	7	12	9
3R/3A trucks	87	7	68	6	7	6	7	9
4R/3A trucks	258	7	63	4	7	6	7	10
4R/4A trucks	303	7	62	4	7	4	7	9
Σ	673	36	218	27	36	27	39	43
Counters								
4–7 counters	4	–	2	2	–	4	–	4

Table 1: Summary of coverage (problems solved or proven unsolvable). The last column shows the best result achieved by planners we tried on domain formulations without state constraints (see Section 7.6 for details). Notes: (a) The unsolved problem was reported unsolvable by the ENHSP planner. This is due to a numeric precision problem; see Section 7.6. (b) It is unknown exactly how many of these are solvable. The solved instances of the largest network sizes have very short plans.

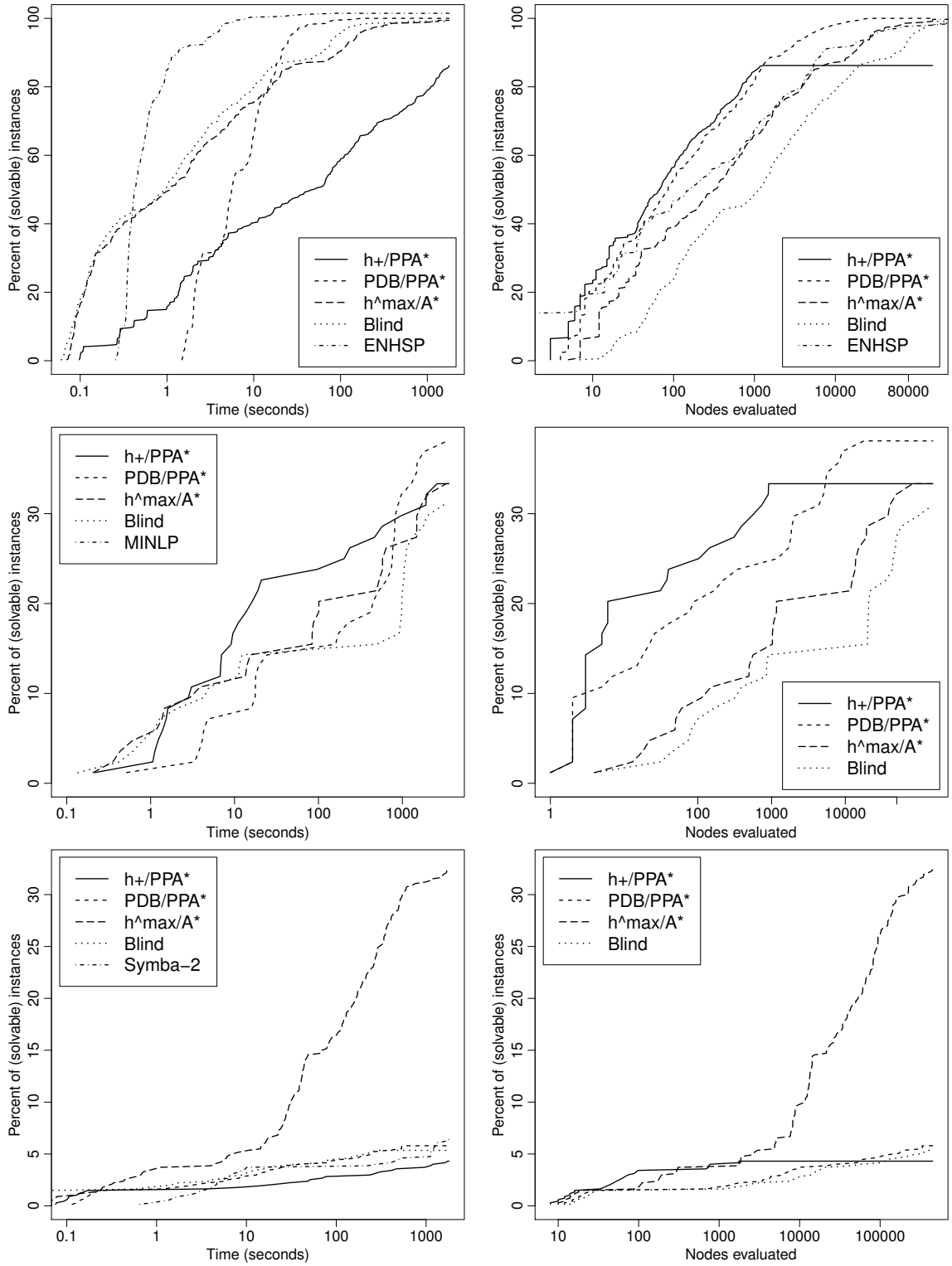


Figure 7: Cumulative distributions of runtime (left) and search effort (right) for different planner configurations in the Hydraulic Blocksworld (top), Power Supply Restoration (middle) and Linehaul Transportation (bottom) domains. ("PPA*" stands for PrefPEA*.)

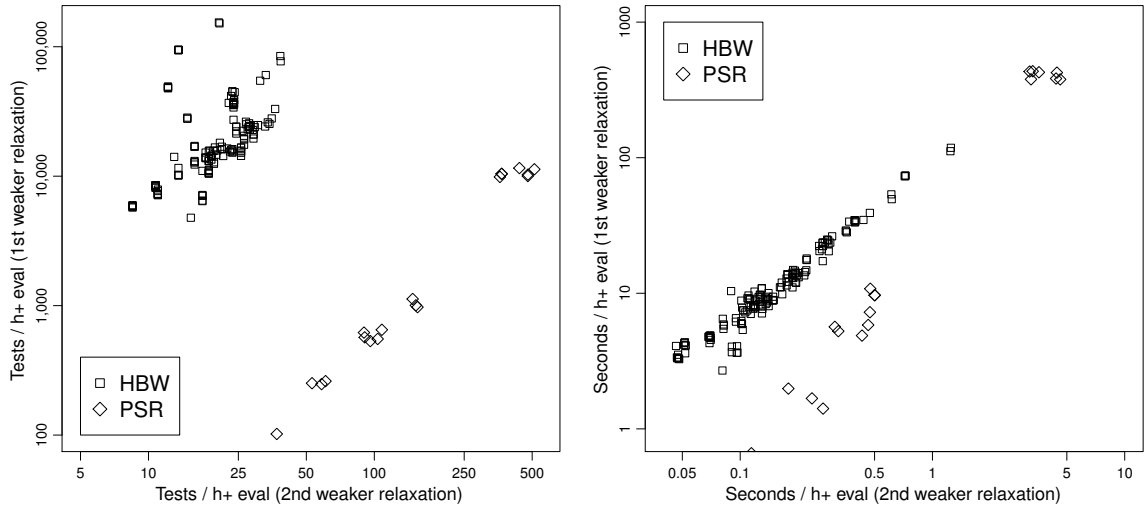


Figure 8: Average number of consistency tests (left) and time in seconds (right) per evaluation of the h^+ heuristic based on the 1st and 2nd relaxation, respectively. (Each point represents one problem solved with both heuristics, and is the average of state evaluations for that problem.) Note that *the scales on the x- and y-axes differ by several orders of magnitude.*

states that it is potentially stronger. Among our examples, this occurs in the HBW and AC-PSR domains. However, the 1st weaker relaxation is also computationally more demanding, and as a consequence it is not effective in either domain. PrefPEA* with h^+ based on the 1st weaker relaxation solves fewer problems (154 vs. 269 in HBW and 19 vs. 28 in AC-PSR, a subset in both domains) and takes substantially more time than using h^+ based on the 2nd weaker relaxation (mean 46.3, median 19.5 times longer in AC-PSR and mean 68.8, median 67.4 times longer in HBW, on commonly solved problems).

