Optimal Planning with Global Numerical State Constraints

Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux,
Optimisation Research Group, NICTA
Research School of Computer Science, ANU
`first.last@anu.edu.au`

Vikas Shivashankar and **Dana S. Nau**
Dept. of Comp. Sci., and Inst. for Sys. Rsrch.
University of Maryland at College Park
{`svikas,nau`}`@cs.umd.edu`

Abstract

Automating the operations of infrastructure networks such as energy grids and oil pipelines requires a range of planning and optimisation technologies. However, current planners face significant challenges in responding to this need. Notably, they are unable to model and reason about the global numerical state constraints necessary to capture flows and similar physical phenomena occurring in these networks. A single discrete control action can affect the flow throughout the network in a way that may depend on the entire network topology. Determining whether preconditions, goals and invariant conditions are satisfied requires solving a system of numerical constraints after each action application. This paper extends domain-independent optimal planning to this kind of reasoning. We present extensions of the formalism, relaxed plans, and heuristics, as well as new search variants and experimental results on two problem domains.

Motivation

New sensor technologies and data analytics provide a wealth of information and prediction about current and future states of technical systems, and about the environment in which they operate. This is true, for instance, of infrastructure networks such as power grids, transport systems and water networks. Smart use of this information, using automated planning, to optimise the systems' resource usage, improve their operations and increase their reliability, has the potential to generate significant economic and environmental benefits.

However, as already observed by several researchers (Aylett et al. 1998; Thiébaux and Cordier 2001; Piacentini et al. 2013), existing planners miss a key capability necessary to handle such networks and other interconnected physical systems: they are unable to process the numerical constraints governing 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 of all network buses, in a way that depends on the state of all other switches.

The global scope of these constraints make them impractical to formalise and handle using existing planning frameworks. Planning requires solving a set of (possibly nonlinear) numerical constraints after each action application, to determine the values of flows and flow-dependent numerical values across the network. These values are used in action preconditions, goal requirements, and invariant conditions that are necessary to ensure that the system operates reliably. For instance, in power systems, we need to solve a complex system of equations linking real and reactive power flows to bus voltages and phase angles, to determine whether closing a switch would lead any line capacities to be exceeded.

This paper deals with domain-independent optimal planning with such global numerical state constraints. We extend the classical planning formalism, the notion of relaxed plan, and admissible planning heuristics to incorporate these constraints. To mitigate the extra complexity of successor state computation, we introduce a new optimal search strategy using preferred actions. In certain circumstances, which we characterise, it can avoid generating many successor states and substantially reduce computation time. Experimental evaluation on two new planning domains with linear constraints suggests that global numerical constraints still raise substantial challenges for planning research.

Existing Work

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 and Nilsson 1971; Liftschitz 1987; Ginsberg and Smith 1988; Winslett 1988 Thiébaux and Herzberg 1992; Sandewall 1994; Levesque et al. 1997).

However, state constraints have not found their way into efficient implementations of modern planners. Much recent research in planning has focused on improving the algorithmic aspects of plan generation for an impoverished version of STRIPS that does not support state constraints. The universally quantified effects found in ADL variants do not really help, since state constraints do not update variables based on the predecessor state, but are equations relating the values of variables in the same state. In principle these constraints could be compiled away at the cost of much more complex preconditions and goals or longer plans. But this can lead to (exponentially) large conditions to represent, and severe blowup in domain description size or plan length (Thiébaux, Hoffmann, and Nebel 2005). In the International Planning Competition (IPC) series, the Power Supply Restoration (PSR) domain is an example where such blowup occurs (Hoffmann et al. 2006).

PDDL2.2's derived predicates and axioms (Thiébaux, Hoffmann, and Nebel 2005; Hoffmann and Edelkamp 2005) enable the compact encoding of a larger class of boolean constraints, 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 et al. 2005; Helmert 2006) do not include substantial improvements to domain-independent heuristics to deal with complications arising from the presence of axioms. Moreover, we know of no research into *optimal* planning with derived predicates.

Finally and despite the importance of application domains that require them, *numerical* state constraints have been neglected by the planning community. No numerical counterpart to derived predicates has been developed to represent constraints linking numerical and boolean variables. Effective domain-independent heuristics for domains with numerical variants of derived predicates are an open question. Approaches to planning for hybrid dynamical systems, such as domain-predictive control (Löhr et al. 2012), are well suited to model the association of a small number of discrete modes with (dynamical) equations, but not to modeling network flows, as the number of modes required is equal to the number of network configurations, which is prohibitively large.

Nevertheless, several papers describe applications of planning in physically interconnected domains. For example, Aylett et. al (1998) consider managing a chemical plant, while Piacentini et. al (2013) address balancing power in an electrical network, both using planning systems. To reason about network flows, both use an architecture in which the planner interacts with a special-purpose flow solver. In both cases, the planner does not plan for discrete topological changes that affect flow Instead, in the chemical plant case, discrete valve operations are also outsourced to the solver, and the planner handles other aspects of the plant management problem; in the case of power balancing, the problem does not include switching operations.

As another example, Thiébaux et. al (2013) present a mixed-integer programming (MIP) model including numerical power flow equations to generate power supply restoration plans. MIP is well-suited in this case, because there are good short plans with known lengths; but it is poorly suited for generating high quality plans over long or indeterminate horizons. Our work is a step towards making planning an alternative to MIP in such circumstances.

