On Stubborn Sets and Planning with Resources

Anna Wilhelm and **Marcel Steinmetz** and **Jörg Hoffmann**
Saarland Informatics Campus
Saarland University
Saarbrücken, Germany
s8anwilh@stud.uni-saarland.de;{steinmetz,hoffmann}@cs.uni-saarland.de

Abstract

Stubborn sets are a well-established technique to admissibly prune permutable parts of forward search in classical planning. But what about planning with resources? The identification of effective stubborn sets relies on non-interfering actions. Yet, a priori, all actions affecting the same resource interfere. We show how to exploit the fact that, nevertheless, with commutativity of addition and subtraction, many resource-affecting action sequences are permutable. We design suitable notions of stubborn sets for planning with resources, with and without resource production. We show empirically, on classical IPC benchmarks with discrete resource variables, that our new pruning methods are often, and sometimes dramatically, superior to previous ones. Together with a novel way of automatically identifying the resource variables, this result holds under IPC conditions.

Introduction

Stubborn sets admissibly prune permutable parts of forward search. They were invented in verification (Valmari 1989; Godefroid and Wolper 1991; Edelkamp, Leue, and Lluch-Lafuente 2004), and later adopted to planning (Alkhazraji et al. 2012; Wehrle et al. 2013; Wehrle and Helmert 2014; Winterer et al. 2017). Stubborn set analysis identifies, in any given search state s, a *safe* subset A_s of actions, where branching only over A_s suffices to guarantee optimality. In a nutshell, the idea is to include into A_s (a) actions a that must be used to achieve some part of the goal, and (b) all actions that *interfere* with these a, i. e., whose preconditions/effects are in conflict with a. Given this, actions outside A_s can just as well be applied later on, i. e., we can ignore them at s.

But what about planning with resources? Many planning problems naturally involve the management of resources like fuel or energy or the amount of space in an elevator. Hence the consideration of resources has a long tradition in planning, e. g. (Ghallab and Laruelle 1994; Koehler 1998; Do and Kambhampati 2001; Srivastava, Kambhampati, and Do 2001; Haslum and Geffner 2001; Nakhost, Hoffmann, and Müller 2012; Coles et al. 2013). Even in the classical IPC benchmarks, i.e. the by far most populated PDDL level 1 part of the IPC, where continuous resources are not supported, many domains contain discrete resource variables.

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Our starting observation is that, in planning with resources, classical notions of interference are unnecessarily strict. Two actions interfere, in particular, if they may disable each other's precondition. This is certainly the case for any two actions a and a' that consume the same resource. Yet so long as there are sufficient resources to apply both a and a', and the two actions do not interfere in any other way, their ordering does not matter. More generally, given the commutativity of addition and subtraction, many resource-affecting action sequences are permutable despite numerous syntactic interactions on the resource variables.

There are several variants of planning with resources, which lead to different stubborn set analyses. In particular, we consider the simple but wide-spread case, sometimes called *resource-constrained planning* (RCP) (Nakhost, Hoffmann, and Müller 2012), where resources can only be consumed, not produced; we denote that case *ConsOnly*. Matters turn out to be quite easy, and beneficial for performance, in ConsOnly: one can simply *ignore* the resources in the stubborn set analysis. Intuitively, in any plan, by definition there are sufficient resources for all actions in the plan, so we don't have to worry about the resources in analyzing permutability. In the presence of resource production, this is no longer true as the availability of sufficient resources for any one action in the plan may depend on previous actions. We show that this can be addressed by extending previous stubborn set concepts. For each of the stubborn set designs we consider, we state a suitable variant of Wehrle and Helmert's (2014) generalized strong stubborn sets, as an idealized analysis ignoring computational cost; and we state a practical syntactic approximation thereof.

Our implementation is in Fast Downward (FD) (Helmert 2006), facilitating direct comparison with the state-of-the-art stubborn sets techniques. While FD does not support continuous resource variables, one can use the widespread discrete encoding of resources via resource "units". Our definitions transfer straightforwardly to this setup.

With manually marked-up resource variables, such experiments would evaluate our contribution in planning with *explicit* resources. We go beyond this through the automatic identification of resource variables *implicit* in the input model. We introduce a concept of discrete *resource variables* in FD's finite-domain representation (FDR) context, along with a method that tests whether a given FDR state

variable qualifies for that concept. Both go beyond previous preliminary work in this direction (Seipp et al. 2016).

We run our machinery on previous ConsOnly benchmarks (Nakhost, Hoffmann, and Müller 2012), and on all IPC and Unsolvability-IPC benchmarks where resource variables can be identified (this includes all but one of the IPC domains with variables intended to encode resources). We test configurations for optimal planning, satisficing planning, and proving unsolvability. We consider strong stubborn sets, which dominate the previous literature on stubborn set pruning in planning, as well as weak stubborn sets which are less well known. It turns out that previous stubborn set methods, due to the high amount of syntactic interference on resources, hardly help in domains with resources; sometimes they are even detrimental. In contrast, our new techniques turn out to often improve over the baseline algorithms, and sometimes outperform them dramatically.

Classical Planning Background

Finite-Domain Representation (FDR)

Our work is placed in the finite-domain representation (FDR) context. An FDR *task* is a tuple $\Pi = (\mathcal{V}, \mathcal{A}, s_\mathcal{I}, s_*)$. $\mathcal{V}$ is a set of state *variables* v, each with a finite *domain* $\mathcal{D}_v$. A *fact* is a variable assignment $\langle v = d_v \rangle$. We often interpret (partial) variable assignments to $\mathcal{V}$ as sets of facts. The *initial state* $s_\mathcal{I}$ is a complete variable assignment to $\mathcal{V}$, the *goal* s_* is a partial variable assignment. Each *action* $a \in \mathcal{A}$ defines a *precondition* pre_a and an *effect* eff_a, both partial variable assignments. Each action has a non-negative *cost* $c_a \in \mathbb{R}_0^+$. A *state* s is a complete variable assignment to V. An action is *applicable* in s if $pre_a \subseteq s$. The application of a to s is denoted $s[\![a]\!]$ and results from reassigning the variables in s to the corresponding values in eff_a where defined. The application of an action sequence π is defined iteratively, and its outcome state is denoted $s[\![\pi]\!]$. We say that π is a *plan* for s if $s_* \subseteq s[\![\pi]\!]$. The plan is *optimal* if its summed-up cost is minimal among all plans; the plan is *strongly optimal* if it also contains a minimal number of 0-cost actions. A plan for $s_\mathcal{I}$ is called a plan for Π.

We denote the set of actions applicable in a state s by $\mathcal{A}|_s$. Pruning methods identify a subset $B \subseteq \mathcal{A}|_s$ to branch over. For solvable non-goal states s, B is called *safe* in s if B contains at least one action starting a strongly optimal plan for s. It is easy to see that optimal search algorithms such as A* remain optimal with safe pruning. In particular, note that for unsolvable and goal states s, *every* pruning method is safe: if there is no plan for s, or if the empty action sequence is a plan for s, then it is optimality-preserving to prune all actions at s. So, like previous works on stubborn set pruning, we will restrict our attention to solvable non-goal states.