The increased runtime is because the number of consistency tests (calls to the external solver) required for each relaxed reachability computation is higher. This difference is substantial: Figure 8 shows the (per-problem) average number of consistency tests (left) and the average time in seconds (right) for each h^+ evaluation. In the HBW domain, the average, over commonly solved problems, number of consistency tests (calls to the LP solver) per h^+ evaluation with the 1st weaker relaxation is more than three orders of magnitude greater than when using the heuristic based on the 2nd weaker relaxation (mean 1317.9, median 782.3), and the average time per h^+ evaluation around 70 times longer (mean 73.1, median 71.6). In the PSR domain, the average number of consistency tests is only one order of magnitude greater (mean 12.7, median 6.6), but since calls to the AC powerflow solver are more time-consuming, the average time per h^+ evaluation is still around 50 times longer (mean 48.9, median 19.3).

The information gain from this extra effort is minor: Comparing the number of nodes expanded to before the optimal f-value is reached (which is guaranteed to be less or equal with a dominating heuristic, with PrefPEA* as well as A*), it is the same for 132 of

the 154 commonly solved problems in the HBW domain and 18 of the 19 commonly solved problems in the AC-PSR domain; where there is a difference, the reduction is between 1 and 7 nodes. Comparing node evaluations (which are not guaranteed to decrease with the more informed heuristic, due to tie-breaking effects in the final f-layer), there is a reduction on 42 of the 154 solved problems in the HBW domain, with on average 17% fewer states evaluated on these 42 problems, and on 5 problems in the AC-PSR domain, averaging 47%. In the HBW domain, there is an increase, of 5.3% on average, in the number of node evaluations on 5 problems.

7.5 The Effect of Partial Expansion by Preferred Actions

Recall that the potential benefit of PrefPEA*, compared to the standard A* search algorithm, lies in avoiding the heuristic evaluation of some siblings of nodes expanded in the final ($f^\star$) layer. If the heuristic is computationally expensive but accurate and states have, on average, many successors but few preferred ones, these savings can be substantial. If, on the other hand, heuristic estimates are far off and the branching factor is small, so are the savings.

The three example domains illustrate both of these cases. The distributions in Figure 9 show the reduction in state evaluations from A* to PrefPEA*, over problems solved by both, using the h^+ (left) and PDB (right) heuristic. (Note that in all three domains and with both heuristrics, PrefPEA* solves all problems that A* can solve; it solves strictly more problems in all cases but for HBW with the PDB heuristic.) Although the distributions are not smooth, it is clear that with h^+, the means as well as modes are ordered Linehaul—HBW—AC-PSR, from lowest to highest. (The average reduction is 70.6% in AC-PSR, 49.8% in HBW and 31.7% in Linehaul; the medians are 64.7%, 47% and 28.4%, respectively.) With the PDB heuristic it is less clear-cut, but the means are still ordered in the same way (71.3% in AC-PSR, 44.1% in HBW and 37.9% in Linehaul).

This is mostly consistent with our expectation of when PrefPEA* should offer an advantage over A*. Both heuristics are, on average, more accurate in the AC-PSR and HBW domains compared to Linehaul. The average ratio of the h^+ heuristic value of the initial state to the optimal plan cost is 0.790 in AC-PSR, 0.641 in HBW and 0.506 in the Linehaul domain; for the PDB heuristic the average ratios are 0.611 in AC-PSR, 0.540 in HBW, and nearly zero in the Linehaul domain. The average branching factor is 7.52 in AC-PSR, 2.52 in HBW and 3.05 in Linehaul. The only really unexpected result is the substantial reduction achieved by PrefPEA* with the PDB heuristic in the Linehaul domain, in many cases needing 50% fewer state evaluations. These are problems on which both algorithms search a large number of nodes (between 10,000 and 350,000 for A*).

When using h^+, heuristic computation as a percentage of total runtime is, on average, 99% in HBW, 93% in AC-PSR and 89% in Linehaul. Hence, the reduction in state evaluations with PrefPEA* leads to a roughly matching reduction in runtime. With the PDB heuristic, on the other hand, state evaluations are relatively fast, and the reduction in state evaluations does not lead to any significant runtime saving; in fact, PrefPEA* with the PDB heuristic is usually slower than A* with the same heuristic.

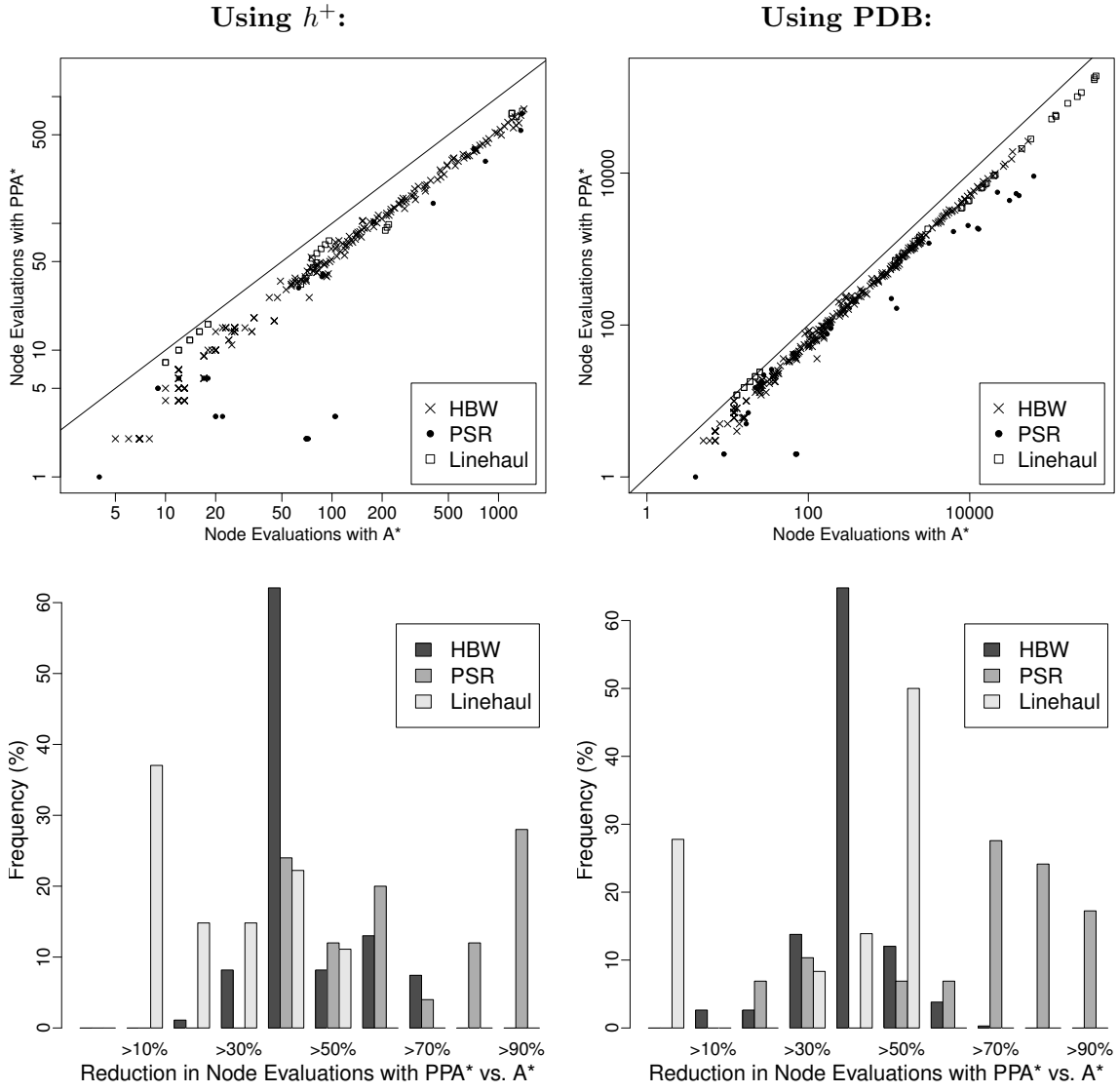


Figure 9: Above: Number of state evaluations made by PrefPEA* compared to plain A*, using h^+ (left) and PDB (right) heuristics. Below: Distribution of the reduction in the number of state evaluations made by PrefPEA* compared to plain A*, using h^+ (left) and PDB (right) heuristics.