Planning Formalism

We now define the extended planning formalism. The next section illustrates these definitions with concrete examples.

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. Actions' 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. Usually, pre- and goal conditions are restricted to be conjunctions of elementary formulas, which are variable–value equalities.

Our planning formalism extends the classical one with an additional set of state variables and with state constraints. To avoid ambiguity, we call the state variables of the classical part of the model the *primary* variables, and the others the *secondary* variables. Secondary variables do not have to have finite domains. (In our implemented planner, they are reals.) The two kinds of variables interact only via state constraints, which take the form of implications, $\varphi \rightarrow \gamma$, where φ is a logical expression over the primary variables, and γ a constraint on the secondary variables. We call these *switched constraints*, and when the triggering condition φ is true we say the constraint is *active*.

State constraints may appear appear in action preconditions, in the goal, and in a designated set C_{inv} of *invariant constraints*. Precondition and goal constraints must be satisfied when the action is applied and the goal is achieved, respectively, while the invariant constraints must be satisfied in all states visited by the plan. Invariant constraints typically serve two purposes: they define (sometimes uniquely, sometimes not) the values of secondary variables, and they impose constraints on these variables as required by the domain. For example, in power systems, they include both the equations that determine the power flows in each state, and operational constraints such as staying within line and generator capacity limits. In meshed networks, there can be multiple power flow solutions.

Secondary variables are not directly assigned by action effects, nor set in the initial state. Instead, the planner is free to choose in each state any assignment to them that satisfies the constraints active in that state. The condition for action applicability and goal achievement (Definition 2(*ii*)) ensures that such an assignment for each state exists, without forcing the planner to commit to one in particular until the action to be taken has been chosen. This treatment of the secondary variables is also different from traditional metric planning, in which numeric variables are locally updated by action effects like other variables, but which has no mechanism for compactly expressing global state constraints. Hence, these extensions to classical planning are orthogonal.

Definition 1. *A* state *is an assignment s of values to all variables in a set V of primary state variables. We write $s(v)$ for the value of v in s, and $s(\varphi)$ for the value of a formula φ over the primary variables. If C is a set of switched constraints, then the set of* active constraints *(from C) in s is*

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

A state s is valid *iff* $\mathrm{active}(C_{\mathsf{inv}}, s)$ *is satisfiable.*

Note that $\mathrm{active}(C_{\mathsf{inv}}, s)$ is a set of (non-switched) constraints over the secondary variables only. The solver that checks satisfiability of these does not need to deal with switched constraints, or indeed any aspect of the primary variables. For example, if the constraints are linear inequalities over the reals, this test can be carried out with a standard LP-solver. (This is what our implemented planner uses.)

Definition 2. *A* partitioned condition *is a pair* (c_P, c_S), *where* c_P *is a set of elementary formulas (variable–value equalities) over the primary variables and* c_S *a set of switched constraints.* (c_P, c_S) holds *in state* s *iff (i)* $s(c_P) =$ *true and (ii)* $\text{active}(c_S \cup C_{\mathsf{inv}}, s)$ *is satisfiable.*

Action preconditions and the goal are partitioned conditions. As with the primary and secondary state variables, we call the two parts of these partitioned conditions the primary and secondary precondition and goal, respectively.

As usual, an action a is applicable in a state s iff its precondition $\text{pre}(a) = (\text{pre}_P(a), \text{pre}_S(a))$ holds in s. We say that a is *allowed* in s if a is applicable and applying it leads to a valid state. Although we restrict primary conditions to be conjunctions of elementary equalities, we can encode more complex conditions on primary variables through constraints. For example, if $p_1, \ldots, p_k$ are propositional variables, and C_{inv} includes the constraints $(p_i = \textit{true}) \rightarrow (v_i = 1)$ and $(p_i = \textit{false}) \rightarrow (v_i = 0)$ for $1 \leq i \leq k$, then $\text{pre}_S(a) = \{\textit{true} \rightarrow (\sum_{i=1}^{k} v_i) = 1\}$ enforces the precondition that exactly one of $p_1, \ldots, p_k$ is true.

Action effects are purely classical, and (directly) affect only the primary variables. Action a's effects, $\text{eff}(a)$, is a set of variable-value assignments, where each primary variable occurs at most once. Applying $\text{eff}(a)$ to a state s leads to a state s' where each primary variable v such that $(v, x) \in \text{eff}(a)$ has the (unique) value x assigned by a, and any variable not appearing in $\text{eff}(a)$ keeps its value from s.

Definition 3. *A planning problem consists of:*

- *A set* V_P *of primary variables.*
- *A set* V_S *of secondary variables.*
- *A set* A *of actions, each action* a *defined by:*
  - *a partitioned* $\text{pre}(a) = (\text{pre}_P(a), \text{pre}_S(a))$*, and*
  - *an effect* $\text{eff}(a)$*, which is a set of primary variable-value assignments.*
- *A set* C_{inv} *of invariant switched constraints.*
- *An initial assignment* s_0 *of values to all variables in* V_P*.*
- *A partitioned goal condition* $G = (G_P, G_S)$*.*

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

π's cost is the sum of the costs of its actions. The *optimal planning problem* is to generate plans with guaranteed minimum cost. In the simple case, actions costs are constants. Later in the paper we also consider the case when the cost of an action is a function of the state in which it is applied.