Generalized Strong Stubborn Sets (GSSS)

Stubborn set methods proceed by identifying an action set $S \subseteq \mathcal{A}$ (the stubborn set) which "is needed to make progress to the goal, and includes all interfering actions". They guarantee that $\mathcal{A}|_s \cap S$ is safe. The most general form of stubborn sets in the planning literature are *generalized strong stubborn sets (GSSS)*, by Wehrle and Helmert (2014) (short: "WH14"). We introduce GSSS in a simplified form (leaving out the "envelope"), suited to formulate our contribution. All results transfer directly to WH14's definition of GSSS.

Definition 1 (Interference, WH14) *Let Π be an FDR task, s be a state, and $a_1, a_2 \in \mathcal{A}|_s$. a_1* disables *a_2 in s if $a_2 \notin \mathcal{A}|_{s[\![a_1]\!]}$; a_1 and a_2* conflict *in s if none disables the other but $s[\![\langle a_1, a_2 \rangle]\!] \neq s[\![\langle a_2, a_1 \rangle]\!]$; a_1 and a_2* interfere *in s if either disables the other, or they conflict in s.*

Definition 2 (Necessary Enabling Set, WH14) *Let Π be an FDR task, s be a state, and $a \notin \mathcal{A}|_s$. A* necessary enabling set *for a in s is a set $N \subseteq \mathcal{A}$ such that, for every action sequence π applicable in s that includes a, π includes some $a' \in N$ before the first occurrence of a.*

Definition 3 (GSSS, WH14) *Let Π be an FDR task, and s be a solvable non-goal state. Let $\mathcal{S}_{Opt}$ be the set of states visited by at least one strongly optimal plan for s. A set $S \subseteq \mathcal{A}$ is a GSSS for s if:*

- *(i) S contains at least one action from at least one strongly optimal plan for s.*
- *(ii) For every $a \in (S \setminus \mathcal{A}|_s)$, S contains a necessary enabling set for a.*
- *(iii) For every $a \in (S \cap \mathcal{A}|_s)$, S contains all $a' \in \mathcal{A}$ that interfere with a in any state $s \in \mathcal{S}_{Opt}$.*

The proof of safety for GSSS was formulated by WH14 in generalization of previous works. We will adapt and extend their proof to our settings later on, so it is important to understand, and to keep in mind, how that proof works.

Proof (GSSS Safety, WH14). Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a strongly optimal plan for s of which at least one action is contained in S. Such π must exist because of (i). Let k be the minimal index with $a_k \in S$. Observe that a_k is applicable in s: if this were not so, then by (ii) S would contain a necessary enabling set for a_k, which by definition would contain an action a_i with $i < k$, in contradiction to the minimality of k. Let $s_0, \ldots, s_n$ be the sequence of states traversed by π. As π is strongly optimal, all these states are contained in $\mathcal{S}_{Opt}$. Given this, observe that a_k does not interfere with a_i for any $1 \leq i < k$ in any of the states $s_0, \ldots, s_{i-1}$: if it did, then by (iii) the interfering action a_i would be contained in S, again in contradiction to the minimality of k. Hence, π can be permuted to move a_k to the front. □

Syntactic Strong Stubborn Sets (SSSS)

GSSS are not, per se, practical, as their ingredients cannot be efficiently obtained. All prior works address this through syntactic approximations. Here we capture such an approximation explicitly through what we call *syntactic SSS (SSSS)*.

A basic ingredient is syntactic *compatibility* in FDR:

Definition 4 (Compatibility) *Let Π be an FDR task, and p, q be partial variable assignments. p and q are* compatible*, denoted $p \parallel q$, if there is no $v \in \mathcal{V}$ where both $p(v)$ and $q(v)$ are defined, and $p(v) \neq q(v)$. Otherwise, p and q are* incompatible*, denoted $p \nparallel q$.*

Keep in mind that "interference" refers to *semantic* properties as per Definition 1, while "compatibility" refers to *syntactic* properties as per Definition 4. We will stick to this terminology – interference vs. compatibility – throughout.

Definition 5 (SSSS, WH14) *Let Π be an FDR task, and s be a solvable non-goal state. $S \subseteq \mathcal{A}$ is an SSSS for s if:*

(i) There exists $g \in (s_* \setminus s)$ s.t. $\{a' \in \mathcal{A} \mid g \in \mathit{eff}_{a'}\} \subseteq S$.

(ii) For every $a \in (S \setminus \mathcal{A}|_s)$, there exists $g \in (\mathit{pre}_a \setminus s)$ s.t. S contains $\{a' \in \mathcal{A} \mid g \in \mathit{eff}_{a'}\}$.

(iii) For every $a \in (S \cap \mathcal{A}|_s)$, S contains all $a' \in \mathcal{A}$ where (1) $\mathit{pre}_a \parallel \mathit{pre}_{a'}$, and (2) either $\mathit{eff}_a \nparallel \mathit{eff}_{a'}$ or $\mathit{eff}_a \nparallel \mathit{pre}_{a'}$ or $\mathit{eff}_{a'} \nparallel \mathit{pre}_a$.

This definition is not actually stated by WH'14, but it corresponds to their approximation of GSSS (modulo their "envelope strategy" which, like for GSSS, we omit here).

It is easy to see that any SSSS is a GSSS, and therefore this definition is safe: SSSS conditions (i) and (ii) obviously imply the corresponding GSSS conditions; SSSS condition (iii 2) collects all syntactic circumstances under which a and a' may interfere – a definition with much history in planning, e.g. action interference in Graphplan (Blum and Furst 1997); SSSS condition (iii 1) is sound because actions with incompatible preconditions can never be jointly applicable.

The implementation of syntactic stubborn set definitions, i. e. Definition 5 and the related definitions below (Definition 5 and Definition 9) is straightforward, see e. g. (Alkhazraji et al. 2012). The computation follows a fix-point procedure starting with actions from (i) and then iterating items (ii) and (iii) until no more new actions have to be included. An important choice point is the selection of g in (i) and (ii). A common strategy, that we also adopt here, is to stick to the same g for as long as possible (intuitively, to avoid oscillation between different subgoals).

Syntactic Weak Stubborn Sets (SWSS)

In his original work on stubborn sets, Valmari (1989) introduced not only strong stubborn sets, but also a variant called *weak stubborn sets (WSS)*. That variant has also been adopted to planning, but no empirical evaluation has been published yet.[1] As WSS sometimes yield superior results in our setting, we consider them in our experiments. The difference between strong and weak stubborn sets is orthogonal to our contribution, and we do not wish to claim any involvement in its design.

We give only the syntactic form of WSS, i. e., SWSS. The only difference to SSSS lies in condition (iii) of Definition 5. In a SWSS, for every $a \in (S \cap \mathcal{A}|_s)$, S is required to contain all $a' \in \mathcal{A}$ where either $\mathit{eff}_a \nparallel \mathit{eff}_{a'}$ or $\mathit{eff}_a \nparallel \mathit{pre}_{a'}$. That is, one considers effect-precondition interference only from a to a', not vice versa; in exchange, one has to drop the precondition-compatibility filter $\mathit{pre}_a \parallel \mathit{pre}_{a'}$. It is not difficult to show that SWSS also yield a safe pruning method.