7.6 Comparison of Problem Formulations

In this section, we compare the effectiveness of our planner in problem domains formulated with linear or non-linear switched constraints against that of planners that take alternative problem formulations. This includes classical, or classical numeric, planners, where such a formulation is possible, as well as a formulation of the AC-PSR domain as a mixed-integer non-linear program (MINLP). Since our planner is cost-optimal, we consider mainly cost-optimal planners in this comparison. However, we have also tried the suboptimal POPF-TIF planner (Bernardini, Fox, Long, & Piacentini, 2017) on the AC-PSR domain, because this planner is extensible with procedurally defined semantic attachments.

7.6.1 Hydraulic Blocksworld

The Hydraulic Blocksworld domain admits a numeric planning formulation. A numeric fluent p_k tracks the total weight of blocks in each cylinder k. This fluent is decreased or increased, respectively, by actions that take or place a block in the cylinder. The height of the fluid column in each cylinder can then be expressed as a function of the total weight on the pistons in all cylinders:

$$h_k = \frac{\overline{v}}{\overline{a_T}} + \left(\sum_{i \neq k} \frac{p_i}{\overline{a_T}} \right) - \frac{\sum_{i \neq k} \overline{a}_i}{\overline{a}_k \overline{a_T}} p_k$$

where $\overline{a_T} = \sum_{i=1,\ldots,\overline{m}} \overline{a}_i$ is the total area of all cylinders. The invariant constraint that the column height remains within the height of the cylinder, i.e., $0 \leq h_k \leq \overline{l}_k$ for $k = 1, \ldots, \overline{m}$, can then be enforced by substituting the above formula for h_k, and making it a precondition of every action and a goal (as explained in Section 4). Note that this formulation is possible only because in this domain the values of all secondary variables in a state are a function of the primary state (i.e., the arrangement of blocks). In the general case, constraints can allow for more than one assignment of the secondary variables to be consistent with a state.

We solved the numeric planning formulation of the HBW domain with the optimal configuration of the ENHSP planner, which uses A* search with the admissible numeric $h^{\max}$ heuristic (Scala, Haslum, & Thiébaux, 2016a). As shown by Figure 7 (top row), ENHSP is faster on most problems it solves, and it generally performs less search (evaluates fewer states) than the closest corresponding configuration (A* with $h^{\max}$) of our planner on the formulation with state constraints. This is in spite of the numeric $h^{\max}$ heuristic not considering the state constraints in its reachability analysis, since they are already satisfied in every evaluated state. In the HBW domain, the state constraints prune infeasible actions but this creates only very shallow dead ends; that the heuristic fails to anticipate them does little harm to the search.

ENHSP reports as unsolvable one problem instance that has a valid plan. We traced this to a problem with the precision of floating point arithmetic. The failed instance traverses at least one state in which the height of the fluid column in some cylinder is equal to the lower bound, and thus satisfies the constraint, but due to round-off errors in the floating point calculation, the height that ENHSP obtains from the formula above is slightly below the lower bound. The plan validator VAL (Howey, Long, & Fox, 2004), which also uses a floating point representation of numeric fluents, makes the same error.

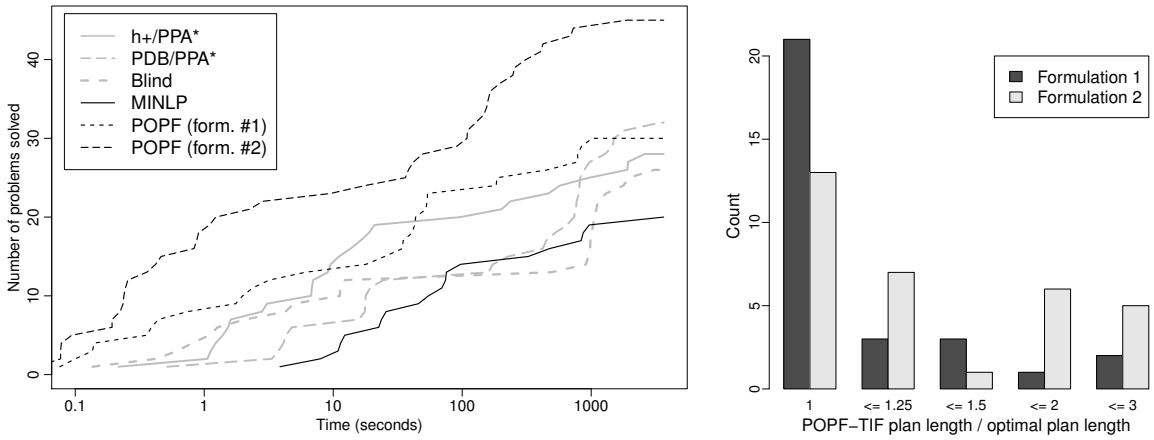
Network	#	Optimal							Suboptimal (POPF-TIF)	
		A*				PPA*				
		Blind	$h^{\max}$	h^+	PDB	h^+	PDB	MINLP	Form. #1	Form. #2
4-bus	1	1	1	1	1	1	1	1	1	1
6-bus	5	5	5	5	5	5	5	5	5	5
9-bus	6	6	6	6	6	6	6	6	6	6
14-bus	11	9	9	6	10	8	11	1	11	11
24-bus	12	0	2	2	2	2	2	2	2	2
29-bus	2	0	0	0	0	1	1	0	0	1
30-bus (a)	24	0	0	0	0	0	1	0	0	12
30-bus (b)	23	5	5	5	5	5	5	5	5	7
Σ	84	26	28	25	29	28	32	20	30	45

Table 2: Problems solved in the AC Power Supply Restoration domain. (The same data as shown in Table 1, but including results for the suboptimal POPF-TIF planner on two domain formulations with different levels of domain-specific advice. Results for the two 30-bus networks have also been separated.)

7.6.2 AC Power Supply Restoration

We are not aware of any optimal planner that could solve the AC-PSR domain off-the-shelf. Instead, we created an encoding of the problem into mixed-integer non-linear programming (MINLP) to take advantage of the capability of the Bonmin solver (Bonami et al., 2008). We also tried the suboptimal POPF-TIF planner (Bernardini et al., 2017), using our AC power flow solver as a semantic attachment.

The MINLP formulation is essentially a classic SATPLAN encoding, with the state replicated for each time step and logical constraints linking consecutive states. The invariant constraints (psr.a.i)–(psr.f.ii) are enforced over the variables representing each state. It allows at most one action per time step. We tried two methods of minimising plan length: The first is the traditional SATPLAN approach, of formulating and solving the encoding for increasing plan lengths, starting from a lower bound on the number of actions, until a solution exists. The second approach is to solve the encoding only once, with a horizon that is (assumed to be) an upper bound on plan length, with an objective function that minimises the number of non-noop actions. Somewhat surprisingly, we found the first approach to be more efficient. Even using a fairly small upper bound (equal to the number of lines, i.e., assuming that every line changes state at most once, which is not necessarily true of an optimal solution), the optimisation-based method is usually slower, and roughly equal in runtime only when the actual plan length is more than half of the bound. Consequently, we evaluate and report results only for the first approach on larger problem instances.



(a) (b)

Figure 10: (a) Distribution of runtimes of different planner configurations and the MINLP solver on problems of the AC-PSR domain. (b) Distribution of the ratio of lengths of plans found by POPF-TIF to optimal plan lengths, over problems where the optimal plan is known.

The POPF-TIF planner is able to use externally defined functions that compute the values of some numeric state variables from the values of other variables. Actions can not have direct effects on these computed variables. In this, they are analogous to PDDL2.2's derived predicates. In our basic formulation of the AC-PSR problem for POPF-TIF, the AC power flow solver computes fed_i for each bus i and a status variable which is 1 (true) if the state is valid and 0 (false) if it is not. Adding the precondition $\mathsf{status} > 0$ to every action and to the goal ensures that any solution plan traverses only valid states, as described in Section 4.