Domain Examples

To illustrate these definitions, we examine the two domains we use in our experiments. In the following, constants are distinguished from variables by an overline bar ($\bar{c}$ vs v).

Hydraulic Blocks World

For our first domain, whose interest is mostly didactical, consider a Blocks World with a fixed number m of towers, where the n blocks have different weights. Each tower sits on a piston inside a vertical cylinder sticking up from a sealed reservoir of hydraulic fluid (Figure 1). The problem is to reach a goal configuration of blocks without letting any piston go above the top or below the bottom of its cylinder.

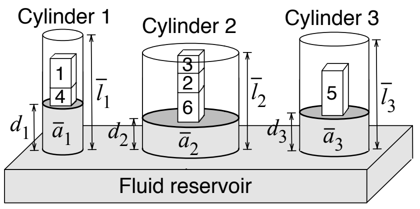


Figure 1: Hydraulic Blocks World

The primary variables are booleans representing what cylinder a block is in and what object it's on. This can be achieved using the usual predicates on, $handempty$ and $clear$, and additional propositions in_{ik} stating that block i is in cylinder k. Actions are the usual $pickup/putdown$ $unstack/stack$, augmented to indicate what cylinder the block is in, e.g., $unstack(i, j, k)$ takes block i off block j in cylinder k. There are no secondary preconditions, but a variant with such preconditions could for instance prevent the robot arm from reaching too far into a cylinder.

The key secondary variable is the height d_k of fluid in each cylinder k, and the main safety constraint is that the piston in each cylinder k does not go any higher than the height $\bar{l}_k$ of the cylinder nor any lower than the bottom cylinder:

$$0 \leq d_k \leq \bar{l}_k \quad k \in 1 \ldots m$$

The fluid heights depend on the total weight p_k of the tower in each cylinder k. To compute p_k, we introduce secondary variables $p_{i,k}$, $i = 0, \ldots, n$, $k = 1, \ldots, m$, representing the sum of the weights $\bar{w}_j$ of all blocks in the range $j \in 1, \ldots, i$ that are in cylinder k. Each $p_{i,k}$ can be computed iteratively from $p_{i-1,k}$ using two switched constraints linking from the primary variable in_{ik}, and an unswitched base case:

$$\begin{array}{lcll} & & p_{0,k} = 0 & \\ in_{ik} = false & \rightarrow & p_{i,k} = p_{i-1,k} & i \geq 1 \\ in_{ik} = true & \rightarrow & p_{i,k} = p_{i-1,k} + \bar{w}_i & i \geq 1 \end{array}$$

The total weight of the blocks in cylinder k is $p_k = p_{n,k}$. Now d_k can be determined via the following system of equations, which states that (a) the total amount of fluid $\bar{v}$ in the cylinders never changes, (b) the downward force f_k at the bottom of cylinder k equals the weight p_k of the tower plus the weight of the fluid in the cylinder (the fluid density $\bar{\rho}$ times the fluid's volume, where $\bar{a}_k$ is the cylinder's cross-sectional area), and (c) the pressure (force per unit area) at the bottom of a cylinder is the same for each cylinder:

$$\begin{array}{lll} \sum_{k=1}^{m} \bar{a}_k d_k = \bar{v} & k = 1, \ldots, m & \text{(a)} \\ f_k = p_k + \bar{\rho} \bar{a}_k d_k & k = 1, \ldots, m & \text{(b)} \\ f_k / \bar{a}_k = f_{k+1} / \bar{a}_{k+1} & k = 1, \ldots, m-1 & \text{(c)} \end{array}$$

Power Supply Restoration

Our second domain exemplifies the kind of useful problem our approach enables planning to address. The power supply restoration problem we consider is to reconfigure a power network to isolate known faults and resupply a given set of loads. In contrast to the PSR benchmark of the IPC, our version handles numeric power flows and capacity constraints, which is vital to network reliability.

The network is a graph $\langle \mathcal{B}, \mathcal{L} \rangle$ whose nodes are buses $i \in \mathcal{B}$ supporting constant consumer loads $\bar{l}_i$; a subset of buses ($\mathcal{G}$) supply variable generation g_i, and another subset ($\mathcal{F}$) are the faulty ones. Edges $(i, j) \in \mathcal{L}, i < j$ are electric lines equipped with switches that, when open, disable the edge. A bus is fed (f_i) iff there is a path (of lines with closed switches) from a generator bus (in $\mathcal{G}$) to it, in which case its entire load must be supplied. Lines (resp. generator buses) also have capacities that constrain the power p_{ij} that can flow through them (resp. the generation g_i produced).

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 output. We model this using the freedom our formalism gives the planner to assign the underconstrained secondary g_i variables, rather than explicit actions. Hence, in a meshed network configuration there can be multiple solutions to the active state constraints.