Planning with Resources

A *resources-FDR (R-FDR) task* is a tuple $\Pi^{\mathcal{R}} = (\mathcal{V}, \mathcal{R}, \hat{R}, \mathcal{A}, s_{\mathcal{I}}, s_*)$. Here, $\mathcal{V}$, $\mathcal{A}$, $s_{\mathcal{I}}$, and s_* are exactly as in FDR. $\mathcal{R}$ is the set of resource variables. The domain of $r \in \mathcal{R}$ is the non-negative reals $\mathbb{R}_0^+$ up to $\hat{R}(r)$, the resource's maximum *capacity*, specified by $\hat{R} : \mathcal{R} \mapsto \mathbb{R}_0^+$. A state now defines a complete assignment to $\mathcal{V} \cup \mathcal{R}$. We will often refer to the value of a resource variable in a state as its *level*.

[1] Personal communication with Martin Wehrle.

Each action a is now also associated with a function $\delta_a : \mathcal{R} \mapsto \mathbb{R}$, whose value is added to $s(r)$ when a is applied. If $\delta_a(r) < 0$, we say that a *consumes* r; if $\delta_a(r) > 0$, we say that a *produces* r. The set of all consumers and producers of r is denoted $\mathit{Cons}(r)$ and $\mathit{Prod}(r)$. An action a is applicable in state s if $\mathit{pre}_a \subseteq s$ and its application would not yield any resource violations, i. e., $0 \leq s(r) + \delta_a(r) \leq \hat{R}(r)$. The remainder of the semantics is defined in the obvious manner.

We denote the fragment of R-FDR where only consumption is permitted, i. e., $\delta_a(r) \leq 0$ for all a and r, by *ConsOnly*. This will play a major role as a simpler yet practically relevant setting. The dual case, *ProdOnly*, can be easily compiled into ConsOnly.

Note that R-FDR disallows (a) resources with *infinite capacity* and (b) *refill* actions that set a resource level back to its (finite) capacity. Furthermore, one may (c) consider *over-production*, beyond a resource's capacity, as a refill instead of disabling the respective action. All of these can, in principle, be handled, but are omitted for the sake of simplicity. Basically, handling (a) is trivial but does not match the finite-state FDR setting we consider later; (b) and (c) can be handled by syntactic incompatibility notions suitable for refills. We will briefly discuss this below, referring to these cases as *InfCapacity*, *ReFill*, and *OverProd*, respectively.

Stubborn Sets with Consumed Resources

As the GSSS concepts are formulated relative to states and state transitions, not syntax, they are not limited to FDR planning. In particular, Definitions 1 – 3 of GSSS and their ingredients can in principle be applied as-is to R-FDR.

It is interesting to consider interference, Definition 1, in this context. The level of a resource r in any state s always ranges between 0 and maximum capacity, $0 \leq s(r) \leq \hat{R}(r)$. Given this, and given the commutativity of addition and subtraction, production vs. consumption of the same resource r never causes two applicable actions a_1, a_2 to interfere in s. Matters are different if both a_1 and a_2 affect r in the *same* way: if both are consumers, they disable each other in s if $s(r)$ is too low for both; if both are producers, they disable each other in s if $s(r)$ is too high for both.

Consider the following example, which shows that the current GSSS definition is unnecessarily conservative when considering resources.

Example 1 *Consider the R-FDR task $\Pi^{\mathcal{R}}$ where $\mathcal{V} = \{v_1, v_2\}$ contains two Boolean variables, $s_{\mathcal{I}}(v_i) = 0$, $s_*(v_i) = 1$, there is a single resource $\mathcal{R} = \{r\}$ with $\hat{R}(r)$=$s_{\mathcal{I}}(r)$=2, and $\mathcal{A}$=$\{a_1, a_2\}$ where pre_{a_i}=$\{\}$, eff_{a_i} = $\{v_i$=$1\}$, cost $c_{a_i} = 1$, and $\delta_{a_i}(r) = -1$. Obviously, a_1 and a_2 can be applied in any order to solve this task.*

Yet GSSS are unable to exploit the permutability of a_1 and a_2, as the set $\mathcal{S}_{Opt}$ of states on optimal plans includes states on which the two actions disable each other. Concretely, consider the (strongly optimal) plan $\pi = \langle a_1, a_2 \rangle$, and the states $s_{\mathcal{I}}, s_1, s_2$ traversed by that plan. Then a_1 and a_2 disable each other in $s_1 \in \mathcal{S}_{Opt}$, as applying any one of them reduces the level of r from 1 to 0. Given this, whenever our GSSS contains one of the two actions, it must contain the other action as well due to interference in states in $\mathcal{S}_{Opt}$.

Importantly, this issue arises only because condition (iii) of Definition 3 considers too many states and actions. Reconsider the GSSS safety proof. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a strongly optimal plan traversing states $s_0, \ldots, s_n$, and let a_k be the minimum-index action contained in our GSSS. To show that a_k cannot interfere with any preceding action a_i, the proof employs as prerequisite that $s_0, \ldots, s_{k-2} \in \mathcal{S}_{Opt}$. But then, why should $\mathcal{S}_{Opt}$ also contain the states $s_{k-1}, \ldots, s_n$? It suffices for a_i and a_k to be non-interfering in the state s_{i-1} where a_i is applied. In Example 1, with $a_k = a_2$ and $a_i = a_1$ there is no need for $\mathcal{S}_{Opt}$ to contain s_1 and s_2 because the only state relevant to the proof argument is s_0 – in which a_1 and a_2 do *not* disable each other. This gives us a slightly generalized notion of GSSS:

Definition 6 (Generalized GSSS) *Let $\Pi^{\mathcal{R}}$ be an R-FDR task, and s be a solvable non-goal state. For $a \in \mathcal{A}$, let $\mathcal{S}_{Opt}[a]$ be the set of all states t where there exists a strongly optimal plan $\pi = \langle a_1, \ldots, a_n \rangle$ for s visiting states $s_0, \ldots, s_n$, with $a = a_k$ and $t = s_{i-1}$ for some $1 \leq i < k \leq n$. Let $\mathcal{A}_{Opt}[a]$ be the set of actions containing $a_1, \ldots, a_{k-1}$ for all such π. A set $S \subseteq \mathcal{A}$ is a* generalized GSSS *for s if:*

- *(i) S contains at least one action from at least one strongly optimal plan for s.*
- *(ii) For every $a \in (S \setminus \mathcal{A}|_s)$, S contains a necessary enabling set for a.*
- *(iii) For every $a \in (S \cap \mathcal{A}|_s)$, S contains all $a' \in \mathcal{A}_{Opt}[a]$ that interfere with a in any state $t \in \mathcal{S}_{Opt}[a]$.*

This definition is identical to Definition 3 except of condition (iii) where $\mathcal{S}_{Opt}$ is replaced by $\mathcal{S}_{Opt}[a]$ and $\mathcal{A}$ by $\mathcal{A}_{Opt}[a]$. With the arguments given above, this modification is safe:

Proposition 1 (Generalized GSSS Safety) *Let $\Pi^{\mathcal{R}}$ be an R-FDR task, s be a solvable non-goal state, and S be a generalized GSSS for s. Then S is safe in s.*

Note that Definition 6 does not rely on any resource-variable specifics, and thus directly applies to FDR tasks as well. Practically speaking, however, since the syntactic approximation in Definition 5 abstracts away $\mathcal{S}_{Opt}$ anyway, this generalization is of rather limited use there.