The planner's heuristic does not automatically infer anything about the effect of actions on externally computed state variables. Instead, it relies on the domain designer manually specifying a linear approximation of the effect of any external function. Without this manually specified "advice", the planner's heuristic has no knowledge of the effects of switching lines, and the planner declares every problem instance unsolvable. We tried two formulations, with different levels of advice: In our first formulation, the heuristic effect of closing a line (i, j) is to increase fed_i by the current value of fed_j, and vice versa. This signals to the planner that buses can be supplied by connecting them to an already supplied part of the network. In our second formulation, the AC power flow solver computes an additional variable unsafe_i for each bus, with value 1 if there is a conducting path from bus i to a faulty bus and 0 otherwise. We add the preconditions $\mathsf{unsafe}_i + \mathsf{fed}_j \leq 1$ and $\mathsf{unsafe}_j + \mathsf{fed}_i \leq 1$ to the action that closes line (i, j), and the heuristic effects $\mathsf{unsafe}_i := 0$ and $\mathsf{unsafe}_j := 0$ to the action that opens line (i, j). This forces the planner to not feed faults (which leads to an immediate constraint violation) and hints that opening a line may isolate the adjacent buses from faults.

Table 2 summarises the number of problems solved by all the planners and the MINLP model. (This is the same data that is shown in Table 1, but including also the suboptimal planner.) Figure 10(a) shows the corresponding runtime distribution. As mentioned above, the approach used here to minimise plan length in the MINLP model is that of solving it multiple times for increasing lengths.

The MINLP solver struggles with networks of size 14 and above, as well as with unsolvable problems. This result contrasts sharply with the informal comparison between the earlier version of our planner, which solved only the linearised DC approximation of the PSR domain (Ivankovic et al., 2014), and the corresponding mixed-integer linear programming (MIP) formulation by Thiebaux et al. (2013), which indicated that the MIP-based planner scales much better.[7] We conjecture that this difference reflects differences in the maturity of MIP- and MINLP-solving technology.

Using our basic domain formulation #1, the suboptimal POPF-TIF planner solves slightly fewer problems as the PrefPEA*/PDB configuration of our heuristic search-based planner, though it is faster on the problems that it does solve. (This is not a strict subset: POPF-TIF solves one instance that is not solved by the PrefPEA*/PDB planner.) Providing more domain-specific rules to the planner, as in our domain formulation #2, improves its coverage. It is of course possible that providing the planner with even more advice would enable it to perform better. The improvement in coverage, however, is at the expense of plan quality. The distribution of the ratio of the length of plans found by POPF-TIF, with the two domain formulations, to optimal plan length, across the problem instances for which we have optimal plans, is shown in Figure 10(b). Note that POPF-TIF is unable to offer any guarantee on the quality of the plans it finds.

7.6.3 Linehaul Transportation

The Linehaul domain also admits a numeric formulation, with numeric fluents representing the amount of goods delivered to each customer and the unused capacity in each truck. Since demands and capacities in our problem set are all expressed in integer units, with known tight bounds, we also created a classical formulation of the problem, using propositionally represented counters. This makes the selection of cost-optimal planners that we can attempt to solve the formulation with much larger. We tried the Fast Downward planner (Helmert, 2006), which implements forward state-space A* search, in a variety of configurations that use different heuristics, because this allows for the most direct evaluation of the two problem formulations. We also included the SymBA* planner (Torralba, Alcazar, Borrajo, Kissmann, & Edelkamp, 2014), which was the best-performing planner in the cost-optimal track of the most recent planning competition. We also ran the optimal configuration of the ENHSP planner – A* with the numeric $h^{\max}$ heuristic (Scala et al., 2016a) – on the numeric problem formulation.

The numeric and classical formulations require explicit actions for trucks delivering ("unloading") goods at customer locations. This also means that a forward state-space search planner must commit to the amount delivered by truck k to customer i while the

7. We say "informal" because although the early version of the heuristic search-based planner and the MIP-based planner were evaluated on problem instances drawn from the same network, the MIP-based planner optimises more a general objective function, and they were not tested under equal conditions. If anything, the differences suggest the MIP-based planner is even more effective.

Fleet	#	State constraints formulation						Classical formulation					
		A*				PPA*		Fast Downward					
		Blind	$h^{\max}$	h^+	PDB	h^+	PDB	$h^{\max}$	LM-Cut	iPDB	m&s	SymBA*	ENHSP
2R/2A	6	6	6	6	6	6	6	5	5	5	5	6	3
3R/2A	19	9	18	7	9	7	12	5	4	5	5	9	3
3R/3A	87	7	68	6	7	6	7	5	4	5	5	9	3
4R/3A	258	7	63	4	7	6	7	5	4	5	5	10	3
4R/4A	303	7	62	4	7	4	7	5	4	4	5	9	3

Table 3: Problems solved in the Linehaul Transportation domain. (The same data as shown in Table 1, but with results for all the tried classical planners. The ENHSP planner is applied to the numeric formulation of the domain.)

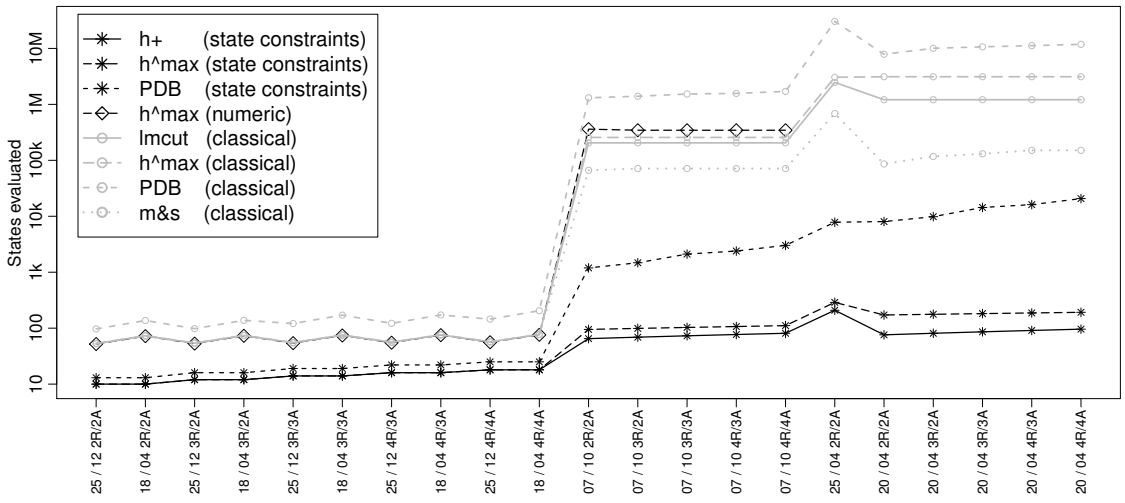


Figure 11: Number of states evaluated by heuristic forward state-space search planners on the 21 commonly solved problem in the Linehaul domain. (Note that the ENHSP planner on the numeric formulation solves only 15 problems in this domain.) The problems are sorted left-to-right by the number of states generated by blind search on the state constraints formulation.

truck is at the customer location. We tried a formulation with one action for each possible amount, i.e., $\mathsf{unload}(k, i, g, l)$ for all $0 \leq l \leq \min(\overline{p}_k, \overline{q}_i^g)$, and one formulation with an action $\mathsf{unload}(k, i, g)$ that delivers a single unit. The former led to an infeasibly large number of ground actions for all but the smallest instances; hence, we use only the latter. In contrast, the formulation with state constraints leaves the choice of how much each truck delivers implicit in the resolution of the secondary goal constraints. Other than this, our classical formulation uses the same symmetry-reducing devices (described in Section 7.2.3) as the state constraints model. We verified with a subset of instances that these improve performance also of the classical planners.