There are three main types of invariant constraints (Thiébaux et al. 2013). The first define the line power flows. Whilst our formalism allows for non-linear constraints such as the steady state AC power flow equations, for simplicity, we use here the well-known linear DC power flow model whereby the power flow p_{ij} from i to j is proportional to the difference $\theta_i - \theta_j$ between the phase angles of the buses the line connects and to the line suceptance $\bar{b}_{ij}$. Of course open lines have no flow, hence the following switched constraints:

$$\begin{array}{llll} y_{ij} = \textit{true} & \rightarrow & p_{ij} = -\bar{b}_{ij}(\theta_i - \theta_j) & (i,j) \in \mathcal{L} \\ y_{ij} = \textit{false} & \rightarrow & p_{ij} = 0 & (i,j) \in \mathcal{L} \end{array}$$

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

$$\begin{array}{lrl} g_i + \sum_{j:(j,i)\in\mathcal{L}} p_{ji} = \bar{l}_i f_i + \sum_{j:(i,j)\in\mathcal{L}} p_{ij} & i \in \mathcal{B} & \text{(a)} \\ f_i = 0 & i \in \mathcal{F} & \text{(b)} \\ f_i = 1 & i \in \mathcal{G} \setminus \mathcal{F} & \text{(c)} \\ y_{ij} = \textit{true} \rightarrow f_i = f_j & (i,j) \in \mathcal{L} & \text{(d)} \end{array}$$

The rest of the constraints encode capacity limits on the generation (e) and power flow (f).

$$\begin{array}{lrl} 0 \le g_i \le \bar{g}_i & i \in \mathcal{B} & \text{(e)} \\ -\bar{p}_{ij} \le p_{ij} \le \bar{p}_{ij} & (i,j) \in \mathcal{L} & \text{(f)} \end{array}$$

Switching actions cause transient phenomena which can threaten network stability. For example, before connecting two buses by closing a line, the difference between their standing phase angles should be sufficiently small; if not, a generation redispatch is necessary to reduce it (Hazarika and Sinha 1999). This can be captured by adding a secondary precondition to the closing action for each line (i, j) constraining the difference $\theta_i - \theta_j$ to lie within safety bounds. However, we have not used it in our experiments.

A restoration plan is a sequence of switching operations. 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. We can capture both objectives 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. We consider both variants of the problem.

Relaxed Plan Heuristic

To derive heuristics to guide search for optimal plans, we use the well-known idea of optimal relaxed planning. We define a notion of *relaxed rechability* that shares the monotonicity property of classical delete-relaxation. From this, we obtain analogues of the classical $h^{\max}$ and h^+ heuristics for our more expressive planning formalism. To compute h^+, we use the landmark-based algorithm of Haslum, Slaney and Thiébaux (2012). By changing the algorithm slightly, we can even get an analogue of the LM-Cut heuristic (Helmert and Domshlak 2009) for problems with unit-cost actions.

Defining the Relaxation

Several researchers (e.g., Gregory et al. 2012) have noted that in classical, propositional planning, delete-relaxation 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, x) adds the new value, x, to the set, without removing any existing value. This extends straightforwardly to logical expressions: writing $s^+(\varphi)$ for the set of values of a formula φ in a relaxed state s^+, we have

- *true* $\in s^+(v = x)$ iff $x \in s^+(v)$;
- *false* $\in s^+(v = x)$ iff $\exists y \in s^+(v) : y \neq x$;
- *true* $\in s^+(\neg\varphi)$ iff *false* $\in s^+(\varphi)$, and vice versa;
- $v \in s^+(\varphi \circ \psi)$ iff $\exists v' \in s^+(\varphi), v'' \in s^+(\psi) : v = (v' \circ v'')$ for any connective $\circ$.

The relaxed planning problem is defined by replacing $s(\varphi) = \textit{true}$ with *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 minimal relaxed plan cost is a lower bound on minimal real plan cost. The key property ensuring this is that the relaxation is *monotonic*: for any formula φ and relaxed state s^+, if $v \in s^+(\varphi)$ ($v \in \{\textit{false}, \textit{true}\}$) then $v \in s^{+\prime}(\varphi)$ for any relaxed state $s^{+\prime}$ reachable from s^+ by relaxed action applications. Extending this to our setting requires one addition: determining which constraints are active in a relaxed state.

Definition 4. *For a relaxed state s^+ and set of switched constraints C, the relaxed active constraints (from C) in s^+ are* $\mathrm{active}^+(C, s^+) = \{\gamma \mid \varphi \to \gamma \in C,\ \mathit{false} \notin s^+(\varphi)\}$.

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.

We define the relaxation of a planning problem by replacing states, state updates, evaluation of primary preconditions and goals, and the sets of active constraints by their relaxed counterparts. The analogue of monotonicity in the classical planning relaxation ("true conditions remain true") is that the set of active constraints can only shrink along a relaxed action sequence; hence, satisfiable sets of switched constraints remain satisfiable.

Lemma 5. *Let s^+ be a relaxed state, $s^{+\prime}$ the state that results from (relaxed) application of an action sequence $a_1, \ldots, a_n$, and (c_P, c_S) a partitioned condition. If (c_P, c_S) holds in s^+, then it also holds in $s^{+\prime}$.*

Proof. Monotonicity of the primary condition c_P follows directly from monotonicity of the classical relaxation.

Let $\varphi \to \gamma$ be a switched constraint in c_S. Since the classical relaxation is monotone, *false* $\in s^{+\prime}(\varphi)$ only if *false* $\in s^+(\varphi)$. Thus if *false* $\notin s^+(\varphi)$ then *false* $\notin s^{+\prime}(\varphi)$, and thus $\mathrm{active}^+(c_S, s^{+\prime}) \subseteq \mathrm{active}^+(c_S, s^+)$. Since removing constraints cannot cause a contradiction, if $\mathrm{active}^+(c_S, s^+)$ is satisfiable then so is $\mathrm{active}^+(c_S, s^{+\prime})$. □

Computing Heuristics

The monotonicity property implies that we can build a relaxed planning graph following the same procedure as in classical planning. Fact layers are simply relaxed states, each action layer includes all (not previously applied) actions that are allowed in the current relaxed state, and the next relaxed state is the result of applying all their effects.[1] 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.