For R-FDR tasks though that improvement turns out to be very important. Let $\Pi^{\mathcal{R}} = (\mathcal{V}, \mathcal{R}, \hat{R}, \mathcal{A}, s_{\mathcal{I}}, s_*)$ be a ConsOnly R-FDR task, s be a solvable non-goal state, and $a \in \mathcal{A}|_s$. Consider any strongly optimal plan $\pi = \langle a_1, \ldots, a_n \rangle$ for s visiting the states $s_0, \ldots, s_n$ where $a_k = a$ for some $1 \leq k \leq n$. Consider any action a_i in the prefix before a. Due to the absence of resource production, it immediately follows that every state s_j, where $0 \leq j < i$, must provide enough resources to apply both a_i and a. In other words, if a_i and a interfere in some s_j, then this cannot be due to resource consumption, but rather they must interfere at FDR level. This immediately entails that, in the syntactic approximation of interference of R-FDR actions, the resource variables can be simply ignored. Intuitively, the initial state must supply sufficient resources for all actions in the plan, so the resources are irrelevant to plan permutability.

Denote by $\Pi^{\mathcal{R}}|_{\mathcal{V}} := (\mathcal{V}, \mathcal{A}, s_{\mathcal{I}}|_{\mathcal{V}}, s_*)$ the *FDR-projection* of $\Pi^{\mathcal{R}}$. We obtain the following definition of syntactic SSS for ConsOnly R-FDR tasks:

Definition 7 (C-SSSS) *Let $\Pi^{\mathcal{R}}$ be a ConsOnly R-FDR task, and s be a solvable non-goal state. A set $S \subseteq \mathcal{A}$ is a* C-SSSS *for s if S is an SSSS for $s|_{\mathcal{V}}$ in $\Pi^{\mathcal{R}}|_{\mathcal{V}}$.*

That every C-SSSS satisfies condition (iii) of Definition 6 follows from the arguments given above. For the other two conditions, it should be easy to see the approximations in Definition 5 on the FDR projection still imply the respective conditions in Definition 6. We obtain:

Proposition 2 (C-SSSS Safety) *Let $\Pi^{\mathcal{R}}$ be a ConsOnly R-FDR task, s be a solvable non-goal state, and S be a C-SSSS for s. Then S is a generalized GSSS for s.*

A syntactic weak stubborn set variant of Definition 7, *C-SWSS*, is defined accordingly.

Stubborn Sets with Producible Resources

Turning our attention to the full scope of R-FDR, observe first that, in the presence of resource production, matters are not that simple anymore. The level of resources may decrease and increase arbitrarily along a plan. Hence, in the safety proof setup, action a_k may interfere with a preceding action a_i in the plan prefix.

Example 2 *Consider the R-FDR task $\Pi^{\mathcal{R}}$ that is like the one from Example 1, except that $\hat{R}(r) = s_{\mathcal{I}}(r) = 1$, there is an additional Boolean variable v^p with $s_{\mathcal{I}}(v^p) = 0$, and action a^p with precondition $\{v^p = 0\}$, effect $\{v^p = 1\}$, cost $c_{a^p} = 1$, and $\delta_{a^p}(r) = 1$.*

Consider now the plan $\pi = \langle a_1, a_p, a_2 \rangle$, traversing states $s_{\mathcal{I}}, s_1, s_2, s_3$. Say that $a_k = a_2$ is already contained in our generalized GSSS S under construction, and consider the prefix action $a_i = a_1$. Then a_k and a_i disable each other in the state $s_{\mathcal{I}}$ to which a_i is applied – something that can never happen in ConsOnly, as argued in the previous section.

A simple solution to this is to consider consume-consume action pairs to be syntactically incompatible, i. e., keeping the previous definition of generalized GSSS and fixing its syntactic approximation through a broader notion of incompatibility. However, this loses the ability to detect commutativity between consumers. In Example 2, we collect a_1 into S and thus are unable to prune anything in $s_{\mathcal{I}}$.

The alternative, in the example, is to catch a^p instead of a_1, i. e., collect a^p into S. The underlying reasoning is that our action $a_k = a_2$ consumes resource r. If that consumption causes interference with another consumer in the prefix, then the prefix must also contain an r-producer to re-enable a_k. We can catch that producer (in the example: a^p), making it the new minimal-index shared action between π and S.

Definition 8 (R-GSSS) *Let $\Pi^{\mathcal{R}}$ be an R-FDR task, and s be a solvable non-goal state. For $a \in \mathcal{A}$, let $\mathcal{S}_{Opt}[a]$ and $\mathcal{A}_{Opt}[a]$ be defined as in Definition 6. A set $S \subseteq \mathcal{A}$ is an* R-GSSS *for s if:*

- *(i) S contains at least one action from at least one strongly optimal plan for s.*
- *(ii) For every $a \in (S \setminus \mathcal{A}|_s)$, S contains a necessary enabling set for a.*
- *(iii) For every $a \in (S \cap \mathcal{A}|_s)$:*
  - *(a) S contains all $a' \in \mathcal{A}_{Opt}[a]$ that interfere with a in $t|_{\mathcal{V}}$ in $\Pi^{\mathcal{R}}|_{\mathcal{V}}$, for some $t \in \mathcal{S}_{Opt}[a]$.*

(b) S contains all $a' \in \mathcal{A}_{Opt}[a]$ that produce (consume) a resource consumed (produced) by a.

This definition modifies Definition 6 in the manner hinted at above. It collects in condition (iii) those a' that (a) interfere with a at the FDR level; or that (b) may be required in support of a on the prefix for attaining a suitable resource level. Observe that (iii b) diverges significantly from the common notions of stubborn sets, in that we catch supporters for an applicable action.

Theorem 1 (R-GSSS Safety) *Let $\Pi^{\mathcal{R}}$ be an R-FDR task, s be a solvable non-goal state, and S be an R-GSSS for s. Then S is safe in s.*

Proof. We adapt WH14's proof. As before, let $\pi = \langle a_1, \ldots, a_n \rangle$ be a strongly optimal plan for s of which at least one action is contained in S; such π must exist with Definition 8 (i). Let k be the minimal index with $a_k \in S$.

Applicability of a_k is shown with Definition 8 (ii) as before. Due to Definition 8 (iii b), none of the actions $a_1, \ldots, a_{k-1}$ produces (consumes) a resource consumed (produced) by a_k, or else that action would be contained in S in contradiction to the minimality of k.

In the same way as before, we can show with Definition 8 (iii a) that a_k does not interfere with any a_i where $1 \leq i < k$ in any of the states $s_0, \ldots, s_{i-1}$, at the FDR level.

Given this, π can be permuted, moving a_k to the front: interferences with respect to the plan prefix π_k up to a_k, as may result from consuming resources r consumed by a_k, cannot occur because r is never produced on π_k so r's initial level must be sufficient. Similarly for r produced by a_k. All other dependencies are left intact by the permutation. □

To make Definition 8 practical, we need to approximate its conditions, similarly as done in SSSS relative to GSSS. For (i) and (iii a), we can use the exact same techniques as for SSSS. For (iii b), we simply include all actions that produce (consume) a resource consumed (produced) by a. This leaves condition (ii), which (in difference to ConsOnly) cannot be used as-is because a necessary enabling set as per Definition 2 may need to include, e. g., resource-producing actions in order to enable resource-consuming ones. Taking the latter into account, we obtain:

Definition 9 (R-SSSS) *Let Π be an FDR task, and s be a solvable non-goal state. A set $S \subseteq \mathcal{A}$ is an R-SSSS for s if:*

(i) There exists $g \in (s_* \setminus s)$ s.t. $\{a' \in \mathcal{A} \mid g \in \mathit{eff}_{a'}\} \subseteq S$.