However, the deferred choice makes a great difference in efficiency between the two problem formulations. The difference in search effort required with equivalent heuristics on the two formulations is between one to five orders of magnitude. The classical heuristic search planners always solve fewer problems (and always a subset) than the corresponding configuration of the planner with state constraints. This is summarised in Table 3. Figure 11 shows a problem–by–problem view of search effort (state evaluations) for the 21 instances solved by all planners. We observe that the heuristics ranked by performance fall in the same order on both problem formulations. (Fast Downward does not implement h^+, but we can compare it with the LM-Cut heuristic as the closest substitute.) The merge-and-shrink abstraction heuristic, for which we have not implemented an analogue in the planner with state constraints, is much better informed than the projection abstraction (i.e., PDB heuristic) in this domain. This agrees with the outcome of other comparisons between these two types of abstraction heuristics on similar classical planning problems (Helmert et al., 2014).

7.6.4 Counters

The Counters domain was proposed by Francès and Geffner (2015) as the simplest illustration of one of the weaknesses of the monotone (delete) relaxation of classical planning. Because this relaxation effectively considers each subgoal $X_i + 1 \leq X_{i+1}$ in isolation, incrementing each counter, except the first, once suffices to achieve all of them in the relaxed model. Therefore, the classical $h^{\max}$ heuristic's estimate will be 1 regardless of the number of counters. The h^+ heuristic considers the union of relaxed plans for each subgoal, but not interference between them; thus, its estimate will be $\overline{n}-1$ for a problem with $\overline{n}$ counters. In the relaxation of our formulation with state constraints, the entire goal is checked for consistency with the relaxed state. Because of this, the goal is achieved only by a state in which $X_i \geq i-1$ for each counter i, also in the relaxation. Thus, the value of the constraint-aware $h^{\max}$ heuristic is $\overline{n}-1$ (the number of increments required for the highest-indexed counter), and the value of the h^+ heuristic on this formulation is $\sum_{i=1,\ldots,\overline{n}}(i-1) = \frac{\overline{n}(\overline{n}-1)}{2}$, which is exactly the optimal plan length. Not only is the value of the heuristic perfect, but the optimal relaxed plan is actually the real optimal plan. As a consequence, PrefPEA* with h^+ on the state constraint formulation generates only the nodes on the optimal path, and for each of them generates no more than one successor. Standard A* will expand only nodes on the optimal path, but will generate (and evaluate) all successors of each one of them. In contrast, the information gap of the corresponding heuristics on the classical formulation means that search effort and runtime grow exponentially. Table 4 shows these results

		Number of counters ($\overline{n}$)			
		4	5	6	7
Optimal plan length ($f^\star$)		6	10	15	21
$h(s_0)$					
state constraints	h^+	6	10	15	21
	$h^{\max}$	3	4	5	6
classical	h^+	3	4	5	6
	$h^{\max}$	1	1	1	1
Nodes expanded					
state constraints	PrefPEA*, h^+	0	0	0	0
	A*, h^+	6	10	–	–
	A*, $h^{\max}$	35	465	–	–
classical	A*, h^+	25	219	–	–
	A*, $h^{\max}$	70	835	14341	275465
Nodes evaluated					
state constraints	PrefPEA*, h^+	6	10	15	21
	A*, h^+	36	77	–	–
	A*, $h^{\max}$	205	3681	–	–
classical	A*, h^+	87	617	–	–
	A*, $h^{\max}$	154	1633	23549	388671
Total time (seconds)					
state constraints	PrefPEA*, h^+	21.0	107.7	407.1	1141.1
	A*, h^+	137.7	950.9	–	–
	A*, $h^{\max}$	2.3	72.0	–	–
classical	A*, h^+	0.7	58.5	–	–
	A*, $h^{\max}$	0.1	1.5	47.9	1555.2

Table 4: Detailed results on Counters problems. The maximum counter value ($\overline{m}$) is 8 in all instances. In PrefPEA*, we count a node as expanded only when it has been fully expanded, i.e., all successors generated.

in detail. (The h^+ heuristic used for the classical formulation is a domain-independent implementation of the same algorithm that is used in our state constraint-aware planner.)

Although computing the h^+ heuristic is in general NP-hard, the Counters problems have the property that all generated landmarks are disjoint. This makes the hitting set problem trivial, and means that PrefPEA* search with h^+ on the state constraint formulation solves arbitrary instances of the Counters domain in polynomial time.

8. Related Work

We survey two areas of related work: (1) planning models and planners that incorporate some form of state constraints; and (2) approaches to interfacing heuristic search-based planners with special-purpose solvers or functions.

8.1 Planning with State Constraints

Early work on planning – including STRIPS, planners based on theorem-proving, and many subsequent action formalisms – used state constraints to concisely represent actions, allowing a rich set of derived predicates and functions in preconditions and goals and helping to alleviate the frame and ramification problems (e.g., Green, 1969; Fikes & Nilsson, 1971; Liftschitz, 1987; Ginsberg & Smith, 1988; Winslett, 1988 Thiébaux & Herzberg, 1992; Sandewall, 1994; Levesque, Reiter, Lespérance, Lin, & Scherl, 1997).

However, state constraints have not found their way into efficient implementations of modern planners. Much of recent work in planning has focused on improving the algorithmic aspects of plan generation for simpler formalisms that do not support state constraints. The quantified and conditional effects found in ADL variants do not achieve the same purpose, since state constraints do not update variables based on the predecessor state, but are equations relating the values of variables in the same state. Weld and Etzioni (1994) proposed integrating safety constraints into a partial-order planner. In their model, however, constraints serve only the role of specifying invalid states; they are not used to compute values of derived variables.

PDDL2.2's derived predicates and axioms (Thiébaux et al., 2005; Hoffmann & Edelkamp, 2005) enable the compact encoding of a larger class of constraints over Boolean variables, suitable for capturing transitive closure and thus the reachability aspects of network flows. However, only a handful of domain-independent planners support them, and with few exceptions (e.g., Gerevini, Saetti, Serina, & Toninelli, 2005; Helmert, 2006) do not include substantial improvements to domain-independent heuristics to deal with complications arising from the presence of axioms. Moreover, there was to our knowledge no work on *optimal* planning with derived predicates before the planner we built on the basis of the framework presented in this paper (Ivankovic & Haslum, 2015).

Extending classical planners to deal with problems with numeric state variables and effects has been an area of research for some time (e.g., Koehler, 1998; Wolfman & Weld, 1999; Hoffmann, 2003; Coles, Coles, Fox, & Long, 2013; Scala et al., 2016a; Scala, Haslum, Thiébaux, & Ramírez, 2016b), and more recently the scope of this work has widened to include planning for hybrid discrete–continuous dynamical systems, in which the state evolves following autonomous processes as time progresses (e.g., Shin & Davis, 2005; Li & Williams, 2008; Della Penna, Magazzeni, Mercorio, & Intrigila, 2009; Coles, Coles, Fox, & Long, 2009; Löhr, Eyerich, Keller, & Nebel, 2012). These planners are, however, designed for models that associate a small number of discrete modes with (dynamical) equations, not to modeling network flows in which the number of modes required is equal to the, prohibitively large, number of network configurations.

Nevertheless, a few authors describe applications of planning in physically interconnected domains. Aylett et al. (1998) plan operating procedures for a chemical plant, while Piacentini et al. (2015) address voltage control in a power distribution network, both using planning systems. To reason about network flows, both use an architecture in which the planner interacts with a special-purpose solver. We discuss these approaches in detail in the following subsection. Vallati et al. (2016) model the flow of traffic in an urban environment as a set of interacting processes. However, in this application the planner controls only

the rate of flows at intersections, by switching traffic lights, and not the direction that the traffic flows.

8.2 Special-Purpose Solvers in Heuristic Search Planning

The idea of delegating parts of the reasoning (or computation) required to construct a plan to one or more specialised solvers has been proposed many times in planning. This can be done for the purpose of broadening the scope of problems that the planner can address, or only for the purpose of improving efficiency on special classes of classical planning models (e.g., Fox & Long, 2001; Srivastava & Kambhampati, 1999).