This construction provides us with a (yes/no) relaxed reachability test, which enables us to compute h^+ using the iterative landmark algorithm (described below). In addition, assuming a cost of 1 for every action, the number of action layers needed before the goal condition holds provides an analogue of the $h^{\max}$ heuristic. In our experiments on the power supply restoration problem with constant action costs, we found using h^+ to be more effective than using $h^{\max}$ or no heuristic, in terms of total runtime.

Optimisations. Recall that an action a with $\mathrm{pre}(a) = (\mathrm{pre}_P(a), \mathrm{pre}_S(a))$ is allowed in a relaxed state s^+ iff *true* $\in s^+(\mathrm{pre}_P(a))$ and both $\mathrm{active}^+(\mathrm{pre}_S(a) \cup C_{\mathsf{inv}}, s^+)$ and $\mathrm{active}^+(C_{\mathsf{inv}}, s^{+\prime})$ are satisfiable, where $s^{+\prime}$ is the result of applying $\mathrm{eff}(a)$ to s^+. Checking satisfiability requires the external constraint solver, hence is much more expensive computationally than the primary precondition check.

[1]There is no need for explicit no-ops, since previously achieved values remain under the value accumulating semantics.

However, since the set of active constraints can only shrink with the relaxed application of more actions (Lemma 5), $\mathrm{active}^+(C_{\mathsf{inv}}, s^{+\prime}) \subseteq \mathrm{active}^+(C_{\mathsf{inv}}, s^+) \subseteq \mathrm{active}^+(\mathrm{pre}_S(a) \cup C_{\mathsf{inv}}, s^+)$. Thus, we do not need to perform the second call, since it will succeed if the first one does. Furthermore, if actions have no secondary preconditions, i.e., $\mathrm{pre}_S(a) = \emptyset$ for all a, we can even eliminate the first call. Because the state s for which we are computing the heuristic is valid, we know that $\mathrm{active}(C_{\mathsf{inv}}, s)$ is satisfiable. Again due to Lemma 5, this means $\mathrm{active}^+(C_{\mathsf{inv}}, s^+)$ is also satisfiable for any s^+ that is relaxed-reachable from s.

The two domains in our experiments have no secondary action preconditions, so we use this optimisation to substantially reduce the number of calls to the external constraint solver. We need only one call, to check if the secondary goals have been achieved, for each relaxed reachability test.

Computing h^+. The iterative landmark algorithm (Haslum, Slaney, and Thiébaux 2012) computes a set of disjunctive action landmarks (Karpas and Domshlak 2009) such that a minimum-cost hitting set over this collection is an optimal relaxed plan, whose cost is h^+. 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 in our setting. We use all algorithm improvements proposed by Haslum, Slaney and Thiébaux (2012); we also use the integer programming solver (Gurobi) to compute cost-optimal hitting sets.

Restricting the algorithm to generating only disjoint landmarks, we obtain a faster-to-compute but potentially weaker heuristic. With unit-cost actions, this is equivalent to the LM-Cut heuristic (Bonet and Helmert 2010).

Computing $h+$ Incrementally. 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 found for s, and 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.

Preferred Actions in A* Search

To compute optimal plans, we rely on A* search. Computing an optimal relaxed plan at every generated state is expensive, but can provide much more information than just an admissible heuristic estimate. We combine 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. This situation is char-

```
 1: procedure PrefPEA$^\star$
 2:   Set $open = \{(s_0, 0, h(s_0), \mathsf{pref}(s_0))\}$, $closed = \emptyset$.
 3:   while $open \neq \emptyset$ do
 4:     Select $n = \min_{\prec} open$, where
        $(n \prec n') \equiv \quad (f(n) < f(n'))$
                    $\vee\ (f(n) = f(n') \wedge h(n) < h(n'))$
                    $\vee\ (f(n) = f(n') \wedge h(n) = h(n')$
                       $\wedge\ \mathsf{pref}(n) \neq \emptyset \wedge \mathsf{pref}(n') = \emptyset))$
 5:     if $n$ is a goal state then return $n$.
 6:     if $\mathsf{pref}(n) \neq \emptyset$ then
 7:       Select $a \in \mathsf{pref}(n)$, remove $a$ from $\mathsf{pref}(n)$.
 8:       Generate $s'$ from $n$ through $a$.
 9:       NewState$(s', g(n) + \mathrm{cost}(a))$
10:     else
11:       for each non-preferred successor $(a', s')$ of $n$ do
12:         NewState$(s', g(n) + \mathrm{cost}(a'))$
13:       Move $n$ to closed.
14:   return null.

15: procedure NewState$(s, g)$
16:   if $\nexists n' \in open \cup closed$ with state $s$ then
17:     Add $(s, g, h(s), \mathsf{pref}(s))$ to open.
18:   else if $g < g(n')$ then
19:     Set $g(n') = g$ and update parent pointer.
20:     if $n' \in closed$ then Move $n'$ back to open.
```

Figure 2: Partial Expansion A* with Preferred Actions. The NewState subroutine handles updating of path cost and node re-opening, as in standard A*.

acteristic of many planning heuristics and problems, including the power supply restoration problem that we tackle.