(ii) For every $a \in (S \setminus \mathcal{A}|_s)$:

- *(a) Either there exists $g \in (\mathit{pre}_a \setminus s)$ s.t. S contains $\{a' \in \mathcal{A} \mid g \in \mathit{eff}_{a'}\}$;*
- *(b) or there exists $r \in \mathcal{R}$ such that either $s(r) + \delta_a(r) < 0$ and S contains $\mathit{Prod}(r)$, or $s(r) + \delta_a(r) > \hat{R}(r)$ and S contains $\mathit{Cons}(r)$.*

(iii) For every $a \in (S \cap \mathcal{A}|_s)$:

- *(a) S contains all $a' \in \mathcal{A}$ where $\mathit{pre}_a \parallel \mathit{pre}_{a'}$, and $\mathit{eff}_a \not\parallel \mathit{eff}_{a'}$ or $\mathit{eff}_a \not\parallel \mathit{pre}_{a'}$ or $\mathit{eff}_{a'} \not\parallel \mathit{pre}_a$.*
- *(b) S contains all $a' \in \mathcal{A}$ that produce (consume) a resource consumed (produced) by a.*

Proposition 3 (R-SSSS Safety) *Let $\Pi^{\mathcal{R}}$ be an R-FDR task, s be a solvable non-goal state, and S be an R-SSSS for s. Then S is an R-GSSS for s.*

As before, the syntactic approximation of strong stubborn sets in Definition 9 can be replaced with one of weak stubborn sets. We refer to the outcome as an R-SWSS.

We remark that, in the ConsOnly special case, Definition 8 simplifies to Definition 6. In contrast, for R-FDR tasks in general, both definitions constitute the basis for two alternative stubborn set computations. More specifically, the difference between both definitions leads to a different treatment of resource variables in the approximation of condition (iii). While in Definition 9 (iiib), resource-supporters are considered also for applicable actions; the approximation of Definition 6 (iii) requires to take into account interference on the resource variables. To implement the latter, we extend the FDR compatibility check by additionally treating two actions syntactically incompatible if both consume (or replenish) the same resource variable. We denote the resulting approximation by I-SSSS, respectively I-SWSS.

Consider finally InfCapacity, ReFill, and OverProd. For InfCapacity, our definitions and arguments remain valid (the only difference is that Definition 9 (ii b) will include only producers). Regarding ReFill and OverProd, a refill effect is not commutative with either of consumption or production, and may disable production. In ReFill this applies to all refill effects, in OverProd it applies to those whose application exceeds capacity. A simple fix to our techniques consists in capturing all these potential interferences through the obvious syntactic incompatibilities. It remains an open question whether there are more fine-grained practical solutions.

FDR Resources and their Automatic Detection

Resources often come naturally in discrete form (e. g. the number of persons fitting into an elevator), or can be encoded as such. To make our techniques amenable to such resources, we next identify circumstances under which an FDR state variable can be interpreted as such. We then show how such FDR resource variables can be automatically identified, enabling the use of our techniques under strict IPC settings, and broadening their scope to non-obvious/non-intended FDR resources.

We need a few further notations. Assume $v \in \mathcal{V}$. We denote by $\mathcal{A}|_v \subseteq \mathcal{A}$ the set of all actions a where either $\mathit{pre}_a(v)$ or $\mathit{eff}_a(v)$ is defined. We say that $a, a' \in \mathcal{A}$ are *v-equivalent*, denoted $a \approx_v a'$, if $\mathit{pre}_a|_{\mathcal{V}\setminus\{v\}} = \mathit{pre}_{a'}|_{\mathcal{V}\setminus\{v\}}$, $\mathit{eff}_a|_{\mathcal{V}\setminus\{v\}} = \mathit{eff}_{a'}|_{\mathcal{V}\setminus\{v\}}$, and $c_a = c_{a'}$. We will interpret v-equivalent actions as ones that differ only in the resource level addressed. The definition then reads:

Definition 10 (FDR Resource Variable) *Let Π be an FDR task, and $v \in \mathcal{V}$. We say that v is an* FDR resource variable *if (i) $s_*(v)$ is not defined; (ii) for all $a \in \mathcal{A}$, $\mathit{pre}_a(v)$ is defined iff $\mathit{eff}_a(v)$ is defined; and (iii) there exist functions $\mu : \mathcal{D}_v \mapsto \mathbb{R}_0^+$ and $\delta : \mathcal{A}|_v \mapsto \mathbb{R}$ such that:*

(a) For all $a \in \mathcal{A}|_v$: $\delta(a) = \mu(\mathit{eff}_a(v)) - \mu(\mathit{pre}_a(v))$.

(b) For all $a, a' \in \mathcal{A}|_v$: if $a \approx_v a'$ then $\delta(a) = \delta(a')$.

(c) For all $d \in \mathcal{D}_v$ and $a \in \mathcal{A}|_v$ s.t. $0 \leq \mu(d) + \delta(a) \leq \mu_{\max}$, where $\mu_{\max} := \max_{d \in \mathcal{D}_v} \mu(d)$: there exists $a' \in \mathcal{A}|_v$ with $a' \approx_v a$ and $\mathit{pre}_{a'}(v) = d$.

We say that v is a consume-only *resource variable if the above holds for a function $\delta : \mathcal{A}|_v \mapsto \mathbb{R}_0^-$.*

Condition (i) simply demands that there is no goal on resources, as in R-FDR. To understand condition (ii), observe that implicit resource preconditions in R-FDR arise from non-zero consumption or production, preventing the resource from going below 0 or above its capacity. In the central condition (iii), we demand the existence of a function μ mapping the values of the FDR variable v to suitable resource levels, along with a matching function δ mapping the actions affecting v to their consumption/production. Conditions (iii a) – (iii c) ensure that this works as intended. First, with (a) we impose that v-affecting actions can be interpreted as consuming/producing the difference between their precondition/effect resource levels. Second, treating v-equivalent actions a as ones that differ only in the level of v before/after applying a – which in R-FDR would be represented as a single action – we impose (b) to ensure that this interpretation makes sense. Third, since v-equivalent actions are assumed to represent a single R-FDR action, (c) ensures that a suitable FDR action exists for all legal resource levels.

It is easy to see that FDR resource variables can be equivalently replaced with R-FDR continuous resource variables. In our FD-based implementation, however, we don't use such a translation. Instead, our stubborn set techniques handle FDR resource variables *as if they were* R-FDR resources. We apply exactly the definitions stated before, and obtain exactly the safety properties stated before.

Given Definition 10, one can identify resources in FDR by processing each $v \in \mathcal{V}$ in turn, and testing whether v satisfies the definition. But how to automatically conduct that test? Conditions (i) and (ii) are easy. Condition (iii) is more difficult, in that we need to test the existence of suitable μ and δ. It turns out that this can be done in polynomial time, via the LP encoding shown in Figure 1.