Dornhege et al. (2009) use the term "semantic attachment" for a predicate or function in the planning model that is evaluated by calling a procedure. This type of integration of special-purpose computational procedures or solvers has a long history of use in planning, appearing in systems such as Prodigy (Veloso, Carbonell, Perez, Borrajo, Fink, & Blythe, 1995), O-Plan (Currie & Tate, 1991; Tate, Drabble, & Kirby, 1994), TLPlan (Bacchus & Kabanza, 2000), CEP (Aylett et al., 1998) and RAX-PS (Jonsson, Morris, & Rajan, 1999). A key property of such mechanisms is that they are, at least in principle, extensible to arbitrary procedures. Hence, we do not consider the use of, for example, specialised temporal reasoning in temporal planners (e.g., Halsey, Long, & Fox, 2003; Shin & Davis, 2005) to be in this category. However, one special case worth mention is the integration of geometric path or motion planning with task planning, which has been a focus of work in planning for robotics (Cambon, Gravot, & Alami, 2003; Cambon, Alami, & Gravot, 2009; Plaku & Hager, 2010; Lagriffoul, Dimitrov, Saffiotti, & Karlsson, 2012; Srivastava, Fang, Riano, Chitnis, Russel, & Abbeel, 2014; Toussaint, 2015; Garrett, Lozano-Pérez, & Kaelbling, 2015).

The partitioning of the planning problem into a primary (usually classical) part and one or more attached subproblems which are solved by other methods may be seen as a form of Bender's decomposition (Hooker, 2000). Factored planning methods (e.g., Amir & Engelhardt, 2003; Domshlak & Brafman, 2006; Fabre, Jezequel, Haslum, & Thiébaux, 2010) decompose the problem into parts that are all classical, but may still distinguish one of those as primary (Gnad & Hoffmann, 2018). Most planners with semantic attachments do not implement an analogue of Bender's cuts, i.e., inferring constraints on the primary model from inconsistencies or suboptimal solutions to the subproblem, though there are exceptions, e.g., the ITSAT temporal planner, which infers causal constraints from inconsistencies in temporal constraints (Rankooh & Ghassem-Sani, 2015). We experimented with a no-good learning approach, which lazily adds constraints to the primary model to eliminate partial state variable assignments that have been discovered to lead to inconsistency in the secondary model, but this did not lead to significant improvement in runtime.

Integrating semantic attachments into search is straightforward, as the planner only needs to call the attached procedure when all its inputs are known. The attached procedure may be a non-deterministic function returning several solutions, the choice over which become backtracking points in the search (e.g., Aylett et al., 1998). For example, in a forward-chaining state space search a predicate defined by an attached procedure is evaluated only in fully defined states, while in a constraint-based planner a procedurally defined constraint can be checked once all involved variables are assigned. What is not so simple is

how to make heuristics that guide the search aware of the domain knowledge embedded in the semantic attachments.[8] As an indication, the early planning systems mentioned above all rely on domain-specific search control knowlege (hand-crafted or learned) rather than the kind of domain-independent search heuristics obtained from problem relaxations that have become favoured more recently.

Three recent proposals have integrated semantic attachments into heuristic state-space search planning. Dornhege et al. (2009) state that "*..., we require that effect-applicators [i.e., the attached functions] always terminate and result, for identical parameters and states, in identical settings of the fluents they act on. In particular, effect applicators should not contain any mechanism for making choices between different outcomes, such as selecting a location for placing an object.*" In other words, they restrict semantic attachments to be deterministic predicates or functions of the planning state. However, one of their example domains – robot manipulation planning – contradicts this description, in that the attached motion planning procedure computes a set of alternative grasping poses, which appears to become a branching point in the search. For guiding the search, Dornhege et al. (2009) use an FF-like heuristic, based on the monotone relaxation. Computed predicate and function values are treated as facts, and therefore subject to the same value-accumulating semantics as normal state variables. This means that attached functions must either be able to take as input a relaxed state (in which variables have a range of values), or all combinations of reachable function inputs enumerated to compute new reachable values. It is unclear which of these two options the planner uses. Enumeration may be feasible if each function's inputs is only a few state variables, but for functions of the entire state (such as, for example, the power flow in a power network) this amounts to enumerating all realisations of the relaxed state, i.e., all of states(s^+). The authors equip attached functions with a flag to indicate when the function is called as part of heuristic evaluation, suggesting that it may then use a computationally cheaper approximation.

Gregory et al. (2012) propose "planning modulo theories" (PMT). Their approach extends the planning model with new state variable *types* (e.g., integers, sets, or arrays). Each type is equipped with a defined set of interpreted functions and predicates. Like Dornhege et al.'s semantic attachments, these functions are deterministic, so that their computation does not introduce branching points in the search. To define a heuristic for the extended planning model, they use a domain abstraction for each attached type. The abstraction may be chosen depending on the state. The implementation of the attached type must provide implementations of all its functions in the abstract value domain, as well as a "folding" operator which combines abstract values; the role of this folding operator is analogous to that of the union of sets of values in the monotone relaxation. Provided this, they show how to compute the $h^{\max}$ heuristic. In principle, it is possible to use $h^{\max}$ as a relaxed reachability test and therefore to compute h^+ or LM-Cut in the way we have shown in this paper also

8. Here, we can make an analogy with search-based constraint solvers: Semantic attachments are similar to propagators, in that they are procedures invoked at a point in the search, which perform specialised inference and provide a result (in the form of reduced variable domains) to the search. However, the heuristics that guide the search (that is, variable and value selection heuristics) are typically unaware of what the propagators do, and are based only on what can be directly observed from the variables and their domains.

in the PMT framework. However, the $h^{\max}$ computation is guaranteed to terminate only if, for every state, every relaxed reachable abstract domain is finite.

Piacentini et al. (2015) present an extension of the temporal numeric planner POPF-TIF for the voltage control problem in AC power networks. Unlike the power network problems we considered, they do not include switching actions that modify the network topology; instead, actions in their problem affect network elements that control voltage, such as changing transformer settings and activating or deactivating capacitors; their problem also includes the possibility of shedding loads. Like in our problem, however, actions are discrete and can have ramifications on continuous state variables (such as voltage and power flow) across the entire network. The power flow calculation is integrated into the planner as a semantically attached function. Different from the works above, Piacentini et al. partition (numeric) state variables into "special" variables, which are computed by the attached function, and normal variables, which are affected by actions; the special variables are updated automatically (by calling the power flow solver) when any variable that they depend on changes. This is similar to PDDL2.2's partitioning of predicates into basic (persistent) and derived (defined by axioms). Note that the power flow solver computes only one solution (flow and voltage values) even through several may exist. To make the indirect effect of actions on the special variables visible in the heuristic calculation, these are approximated by a linear effect. For example, if the action changing the tap setting of a transformer up by one step in a given state s increases the voltage at bus b by δ, then the heuristic is evaluated in a model where this action has the constant effect `(increase (voltage` b`)` δ`)`. The POPF planner is a temporal numeric planner, so its heuristic is able to deal with linear numeric effects of this kind. Note, however, that this is an approximation, because as the state changes so does the effect of the action on the voltage. They compare several versions of this heuristic, which differ in how and when the linear approximation is computed. It may be computed only once, in the initial state, by more or less precise methods, or recomputed for every evaluated state. Bernardini et al. (2017) applied the POPF-TIF approach to several other problems using semantic attachments. In each case, the approximate effect used in the heuristic is a form of "advice" that must be manually provided by the domain modeller; the planner does not attempt to extract any information from attached functions automatically.

Although the state constraints in our extended planning model can be seen as a form of semantically attached predicates, a significant difference to the approaches described above is that in our model, the values computed by the constraint solver do not persist between states. As shown in Section 4, this limits the expressivity of the extended formalism. However, that limitation is also what makes the problem decidable for any type of secondary variables and constraints (as long as consistency of the constraints is decidable) and enables us to guarantee plan optimality. Allowing persistent state variables with non-finite domains leads very easily to an undecidable plan existence problem, as demonstrated in the case of numeric planning (Helmert, 2002). Of the works above, Gregory et al. (2012) are the only to propose a method of cost-optimal planning for the extended planning model, and that is only possible with finite types. Another important advantage of our way of extending the planning formalism, which sets it apart from the approaches described above, is that problem relaxations, and therefore heuristics which are computed automatically from

the domain formulation, can be made aware of the secondary model without the external constraint solver needing to implement any specialised functionality for relaxed reasoning.