The preferred actions (also known as "helpful") in a state s are actions in the relaxed plan computed 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 and Helmert 2009; Hoffmann 2000), but as far as we are aware there has been no use of this source of information in optimal search.

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, and 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. However, 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 is sometimes possible to determine the partial successor set without generating and evaluting all successors.

We adopt the idea of PEA*, but stage node expansion by the preferredness of successors instead of f-value. Pseudocode for the procedure is shown in Figure 2. When a state is generated, its set of preferred actions are found (as a side effect of computing its heuristic value) 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. We prioritise 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 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'.

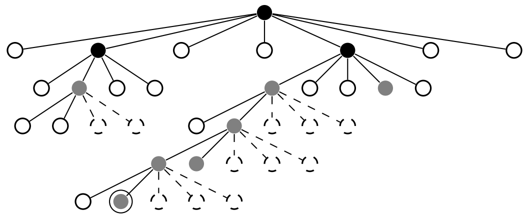
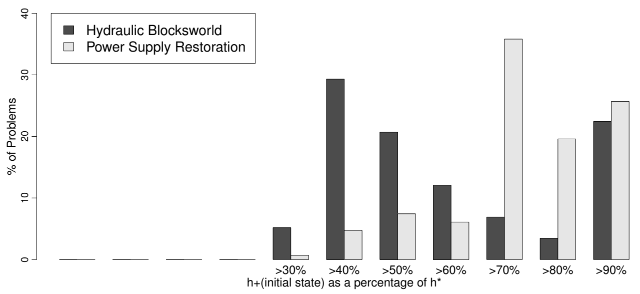


Figure 3: Illustration of PrefPEA*. Black nodes have $f(n) < f^\star$; these must be fully expanded. Gray nodes have $f(n) = f^\star$; some of these will be expanded, and may be partially expanded. White nodes have $f(n) > f^\star$. The dashed part represents non-preferred successor nodes that are never 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.

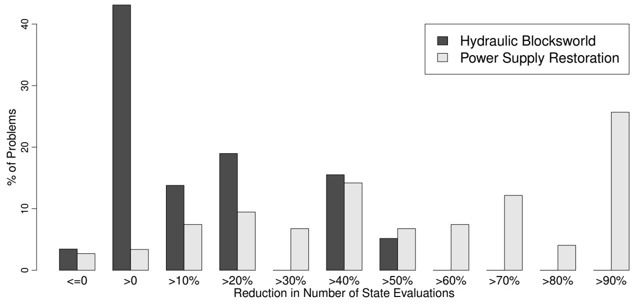
Impact on Search Efficiency

The benefit of PrefPEA* is limited to avoiding heuristic evaluation of some siblings of nodes expanded in the $f^\star$ layer. This is illustrated in Figure 3. However, if the heuristic is 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. Our two example domains illustate these two cases. Because PrefPEA* changes the order of expansion, and because remaining ties are still broken arbitrarily, it is possible for PrefPEA* to be "unlucky" and expand more nodes than A* on a particular problem instance, even if the aggregate results are better. We observed this in a small number of cases ($< 2\%$).

For the power supply restoration (PSR) domain, we use a semi-rural network (from Thiébaux et al. 2013) and 171 problems with 1–3 faults. We use a constant (state-independent) cost of 1 for each action. For hydraulic blocksworld (HBW), we use 80 problems with 4–7 blocks and 3–5 cylinders. Parameters such as block weights, cylinder heights and areas, etc. are set randomly, with the aim of creating the problems that are solvable, but where the con-



(a)



(b)

Figure 4: (a) Accuracy of the relaxed plan heuristic, measured by $h^{+}(s_0)$ as a percentage of f^{*}. (b) Distribution of the reduction in number of state evaluations using PREFPEA* compared to plain A*. (A "reduction" < 0 means PREFPEA* evaluates more states, as a result of expanding more nodes. This increase is never more than 15%.)

straints force plans to be different than in the unconstrained case. Note that the addition of both a limit on the number of towers and the global numeric state constraints makes this domain much harder than the usual STRIPS Blocksworld. (Blind A* search solves only 4 out of 20 6-block problems, while A* search with the h^{+} heuristic solves 17.)

With plain A*, the planner solves 150 PSR problems and 62 HBW problems (within 30 minutes). With PREFPEA*, it solves 2 more problems in the PSR domain.

Figure 4(a) shows how close the heuristic estimate in the initial state is to the optimal plan cost (i.e., $h^{+}(s_0)$ as a percentage of $f^{\star}$). It is more accurate in PSR than in HBW. Furthermore, the average branching factor in PSR is 26.7, while in HBW it is only 1.99. Given this, the difference in the reduction in number of state evaluations between the two domains, shown in Figure 4(b), is to be expected. In PSR, using PREFPEA* saves 42.8% of state evaluations compared to using A* (aggregated over problems solved with both), and over 90% on a quarter of the instances. Since heuristic computation accounts for 95% of total runtime, on average, in this domain, this translates into a roughly proportional 41.6% reduction in aggreated runtime.

In HBW, the aggregate reduction in evaluations is only 3.4%, and because heuristic evaluations are also much faster in this domain (averaging only 16.2% of total time), this does not lead to any reduction in total runtime.

State-Dependent Action Costs

Recall that the primary objective of power supply restoration is to maximise the load supplied over time (Thiébaux et al. 2013). Equivalently, we can say it is to minimise the *un*supplied load at each plan step. This can be expressed as a sum of action costs, but those costs must depend on the state in which the action is applied.