Variables:
$\mu_d \in \mathbb{R}_0^+$ for $d \in \mathcal{D}_v$
$\delta_a \in \mathbb{R}_0^-$ for $a \in \mathcal{A}|_v^C$
$\delta_a \in \mathbb{R}_0^+$ for $a \in \mathcal{A}|_v^P$
$\mu_{\max} \in \mathbb{R}_0^+$
Constraints:
1. $\mu_{pre_a(v)} - \mu_{eff_a(v)} = \delta_a$ for $a \in \mathcal{A}|_v$
2. $\delta_a = \delta_{a'}$ for $a, a' \in \mathcal{A}|_v$ where $a \approx_v a'$
3. $\mu_{\max} \geq \mu_d$ for $d \in \mathcal{D}_v$
4. $\mu_d + \delta_a < 0$ for $a \in \mathcal{A}|_v^C$ and $d \in \mathcal{D}_v$
   s.t. $\nexists\, a'$ where $a' \approx_v a$ and $pre_{a'}(v) = d$
5. $\mu_d + \delta_a > \mu_{\max}$ for $a \in \mathcal{A}|_v^P$ and $d \in \mathcal{D}_v$
   s.t. $\nexists\, a'$ where $a' \approx_v a$ and $pre_{a'}(v) = d$

Figure 1: LP testing existence of μ and δ satisfying conditions (iii a – c). $\mathcal{A}|_v^C$ ($\mathcal{A}|_v^P$) are the consumers (producers).

Constraints 1. and 2. of this LP are straightforward encodings of conditions (iii a) respectively (iii b). The role of constraint 3. is to define $\mu_{\max}$. The more interesting challenge is to encode (iii c): when there does *not* exist a suitable a', the reason can be either (A) $\mu(d) + \delta(a) < 0$ or (B) $\mu(d) + \delta(a) > \mu_{\max}$. Per se, this is a disjunction. Our key to addressing this is the observation that, as we shall explain in a moment, the actions $\mathcal{A}|_v$ affecting v can be partitioned into consumers $\mathcal{A}|_v^C$ and producers $\mathcal{A}|_v^P$, up front before creating the LP. Given this, the disjunction disappears as (A) pertains only to consumers and (B) pertains only to producers. Constraint 4. models (iii c) for the former, constraint 5. models (iii c) for the latter.

So how do we obtain the required partitioning into the $\mathcal{A}|_v^C$ and $\mathcal{A}|_v^P$ action classes? It turns out that it suffices to pick an arbitrary action $a_0 \in \mathcal{A}|_v$, and fix its class arbitrarily: the remaining class memberships follow from this single decision, for any variable v that satisfies Definition 10.

Denote the set of actions v-equivalent to an action a by $[a]_v$. Assume we fix a_0 to be a consumer. With condition (iii b), all $a_0' \in [a_0]_v$ are consumers as well. With condition (iii a), each such a_0' constrains its effect value to be smaller than its precondition value. With condition (iii c), a suitable a_0' exists for every feasible value d, so that $[a_0]_v$ induces a total ordering over these d. The same happens if we fix a_0 to be a producer instead, we just get the inverse order. By the same arguments, *any* set $[a]_v$ of v-equivalent actions induces an ordering over the touched values d. Now, since every $[a]_v$ must tackle all feasible values d, it is easy to see that, for any $[a]_v$ and $[a']_v$, the touched value subsets must overlap. So if we fix the ordering in $[a_0]_v$, then transitively we fix it in all $[a]_v$, and therewith we fix the partitioning into consumers and producers.

Experiments

We implemented our techniques in Fast Downward (FD) (Helmert 2006). The FDR resource variable detection uses CPLEX for the LP tests. All experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

We consider optimal planning, satisficing planning, and proving unsolvability. Our baseline solver configurations for these categories are A* with LM-cut (Helmert and Domshlak 2009); greedy best-first search with h^{FF} (Hoffmann and Nebel 2001) and a dual queue for preferred operators; and A* with $h^{\max}$ (Bonet and Geffner 2001). We evaluate the baseline itself, its enhancement with the previous state of the art in strong and weak stubborn set pruning, and its enhancement with our new techniques. For ease of reference, Table 1 gives an overview of our stubborn set acronyms.

SSSS	previous state of the art strong stubborn set
C-SSSS	SSSS ignoring resources in ConsOnly
R-SSSS	SSSS catching resource-supporters in R-FDR
I-SSSS	SSSS catching resource-interferers in R-FDR

Table 1: Strong stubborn set acronyms. Acronyms for weak stubborn set variants are defined similarly.

We consider all STRIPS benchmarks of all IPCs; the Unsolvability-IPC (UIPC) 2016 benchmarks; the RCP benchmarks by Nakhost et al. (2012); and unsolvable variants of the latter (Steinmetz and Hoffmann 2016).

We next report about the performance of resource variable detection, then we discuss our results in optimal planning, satisficing planning, and proving unsolvability. Throughout, we do not include data for SWSS, as their performance is dominated by, and almost identical to, that of SSSS.

Domain	#	time limit 10 sec. t	time limit 10 sec. ≥ 1	time limit 10 sec. C	time limit 10 sec. P	no time limit t	no time limit ≥ 1	no time limit C	no time limit P
IPC									
Childsnack	40	1.26	100	26.5	0.0	1.26	100	26.5	0.0
Elevators	100	0.48	100	0.0	4.5	1.51	100	0.0	4.5
Freecell	80	2.31	100	4.0	2.0	1.00	100	4.0	2.0
Mprime	35	4.22	91	0.0	4.3	7.50	100	0.0	5.6
Mystery	30	3.49	93	10.4	4.3	5.21	93	10.4	5.5
NoMystery	40	0.41	100	1.0	0.0	0.41	100	1.0	0.0
Openstacks	140	1.40	100	0.0	1.0	1.42	100	0.0	1.0
ParcPrinter	70	0.26	100	8.0	0.0	0.27	100	8.0	0.0
Pathways	30	0.23	100	5.2	0.0	0.23	100	5.2	0.0
Pipesworld-Tankage	50	4.44	100	0.0	40.3	5.12	100	0.0	41.6
Pipesworld-NoTankage	50	0.32	28	0.4	0.0	0.34	28	0.4	0.0
Rovers	40	1.12	100	19.0	6.4	1.12	100	19.0	6.4
Thoughtful	20	1.45	100	24.4	1.0	1.47	100	24.4	1.0
TPP	30	3.91	100	24.2	73.6	4.50	100	26.4	81.6
Transport	140	1.66	100	0.0	3.1	1.66	100	0.0	3.1
Woodworking	100	0.31	61	3.2	0.0	0.33	61	3.2	0.0
UIPC									
BagBarman	20	1.35	20	0.0	0.2	1.36	20	0.0	0.2
BagTransport	29	5.34	100	0.0	56.4	103.64	100	0.0	893.4
Bottleneck	25	0.89	100	33.2	0.0	0.91	100	33.2	0.0
CaveDiving	25	0.58	100	0.0	6.0	0.63	100	0.0	6.0
ChessBoard	23	6.11	100	30.0	205.0	7.67	100	30.0	266.0
Diagnosis	20	1.26	55	41.1	2.8	1.17	55	41.0	2.8
NoMystery	24	0.58	100	1.0	0.0	4.89	100	1.0	0.0
PegSolRow5	15	3.35	93	0.0	120.9	3.12	93	0.0	120.9
Rovers	20	0.45	100	4.4	1.0	0.46	100	4.4	1.0
TPP	30	0.45	100	1.0	0.0	0.41	100	1.0	0.0
RCP: Nakhost et al. (2012), Steinmetz and Hoffmann (2016)									
NoMystery	360	0.29	100	1.7	0.0	0.30	100	1.7	0.0
Rovers	360	0.35	100	2.0	1.7	0.36	100	2.0	1.7
TPP	55	2.50	100	1.0	0.0	2.54	100	1.0	0.0

Table 2: Resource detection results (domains with at least one detected resource). "t": average runtime in seconds; "≥ 1": % instances where at least one resource variable was detected; "C": average number of consumed-only such variables; "P": average number of producible such variables.