In planning for mobile robots, the main constraint is collision-free reachability in the physical world. An abstract task, such as pickup(o,table), can correspond to many concrete movements, as long as they bring the robot's manipulator into a position where it can grasp object o, and neither the robot nor the object collide with any obstacle along the way. In this way, the specific robot poses and object positions chosen for an abstract action sequence can be seen as under-constrained variables, akin to the secondary variables in our formalism but with the important difference that the collision-free reachability constraint spans pairs of subsequent states, rather than just a single state. Several integrated task and motion planners take the approach of formulating a motion planning problem that is constrained by a proposed abstract action sequence, interleaved with planning the action sequence (e.g., Cambon et al., 2003, 2009; Plaku & Hager, 2010; Lagriffoul et al., 2012; Srivastava et al., 2014; Toussaint, 2015). To guide the exploration of potential action sequences, these planners plan in the abstract action space using some relaxation of the geometric and motion constraints to evaluate candidate actions, or (incrementally) compile some of the constraints into the abstract planning model. This use of a relaxed motion constraint check, which differs from our approach that uses the same consistency solver for both action validation and heuristic computation, is motivated by the computational cost of invoking a complete motion planner. As demonstrated by our experiments (Section 7.3), invoking the constraint solver in heuristic computation can be costly, and the idea of using a weaker, sound but incomplete, inference algorithm can be applied in our setting as well. This approach was taken by Francès and Geffner (2015) in their constrained planning graph heuristic, and in the application of our framework to planning with PDDL2.2's derived predicates (Ivankovic & Haslum, 2015).

9. Conclusion

We have introduced a principled way of extending a classical planning formalism with systems of state constraints. Checking the consistency of state constraints, both in validating actions and computing heuristics, is outsourced to suitable solver; the integration of constraints into the formalism and of the solver into the planner is independent of the type of the constraints and requires no additional capabilities of the solver. In this paper, we have applied our approach to planning with linear and non-linear constraints over numeric variables, but we have also applied it to planning with derived predicates defined by logical axioms, demonstrating its generality (Ivankovic & Haslum, 2015).

Numerical state constraints provide support for modelling interconnected physical systems, in which a single discrete control action can have global effects (e.g., network flows) that depend on the states of many components, and thereby can enable the application of automated planning to a wider range of problems in areas such as power systems. We also demonstrated that even for problems that can be modelled in a purely classical formalism, deferring some choices from the planner's search space to a constraint solver can have computational advantages. This supports the argument that more expressive formalisms can enable more efficient modelling (e.g., Francès & Geffner, 2015). It is related to the idea of improving planner efficiency by domain reformulation in the classical setting, for

example by modelling identical objects with counters (Riddle, Douglas, Barley, & Franco, 2016; Fuentetaja & De la Rosa, 2016), but more powerful in that the secondary model can compactly express complex state constraints. However, repeatedly calling the constraint solver, for example during heuristic computation, consumes time, and it is not always cost effective. In our experiments with the h^+ heuristic on the Counters domain, the time spent calling the LP solver is between 82%–90% of total runtime (averaging 87%). However, of that time, less than 10% is spent actually solving the LP; the remaining 90% or more is spent in the interface between the planner and the LP solver. This suggests some of the computational overhead can be engineered out of the integration.

The main limitation of our extension framework is that it includes only constraints over each state in the plan trajectory. On the one hand, this limitation means that we are able to solve planning problems in the extended formalism cost-optimally, but on the other, it also means that problems with constraints that require persistence of secondary variables – for example, trajectories in space or slow-moving network flows – cannot be easily expressed. Finding a restricted fragment of cross-state constraints, or a combination of state constraints with a restricted use of numeric primary variables (see, e.g., Scala, Ramírez, Haslum, & Thiébaux, 2016c), that allows for modelling relevant problems while maintaining the desirable properties of our framework is a topic of future work.

Acknowledgements

We thank Chiara Piacentini and Sara Bernardini for their advice on how to use semantic attachments in POPF-TIF.

This work was supported by ARC project DP140104219, "Robust AI Planning for Hybrid Systems", and in part by ARO grant W911NF1210471 and ONR grant N000141210430. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred. Miquel Ramirez's contribution to this work was made while he was at the Australian National University. Vikas Shivashankar's contribution to the work was made while he was at the University of Maryland.

References

Amir, E., & Engelhardt, B. (2003). Factored planning. In *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 929–935.

Aylett, R., Soutter, J. K., Petley, G. J., & Chung, P. W. H. (1998). AI planning in a chemical plant domain. In *European Conference on AI*, pp. 622–626.

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116*, 123–191.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, *11*, 625–656.

Bernardini, S., Fox, M., Long, D., & Piacentini, C. (2017). Boosting search guidance in problems with semantic attachments. In *Proc. 27th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 29–37.

Bonami, P., Biegler, L. T., Cohn, A. R., Cournéjouls, G., Grossmann, I. E., Laird, C. D., Lee, J., Lodi, A., Margot, F., Sawaya, N., & Wächter, A. (2008). An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, *5*(2), 186–204. https://doi.org/10.1016/j.disopt.2006.10.011.

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

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

Cambon, S., Alami, R., & Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, *28*(1), 104–126.

Cambon, S., Gravot, F., & Alami, R. (2003). Overview of aSyMov: Integrating motion, manipulation and task planning. In *Proc. ICAPS Doctoral Consortium*, pp. 21–25.

Coffrin, C., Gordon, D., & Scott, P. (2014). Nesta, the NICTA energy system test case archive. *CoRR*, *abs/1411.0359*.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2009). Temporal planning in domains with linear processes. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1671–1676.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2013). A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of AI Research*, *46*, 343–412.

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(3), 318–334.

Currie, K., & Tate, A. (1991). O-Plan: the open planning architecture. *Artificial Intelligence*, *52*, 49–86.

Della Penna, G., Magazzeni, D., Mercorio, F., & Intrigila, B. (2009). UPMurphi: A tool for universal planning on PDDL+ problems. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 106–113.

Domshlak, C., & Brafman, R. (2006). Factored planning: How, when and when not. In *Proc. 21st National Conference on Artificial Intelligence (AAAI)*.

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

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., & Nebel, B. (2009). Semantic attachments for domain-independent planning systems. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Fabre, E., Jezequel, L., Haslum, P., & Thiébaux, S. (2010). Cost-optimal factored planning: Promises and pitfalls. In *Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 65–72.

Felner, A., Goldenberg, M., Sharon, G., Stern, R., Beja, T., Sturtevant, N., Schaeffer, J., & Holte, R. C. (2012). Partial-expansion A* with selective node generation. In *AAAI*, pp. 471–477.

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

Fox, M., & Long, D. (2001). Hybrid STAN: Identifying and managing combinatorial optimisation subproblems in planning. In *Proc. 17th International Joint Conference on AI (IJCAI)*, pp. 445–452.

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

Fuentetaja, R., & De la Rosa, T. (2016). Compiling irrelevant objects to counters: Special case of creation planning. *AI Communications*, *29*(3), 435–467.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2015). FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pp. 179–195.

Geißer, F., Keller, T., & Mattmüller, R. (2015). Delete relaxations for planning with state-dependent action costs. In *Proc. 24th International Joint Conference on AI (IJCAI)*, pp. 1573–1579.

Geißer, F., Keller, T., & Mattmüller, R. (2016). Abstractions for planning with state-dependent action costs. In *Proc. 26th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 140–148.

Gerevini, A., Saetti, A., Serina, I., & Toninelli, P. (2005). Fast planning in domains with derived predicates: An approach based on rule-action graphs and local search. In *AAAI*, pp. 1157–1162.

Ginsberg, M. L., & Smith, D. E. (1988). Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, *35*(2), 165–195.

Gnad, D., & Hoffmann, J. (2018). Star-topology decoupled state space search. *Artificial Intelligence*, *257*, 24–60.