Formulating state-dependent action costs in unrestricted numeric PDDL (Fox and Long 2003) is easy, since the problem metric can be any fluent expression, and actions can have arbitrarily complex (and conditional) effects on numeric fluents. Another way to express state-dependent action costs is through PDDL3's preferences in action preconditions (Gerevini et al. 2009), which incur a penalty (cost) when applying an action in a state where the preference is unsatisfied. However, optimal planning with state-dependent action costs has received very little attention. The optimal MIPS-BDD planner handles an expressive fragment of PDDL3's preferences, but not preferences in action preconditions (Edelkamp 2006). The recent LTOP planner (Tierney et al. 2012) extends optimal temporal planning to complex objectives, including action costs that are a function of the action's duration, but not of state. We are not aware of any optimal planner for the kind of problem we consider.

Formalism. For an action's state-dependent cost, we use a finite sum of conditional costs, $\mathrm{cost}(a) = \{(\varphi_i, c_i)\}_{i=1\ldots k} + c_0$, where φ_i is a (partitioned) condition and c_i is a positive constant (c_0 is an unconditional constant cost, which may be zero). The cost of applying a in a state s is $\mathrm{cost}(a, s) = \sum\{c_i \mid (\varphi_i, c_i) \in \mathrm{cost}(a), s(\varphi_i) = \textit{true}\} + c_0$. Incorporating this cost in a forward state-space search is trivial, since the state in which each action is applied is fully known.

In the power supply restoration problem, the conditional costs are the same for each action: $\{(f_i = 0, \bar{l}_i) \mid i \in \mathcal{B}\}$. (Recall that f_i is a constrained variable for whether bus i is fed, and $\bar{l}_i$ is the load attached to the bus.) We add to this a constant cost of 1, for the secondary objective of minimising the number of switching operations.

Extending h^{+}. Extending h^{+} to accurately estimate the state-dependent cost-to-go is far more challenging. The way we express state-dependent action costs is essentially a form of conditional action effects. Hence, we could apply the same kind of problem transformation that is used to compile away conditional effects to reduce the problem to one that has only constant action costs: Replace each action a with one copy a_X for each subset $X \subseteq \mathrm{cost}(a)$. Each copy's cost is constant, $\mathrm{cost}(a_X) = \sum\{c_i \mid (\varphi_i, c_i) \in X\} + c_0$, i.e., the sum of costs in the subset X, and its precondition is $\mathrm{pre}(a_X) = \mathrm{pre}(a) \wedge \{\neg\varphi_i \mid (\varphi_i, c_i) \in \mathrm{cost}(a) - X\}$, ensuring it is applicable only in states where the conditional costs not in X would not apply. However, this clearly can lead to an exponential blow-up. In our PSR problems there can be as many as 44 loads to supply, so this full compilation would split each action into 2^{44} copies.

To avoid this, we take an approach similar to optimal relaxed planning with conditional effects (Haslum 2013). First, we relax the problem further by assuming that every action will be applied in a least-cost state, making its cost

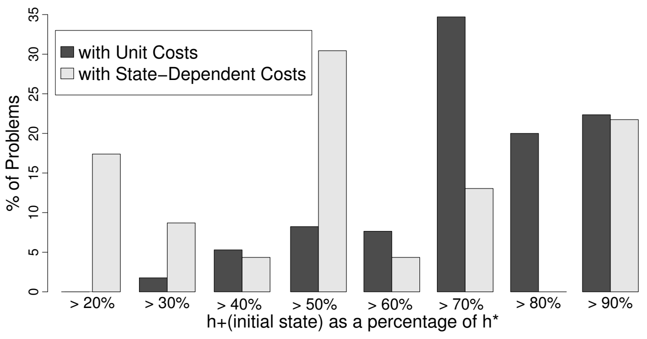


Figure 5: Accuracy of the relaxed plan heuristic with state-dependent action costs, and with unit cost actions. The accuracy in the unit-cost case here is different from that in Figure 4(a), because the problem set is smaller.

constant. Then, we compute an optimal relaxed plan (action set) under this assumption, using the algorithm in the earlier section. Next, we use systematic branch-and-bound search to try to sequence the resulting set of actions so that the assumed cost is actually achieved. If sequencing is not possible, we split each action a in the set that could not achieve its assumed cost, but only into two copies. We select one of the conditional costs $(\varphi_i, c_i) \in \mathrm{cost}(a)$ that was triggered in one of the failed attempts to sequence the action set, and split the action on whether this cost applies or not. That is, one copy, a^0, gets the added precondition $\neg\varphi_i$ (corresponding to any choice of X with $(\varphi_i, c_i) \notin X$) and the other copy, a^1, has c_i added to its unconditional cost (corresponding to any choice of X with $(\varphi_i, c_i) \in X$). From both copies, we remove the selected conditional cost. This process is then repeated from the first step, using the modified problem, until the sequencing step succeeds.

Results

The iterated relaxed plan construction is very effective at reducing the number of actions copies: For the initial states of single-fault PSR problems, the average number of iterations is 3.62 and the average number of action copies created less than ten. In contrast, full compilation could create as many as 2^{44} copies of each action.

However, the heuristic is not time-efficient. The time to evaluate a single state is, on average, 6.85 times higher than in the unit-cost case. This reflects both the repeated $h+$ computation and the overhead of sequencing the action set. The heuristic also becomes less accurate than it is with constant costs, as shown in Figure 5. As a result, blind search solves more of these problems than search with the heuristic.