Resource Variable Detection

As resource detection must be a run as a preprocess before our techniques can even be attempted, we experimented with an overall time limit of 10s. Once the time limit is reached, the algorithm stops, reporting only the resources identified up to this point. Consider Table 2. Clearly, resource detection is usually fast and the time limit hardly impedes its performance. For consumed-only resource variables, indeed the only difference is in IPC TPP (the small difference in Diagnosis is due to non-determinism in FD's pre-processor). For producible resource variables, the difference is somewhat larger, with less resource variables identified in six domains, most notably in BagTransport.

Observe that Table 2 contains quite a few domains with non-obvious/non-intended resources. Some examples are: In Childsnack cooking ingredients are considered resources that are consumed when preparing meals. In Bottleneck, the consumed resources are grid cells which become impassable once visited. In ChessBoard and PegSolRow5, cells are resources that can be either freed (produced) or occupied (consumed) by pegs. There are also several cases where one-time only consumable resources are detected. The corresponding FDR variables are used for example in Pipesworld to encode in the initial state pipes holding incompatible fluids. In Thoughtful, such variables encode in the initial state the position of a card on top of another card, not complying with valid orders. The only domain with intentional resources – to our knowledge – missing from Table 2 is Zenotravel, where "fly" and "zoom" actions have different consumption yet are *fuel*-equivalent, i. e., violate Definition 10 (iiib).

Domain	#	Coverage B	Coverage S	Coverage RS	Coverage RW	Eval S	Eval RS	Eval RW	Runtime S	Runtime RS	Runtime RW
IPC											
Childs*	20	0	0	0	0						
Elevat	50	40	40	40	40	**1.6**	1.2	1.5	**1.4**	1.0	1.1
Freec	80	**15**	14	**15**	**15**	1.0	1.0	1.0	0.4	**0.9**	0.8
Mprime	29	21	21	21	21	1.0	1.0	1.0	**1.1**	0.8	0.6
Mystery	28	19	**20**	**20**	**20**	**1.1**	1.0	1.0	**0.8**	0.4	0.4
NoMyst*	20	14	14	14	14	1.0	1.0	**1.2**	0.8	1.1	**1.2**
Openst	70	40	36	**45**	**45**	1.3	**2.5**	**2.5**	0.5	1.3	**1.4**
ParcPr*	50	31	**50**	**50**	31	**13.9**	**13.9**	1.0	**1.0**	**1.0**	0.4
Pathway*	30	5	5	5	5	2.1	28.7	**31.9**	1.2	**4.8**	1.7
PipesT	50	**12**	11	**12**	**12**	1.0	1.0	1.0	**1.0**	0.7	0.6
PipesNoT*	14	1	1	1	1	1.0	1.0	1.0	0.9	0.9	**1.0**
Rovers	40	7	**9**	**9**	**9**	2.0	2.2	**2.3**	1.5	1.5	**2.1**
TPP	30	6	6	6	6	1.0	1.0	1.0	**1.0**	0.3	0.7
Transp	70	23	23	23	23	1.0	1.0	1.0	**1.0**	0.7	0.6
Woodwor*	20	11	**19**	**19**	**19**	14.9	**17.0**	**17.0**	**4.9**	1.9	2.5
Zenot	20	13	13	13	13	1.1	1.1	1.1	0.7	**1.0**	0.5
$\sum$	621	258	282	**293**	274						
RCP (Nakhost, Hoffmann, and Müller 2012)											
NoMyst*	210	66	60	66	**68**	1.0	1.0	**1.2**	0.7	1.0	**1.1**
Rovers	210	0	0	0	0						
TPP*	30	0	0	**26**	**26**						
$\sum$	450	66	60	92	**94**						

Table 3: Optimal planning. "*" marks ConsOnly. "Eval": reduction factor in number of evaluations, relative to baseline. "Runtime": reduction factor in total runtime. "B": Baseline, A* with LM-cut; "S": SSSS; "RS": R-SSSS (C-SSSS if ConsOnly); "RW": R-SWSS (C-SWSS if ConsOnly). Eval and Runtime show geometric averages over commonly solved instances. Best results shown in **bold**.

Henceforth, we assume that resource variable detection is run with the 10s time limit. We consider only those benchmarks where at least one resource variable is found.[2] We include this additional preprocess in our techniques only, and we count it as part of overall runtime.

Optimal Planning

Consider Table 3. Our new techniques, R-SSSS and R-SWSS, clearly perform best overall. Both are useful in Openstacks, where SSSS is detrimental. In RCP NoMystery, the same applies to R-SWSS. Both techniques manage to reduce state space size significantly in Pathways, an advantage which disappears in coverage due to the runtime overhead. In IPC Rovers and Woodworking, both techniques inherit the advantages of SSSS; in ParcPrinter that is so for R-SSSS only. The largest advance occurs in RCP TPP, where the domain of the single (consume-only) resource variable is par-

[2]Where no FDR resource variable is detected, we could use an arbitrary planning technique (e.g. previous stubborn set methods). We choose to not report about such cases at all, as that would only obscure those results where something interesting happens.

Domain	#	Coverage B	S	RS	RW	Eval S	RS	RW	Runtime S	RS	RW
IPC											
Childs*	20	9	9	9	9	1.3	1.1	**1.7**	1.2	1.0	**1.6**
Elevat	50	11	**12**	11	11	**1.7**	1.0	1.3	**1.6**	0.8	0.9
Freec	80	**80**	78	**80**	**80**	1.0	1.0	1.0	0.4	**0.8**	**0.8**
Mprime	29	29	29	29	29	**1.1**	1.0	1.0	**0.7**	0.5	0.5
Mystery	28	20	**23**	21	21	1.1	1.1	1.1	**0.6**	0.4	0.4
NoMyst*	20	**13**	11	**13**	**13**	1.0	1.0	**1.2**	0.4	0.8	**1.1**
Openst	70	6	6	**7**	**7**	1.2	**1.9**	**1.9**	0.9	**2.5**	1.6
ParcPr*	50	27	**50**	**50**	27	**155.8**	**155.8**	0.9	**9.7**	3.3	1.0
Pathway*	30	23	24	**26**	**26**	1.3	2.6	**3.3**	**0.7**	0.5	0.4
PipesT	50	32	32	**33**	**33**	1.0	1.0	1.0	**0.7**	0.5	0.5
PipesNoT*	14	10	10	10	10	1.0	1.0	1.0	**1.0**	0.7	0.7
Rovers	40	**40**	39	38	38	1.4	**1.7**	**1.7**	**0.8**	0.7	**0.8**
Thought	20	10	10	10	10	1.0	**1.2**	**1.2**	0.8	**1.1**	0.8
TPP	30	30	30	30	30	1.0	1.0	1.0	**0.7**	0.5	0.4
Transp	70	23	23	23	23	1.0	1.0	1.0	**0.7**	0.6	**0.7**
Woodwor*	41	41	41	41	41	3.2	**3.3**	**3.3**	**1.1**	1.0	1.0
Zenot	20	20	20	20	20	1.0	1.0	1.0	0.6	**1.0**	0.8
$\sum$	719	473	499	**501**	478						
RCP: Nakhost et al. (2012)											
NoMyst*	210	125	102	**126**	**126**	1.1	1.1	**1.4**	0.2	0.9	**1.0**
Rovers	210	18	9	**22**	**22**	1.7	**3.1**	**3.1**	0.4	**1.7**	1.3
TPP*	30	10	10	**26**	**26**	1.0	**2.2**	**2.2**	0.4	**1.1**	**1.1**
$\sum$	450	153	121	**174**	**174**						

Table 4: Satisficing planning. Abbreviations as in Table 3; baseline is greedy best-first search with h^{FF} and a dual queue for preferred operators. Best results shown in **bold**.

ticularly large, strongly reducing the number of action pairs considered to be incompatible. R-SSSS never has worse coverage than either of the baseline and SSSS; it also dominates R-SWSS except in RCP NoMystery.