Green, C. C. (1969). Application of theorem proving to problem solving. In *Proc. 1st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 219–240.

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

Gurobi Optimization Inc. (2016). Gurobi optimizer..

Halsey, K., Long, D., & Fox, M. (2003). Isolating where planning and scheduling interact. In *Proc. of the 22nd UK Planning and Scheduling Special Interest Group (PlanSIG)*, pp. 104–114.

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

Haslum, P. (2009). $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from h^{max} to h^m. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Haslum, P., Bonet, B., & Geffner, H. (2005). New admissible heuristics for domain-independent planning. In *Proc. 20th National Conference on AI (AAAI'05)*, pp. 1163–1168.

Haslum, P., Helmert, M., Bonet, B., Botea, A., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI'07*, pp. 1007 – 1012.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *Proc. 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 303–312.

Helmert, M. (2006). The Fast Downward planning system. *Journal of AI Research*, *26*, 191–246.

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

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, *61*(3), 16:1–16:63. `http://dx.doi.org/10.1145/2559951`.

Hijazi, H., & Thiébaux, S. (2015). Optimal distribution systems reconfiguration for radial and meshed grids. *International Journal of Electrical Power & Energy Systems*, *72*, 136–143. The Special Issue for 18th Power Systems Computation Conference.

Hoffmann, J. (2000). A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proc. 12th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pp. 216–227.

Hoffmann, J. (2003). The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of AI Research*, *20*, 291–341.

Hoffmann, J., & Edelkamp, S. (2005). The deterministic part of IPC-4: An overview. *Journal of AI Research*, *24*, 519–579.

Hoffmann, J., Edelkamp, S., Thiébaux, S., Englert, R., dos S. Liporace, F., & Trüg, S. (2006). Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of AI Research*, *26*, 453–541.

Hooker, J. (2000). *Logic-based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. John Wiley & Sons.

Howey, R., Long, D., & Fox, M. (2004). VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 294–301.

Ivankovic, F., & Haslum, P. (2015). Optimal planning with axioms. In *Proc. of the 24th International Joint Conference on Artificial Intelligence*, pp. 1580–1586.

Ivankovic, F., Haslum, P., Thiébaux, S., Shivashankar, V., & Nau, D. (2014). Optimal planning with global numerical state constraints. In *Proc of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 145–153. AAAI Press.

Jabr, R. A. (2006). Radial distribution load flow using conic programming. *IEEE Transactions on Power Systems*, *21*(3), 1458–1459.

Jonsson, A. K., Morris, P. H., & Rajan, N. M. K. (1999). Next generation remote agent planner. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*.

Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1728–1733.

Kilby, P., Abio, I., Guimarans, D., Harabor, D., Haslum, P., Mayer-Eichberger, V., Siddiqui, F., Thiébaux, S., & Urli, T. (2015). There's more than one way to solve a long-haul transportation problem. In *Vehicle Routing and Logistics (VeRoLog)*. (Abstract.).

Koehler, J. (1998). Planning under resource constraints. In *Proc. 13th European Conference on Artificial Intelligence (ECAI)*, pp. 489–493.

Lagriffoul, F., Dimitrov, D., Saffiotti, A., & Karlsson, L. (2012). Constraint propagation on interval bounds for dealing with geometric backtracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 957–964.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., & Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, *31*(1-3), 59–83.

Li, H. X., & Williams, B. C. (2008). Generative planning for hybrid systems based on flow tubes. In *Proc. 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 206–213.

Liftschitz, V. (1987). On the semantics of STRIPS. In *Proc. 1986 Workshop on Reasoning about Actions and Plans*, pp. 1–9.

Löhr, J., Eyerich, P., Keller, T., & Nebel, B. (2012). A planning based framework for controlling hybrid systems. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms.. *Journal of AI Research*, *12*, 271–315.

Piacentini, C., Alimisis, V., Fox, M., & Long, D. (2015). An extension of metric temporal planning with application to AC voltage control. *Artificial Intelligence*, *229*, 210–245.

Plaku, E., & Hager, G. D. (2010). Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5002–5008.

Pommerening, F., & Helmert, M. (2012). Optimal planning for delete-free tasks with incremental LM-cut. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Pommerening, F., Röger, G., & Helmert, M. (2013). Getting the most out of pattern databases for classical planning. In *Proc. of the 23rd International Joint Conference on AI (IJCAI)*, pp. 2357–2364.

Rankooh, M. F., & Ghassem-Sani, G. (2015). ITSAT: An efficient sat-based temporal planner. *Journal of AI Research*, *53*, 541–632.

Richter, S., & Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 273–280.

Riddle, P., Douglas, J., Barley, M., & Franco, S. (2016). Improving performance by reformulating PDDL into a bagged representation. In *ICAPS Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, pp. 26–36.

Sandewall, E. (1994). *Features and fluents (vol. 1): the representation of knowledge about dynamical systems*. Oxford University Press, Inc., New York, NY, USA.

Scala, E., Haslum, P., & Thiébaux, S. (2016a). Heuristics for numeric planning via subgoaling. In *Proc. 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3228–3234.

Scala, E., Haslum, P., Thiébaux, S., & Ramírez, M. (2016b). Interval-based relaxation for general numeric planning. In *22nd European Conference on Artificial Intelligence (ECAI)*, pp. 655–663.

Scala, E., Ramírez, M., Haslum, P., & Thiébaux, S. (2016c). Numeric planning with disjunctive global constraints via SMT. In *Proc. 26th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 276–284.

Shin, J.-A., & Davis, E. (2005). Processes and continuous change in a SAT-based planner. *Artificial Intelligence*, *166*, 195–254.

Slaney, J., & Thiébaux, S. (2001). Blocks world revisited. *Artificial Intelligence*, *125*. `http://users.cecs.anu.edu.au/~jks/bw.html`.

Srivastava, B., & Kambhampati, S. (1999). Scaling up planning by teasing out resource scheduling. In *Proc. 5th European Conference on Planning (ECP)*, pp. 172–186.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russel, S., & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pp. 639–646.

Tate, A., Drabble, B., & Kirby, R. (1994). O-Plan2: An open architecture for command, planning and control. In Zweben, M., & Fox, M. (Eds.), *Intelligent Scheduling*, pp. 213–239. Morgan-Kaufmann.

Thiébaux, S., Coffrin, C., Hijazi, H., & Slaney, J. K. (2013). Planning with MIP for supply restoration in power distribution systems. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

Thiébaux, S., & Cordier, M. (2001). Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *Proc. 6th European Conference on Planning (ECP)*.

Thiébaux, S., & Herzberg, J. (1992). A semi-reactive planner based on a possible models action formalization. In *Proc. 1st International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 228–235.

Thiébaux, S., Hoffmann, J., & Nebel, B. (2005). In defense of PDDL axioms. *Artificial Intelligence*, *168*(1-2), 38–69.

Torralba, A., Alcazar, V., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014). Symba*: A symbolic bidirectional a* planner. In *The 2014 International Planning Competition – Description of Participating Planners*, pp. 105–109.

Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proc. 24th International Joint Conference on AI (IJCAI)*, pp. 1930–1936.

Vallati, M., Magazzeni, D., De Schutter, B., Chrpa, L., & McCluskey, T. L. (2016). Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In *Proc. 30th AAAI Conference on Artificial Intelligence*, pp. 3188–3194.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental AI*, *7*(1), 81–120.

Wächter, A., & Biegler, L. T. (2006). On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57.

Weld, D., & Etzioni, O. (1994). The first law of robotics (a call to arms). In *Proc. 12th National Conference on Artificial Intelligence (AAAI'94)*, pp. 1042–1047.

Winslett, M. (1988). Reasoning about action using a possible models approach. In *AAAI*, pp. 89–93.

Wolfman, S. A., & Weld, D. S. (1999). The LPSAT engine & its application to resource planning. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 310–317.

Yoshizumi, T., Miura, T., & Ishida, T. (2000). A* with partial expansion for large branching factor problems. In *AAAI*, pp. 923–929.