Conclusion

We have introduced an extended planning formalism that combines the classical, discrete state/action planning model with systems of constraints. It can model 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. This can be seen as a special case of *semantic attachment* (Dornhege et al. 2009), in which the semantically attached symbols (our secondary variables) are restricted to occur in preconditions. However, because our formalism provides a precise way of describing the interaction between the attachements and the primary variables, via switched constraints, we are able to derive domain-independent problem relaxations that incorporate the result of the semantically attached "procedure" in a meaningful way. From this relaxation, existing techniques can be used almost off-the-shelf to derive a variety of admissible heuristics. We believe this methodology can be applied to other non-classical planning formalisms, e.g., the Planning-Modulo-Theory framework (Gregory et al. 2012).

To solve problems in this formalism, we couple a planner with an external solver, in a way that retains the advantage of heuristic search whilst allowing reasoning within any type of theory (e.g. linear or non-linear numerical equations, chemical reactions, or qualitative spatial calculi). For instance, whilst we model the power supply restoration (PSR) problem with a linear DC power flow model, and use an LP solver in our experiments, nothing prevents us from replacing it with an AC power flow solver, as used by Piacentini et. al (2013)—other than perhaps the lack of convergence guarantees offered by these solvers in the general case.

Piacentini et al. (2013) use a similar, coupled architecture to address the power balancing problem. However, their heuristic relies on knowing certain pre-computed values that depend on some primary variables (including the switch states) and are not updated as these primary variable change—which makes it difficult to handle switching operations. In contrast, our approach naturally integrates the external solver within the heuristic, enabling us to plan with arbitrary primary variable changes, including switching.

The type of planning problems we consider are hard. On the PSR problem, our planner runs at least two orders of magnitude slower than the MIP-based solution by Thiébaux et al. (2013). To improve planning performance we have used, and extended, several ideas from the planning and search literature. Our PREFPEA* algorithm exploits preferred actions in optimal search. Although preferred actions have been widely used in non-optimal planning, we are not aware of any previous optimal planner that makes effective use of them. PREFPEA* can significantly outperform A* when the branching factor is high and the heuristic function is accurate but computationally expensive. This occurs, for example, in our PSR domain, but can also be found in many cases of purely classical planning.

Finally, we considered the extension of optimal planning to state-dependent action costs. Although there has been a recent shift away from simple metrics like plan length towards more general notions of plan cost, no optimal planner that we know of has tackled problems where the cost of an action is conditional on the state in which it is taken. Our results show that much more work is needed for optimal planning with state-dependent action costs to match the recent advances of its constant-costs counterpart.

Acknowledgements This work was supported in part by ARO grant W911NF1210471, ONR grant N000141210430, and ARC project DP140104219, “Robust AI Planning for Hybrid Systems”. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

References

Aylett, R.; Soutter, J. K.; Petley, G. J.; and Chung, P. W. H. 1998. AI planning in a chemical plant domain. In *ECAI*, 622–626.

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

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *ICAPS*.

Edelkamp, S. 2006. Optimal symbolic PDDL3 planning with MIPS-BDD. In *5th International Planning Competition Booklet*. Available at http://zeus.ing.unibs.it/ipc-5/.

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

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

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

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

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.* 173(5-6):619–668.

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

Green, C. C. 1969. Application of theorem proving to problem solving. In *IJCAI*, 219–240.

Gregory, P.; Long, D.; Fox, M.; and Beck, C. 2012. Planning modulo theories: Extending the planning paradigm. In *ICAPS*, 65–73.

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

Haslum, P. 2013. Optimal delete-relaxed (and semi-relaxed) planning with conditional effects. In *IJCAI*, 2291–2297.

Hazarika, D., and Sinha, A. 1999. An algorithm for standing phase angle reduction for power system restoration. *IEEE Trans. Power Systems* 14(4):1213–1218.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What’s the difference anyway? In *ICAPS*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *JAIR* 24:519–579.

Hoffmann, J.; Edelkamp, S.; Thiébaux, S.; Englert, R.; dos S. Liporace, F.; and Trüg, S. 2006. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *JAIR* 26:453–541.

Hoffmann, J. 2000. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *ISMIS*, 216–227.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. Golog: A logic programming language for dynamic domains. *J. Log. Program.* 31(1-3):59–83.

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

Löhr, J.; Eyerich, P.; Keller, T.; and Nebel, B. 2012. A planning based framework for controlling hybrid systems. In *ICAPS*.

Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2013. Combining a temporal planner with an external solver for the power balancing problem in an electricity network. In *ICAPS*.

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

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *ICAPS*, 273–280.

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

Thiébaux, S., and Cordier, M. 2001. Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *ECP*.

Thiébaux, S., and Herzberg, J. 1992. A semi-reactive planner based on a possible models action formalization. In *AIPS*, 228–235.

Thiébaux, S.; Coffrin, C.; Hijazi, H.; and Slaney, J. K. 2013. Planning with MIP for supply restoration in power distribution systems. In *IJCAI*.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.* 168(1-2):38–69.

Tierney, K.; Coles, A.; Coles, A.; C., K.; Britt, A.; and Jensen, R. 2012. Automated planning for liner shipping fleet repositioning. In *ICAPS*, 279–287.

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

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *AAAI*.