Regarding I-SSSS and I-SWSS, these are not included in Table 3, and neither in the tables below, as their performance is almost identical to that of R-SSSS and R-SWSS respectively. The disadvantage of I-SSSS and I-SWSS in treating common consumption of a resource as syntactic incompatibility seems to be counterbalanced by not having to include producers for applicable consumers.

Satisficing Planning

Stubborn set pruning has, in classical planning, previously been evaluated for optimal planning only. Yet it can be useful for satisficing planning too, in cases where known search heuristics are weak. This can be expected, in particular, in resource-constrained planning with delete relaxation heuristics, where forward search spaces are full of dead-end states not detected by the heuristic function. Our results show that, indeed, this is the case. Table 4 shows the data.

Like in optimal planning, R-SSSS is dominant overall, with best overall coverage in both benchmark sets. Further, our new techniques exhibit advantages in Openstacks and Pathways, and inherit the strengths of SSSS in ParcPrinter (R-SSSS only) and Woodworking. In IPC Rovers, that is no longer true; no stubborn set technique helps here anymore.

In Elevators and Mystery, the advantage of SSSS, which is present yet small for optimal planning, becomes more pronounced. In IPC NoMystery, on the other hand, SSSS is now detrimental while our new techniques are not.

Domain	#	Coverage B	S	RS	RW	Eval S	RS	RW	Runtime S	RS	RW
UIPC 2016											
BagBar	4	**4**	0	**4**	**4**						
BagTran	29	6	6	6	6	1.0	1.0	1.0	0.6	0.6	**0.7**
Bottlen*	25	21	21	**25**	**25**	1.0	**204.3**	**204.3**	1.2	1.5	**1.7**
CavDiv*	25	7	7	7	7	1.0	1.4	**1.8**	0.6	0.9	**1.5**
ChessBo	23	5	**6**	5	5	**2.3**	1.1	1.1	**0.9**	0.4	0.3
Diagno*	20	7	7	**8**	**8**	1.7	**2.3**	2.2	**1.0**	0.8	0.8
NoMyst*	24	2	2	2	2	1.0	1.0	**2.4**	0.9	**2.7**	1.6
PegRow5	14	**4**	**4**	3	3	1.0	1.0	1.0	**6.0**	0.7	0.4
Rovers	20	7	7	7	7	1.3	1.3	1.3	0.4	**1.0**	0.9
TPP*	30	16	14	**21**	**21**	1.0	36.9	**42.6**	0.5	1.6	**2.2**
$\sum$	214	79	74	**88**	**88**						
Unsolvable RCP: Steinmetz and Hoffmann (2016)											
NoMyst*	150	53	42	**55**	**55**	**1.4**	1.2	**1.4**	0.3	**0.8**	**0.8**
Rovers	150	7	7	**8**	7	1.6	1.6	1.6	0.5	**1.5**	1.0
TPP*	25	5	2	**18**	**18**	1.0	**372.5**	**372.5**	0.6	20.3	**21.0**
$\sum$	325	65	51	**81**	80						

Table 5: Proving unsolvability. Abbreviations as in Table 3; baseline is A* with $h^{\max}$. Best results shown in **bold**.

The most consistent gains for our new methods occur on the RCP benchmarks. In NoMystery, our methods yield a small advantage, and in Rovers and TPP the advantage is huge. SSSS, in comparison, is mostly detrimental here.

Using R-SSSS is not as risk-free as in optimal planning, but still the cases where we lose coverage are rare. Relative to the baseline, this happens only in IPC Rovers; relative to SSSS, it happens in Elevators, Mystery, and IPC Rovers.

Proving Unsolvability

We finally consider proving unsolvability. Stubborn set techniques have been employed for that purpose by some systems in the 2016 Unsolvability-IPC (UIPC) competition. But otherwise, this purpose has not been evaluated in the literature on stubborn sets in planning. Consider Table 5.

Both R-SSSS and R-SWSS clearly dominate overall coverage. In UIPC Bottleneck and TPP, and Diagnosis and NoMystery to a smaller degree, our techniques yield an advantage whereas SSSS yields none or is detrimental. In ChessBoard, SSSS has the advantage. In RCP, we obtain a huge advantage in TPP, and a small one in NoMystery. Both R-SSSS and R-SWSS lag behind in coverage only rarely (in BagBarman vs. the baseline and in ChessBoard vs. SSSS).

Conclusion

We have shown that continuous resources, and their discrete FDR counterparts, have commutativity properties that can be exploited in stubborn set pruning. Empirically, this seldom hurts, and sometimes yields dramatic benefits.

An obvious piece of further work are experiments on PDDL benchmarks with continuous resources (IPC'02), though the implementation and comparison basis there is far less advanced than for PDDL level 1. Beyond the immediate horizon, our work shows the benefits of tackling resources explicitly in stubborn set pruning, which suggests the investigation of other search reduction methods specifically for planning with resources. This may be interesting, e. g., for dominance pruning and symmetry reduction.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, "Critically Constrained Planning via Partial Delete Relaxation", and was partially supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA, grant no. 16KIS0656).

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In Raedt, L. D., ed., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 891–892. Montpellier, France: IOS Press.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1–2):279–298.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2013. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research* 46:343–412.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 109–120. Springer-Verlag.

Edelkamp, S.; Leue, S.; and Lluch-Lafuente, A. 2004. Partial-order reduction and trail improvement in directed model checking. *International Journal on Software Tools for Technology Transfer* 6(4):277–301.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In Hammond, K., ed., *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS'94)*, 61–67. Chicago, IL: AAAI Press, Menlo Park.

Godefroid, P., and Wolper, P. 1991. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*, 332–342.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 121–132. Springer-Verlag.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Koehler, J. 1998. Planning under resource constraints. In Prade, H., ed., *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98)*, 489–493. Brighton, UK: Wiley.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A Monte Carlo random walk approach. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 181–189. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Wehrle, M. 2016. Fast downward aidos. In *UIPC 2016 planner abstracts*, 28–38.

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1-2):73–134.

Steinmetz, M., and Hoffmann, J. 2016. Towards clause-learning state space search: Learning to recognize dead-ends. In Schuurmans, D., and Wellman, M., eds., *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, 491–515.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Do, M.; Fern, A.; and Ruml, W., eds., *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Rome, Italy: AAAI Press.

Winterer, D.; Alkhazraji, Y.; Katz, M.; and Wehrle, M. 2017. Stubborn sets for fully observable nondeterministic planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.