Decoupled Strong Stubborn Sets

Daniel Gnad
Saarland University
Saarbrücken, Germany
gnad@cs.uni-saarland.de

Martin Wehrle
University of Basel
Basel, Switzerland
martin.wehrle@unibas.ch

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Abstract

Recent work has introduced fork-decoupled search, addressing classical planning problems where a single center component provides preconditions for several leaf components. Given a fixed center path $\pi^C$, the leaf moves compliant with $\pi^C$ can then be scheduled independently for each leaf. Fork-decoupled search thus searches over center paths only, maintaining the compliant paths for each leaf separately. This can yield dramatic benefits. It is empirically complementary to partial order reduction via strong stubborn sets, in that each method yields its strongest reductions in different benchmarks. Here we show that the two methods can be combined, in the form of strong stubborn sets for fork-decoupled search. This can yield exponential advantages relative to both methods. Empirically, the combination reliably inherits the best of its components, and often outperforms both.

1 Introduction

In classical AI planning, the task is to find a sequence of actions leading from a given initial state to a state that satisfies a given goal condition, in a large deterministic transition system (the task’s state space). Gnad and Hoffmann [2015a] (henceforth: GH) have recently introduced a new approach, fork-decoupled search, to decompose the state space. The approach relates to factored planning (e.g. [Amir and Engelhardt, 2003; Kelareva et al., 2007; Fabre et al., 2010; Brafman and Domshlak, 2013]), where the factors are disjoint subsets of state variables. Fork-decoupled search assumes that the factors induce a fork structure: a single center factor provides preconditions for several leaf factors, and no other cross-factor interactions exist. As GH show, such a fork factoring, if one exists, can be easily identified in a pre-process to planning, based on the task’s causal graph (e.g. [Knoblock, 1994; Jonsson and Bäckström, 1995; Brafman and Domshlak, 2003; Helmer, 2006]).

In a fork factoring, the leaves are “conditionally independent”, in the sense that, given a fixed center path $\pi^C$, the compliant leaf moves – those leaf moves enabled by the preconditions supplied along $\pi^C$ – can be scheduled independently for each leaf. This can be exploited by searching only over center paths, and maintaining the possible compliant paths separately for each leaf, thus avoiding the enumeration of state combinations across leaves. GH show how to employ standard heuristic search planning algorithms, preserving optimality guarantees. They obtain dramatic benefits in several International Planning Competition (IPC) benchmarks.

Fork-decoupling can be thought of as a reformulation of the search space. Can known search reduction methods be applied on the reformulated search space as well? We herein answer this in the affirmative for a prominent reduction method, namely state-of-the-art partial order reduction via strong stubborn sets (SSS) [Valmari, 1989; Alkhazraji et al., 2012; Wehrle and Helmer, 2012; 2014]. This method prunes applicable actions on states $s$ during (standard, non-decoupled) search, namely those not contained in an SSS for $s$. The SSS is guaranteed to contain at least one action starting an optimal plan for $s$, so optimality is preserved.

Fork-decoupled search and SSS yield their respective best reductions in different IPC domains. We show that the two methods are indeed exponentially separated, i.e., that there are cases where one yields exponentially stronger reductions than the other. We show how to combine them in the form of decoupled strong stubborn sets (DSSS), for fork-decoupled search. We show that this combination is exponentially separated from both its components. There are cases – not complex artificial examples, but simple variants of the Logistics benchmark – where DSSS yield exponentially stronger reductions than both fork-decoupling and SSS. Empirically, DSSS reliably inherit the strengths of each component, and sometimes outperform both. In some cases, DSSS even is “more than the sum of its components”, yielding a stronger reduction in fork-decoupled search than SSS do in standard search.

For space reasons, we give proof sketches only. The full proofs are available in an online TR [Gnad et al., 2016].

2 Background

We employ a finite-domain state variable formalization of planning (e.g. [Bäckström and Nebel, 1995; Helmer, 2006]). A finite-domain representation planning task, short FDR task, is a tuple $Π = \langle V, A, s_0, s_*$ $\rangle$. $V$ is a set of state variables, each $v \in V$ associated with a finite domain $D(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to $V$ is a state. $s_0$ is the initial state, and the goal $s_*$ is a partial assignment to $V$. $A$
is a finite set of actions. Each action \( a \in A \) is a triple \(<\text{pre}(a), \text{eff}(a), \text{cost}(a)>\) where the precondition \( \text{pre}(a) \) and effect \( \text{eff}(a) \) are partial assignments to \( V \), with \( \text{eff}(a) \neq \emptyset \); \( \text{cost}(a) \in \mathbb{R}^{+} \) is a non-negative cost.

Given a partial assignment \( p \), by \( \text{vars}(p) \subseteq V \) we denote the subset of state variables on which \( p \) is defined. For \( V \subseteq \text{vars}(p) \), by \( p[V] \) we denote the assignment to \( V \) made by \( p \).

Action \( a \) is applicable in state \( s \) if \( s \models \text{pre}(a) \), i.e., \( \text{pre}(a) \subseteq s \). Applying \( a \) in \( s \) changes the value of \( v \in \text{vars}(\text{eff}(a)) \) to \( \text{eff}(a)[v] \), and leaves \( s \) unchanged elsewhere. A plan for \( II \) is an action sequence \( \pi \) applicable in \( s_0 \) and ending in a state \( s \) such that \( s \models s_\pi \). The plan is optimal if its summed-up cost, denoted \( \text{cost}(\pi) \), is minimal among all plans for \( II \).

We next give a summary of fork-decoupled search. We will often write “decoupled” instead of “fork-decoupled”.

A factoring \( F \) is a partition of \( V \). \( F \) is a fork factoring if \( |F| \geq 2 \) and there exists \( F^C \in F \) s.t. the arcs in \( F^C \)’s interaction graph are exactly \( \{(F^C, F^L) | F^L \in F \setminus \{F^C\}) \}. Here, the interaction graph is the quotient of the task's causal graph over \( F \), i.e., it contains an arc \((F, F')\) if there exists \( a \in A \) s.t. \( F \cap \{\text{vars}(\text{pre}(a)) \cup \text{vars}(\text{eff}(a))\} \neq \emptyset \) and \( F' \cap \text{vars}(\text{eff}(a)) \neq \emptyset \). We refer to \( F^C \) as the center of \( F \), and to all other factors \( F^L \in F \setminus F^C \) as its leaves.

As a running example, consider a Logistics-style planning task with 1 truck variable \( t \), N packages variables \( p_i \), and two locations \( A \) and \( B \). The truck and all packages are initially at \( A \), and the goal is for the packages to be at \( B \). The actions (unit costs) are drive, load, and unload, with the usual preconditions and effects (e.g. load\((t, p, A)\) requires both \( t \) and \( p \) to be at \( A \), and moves \( p \) into the truck). Setting \( \{t\} \) as the center and each \( \{p_i\} \) as a leaf, we obtain a fork factoring.

Not every task \( II \) has a fork factoring. We assume GH’s approach of analyzing \( II \)’s causal graph in a pre-process, identifying a fork factoring if one exists, else abstaining from solving \( II \). In what follows, assume a fork factoring \( F \).

Given the structure of the interaction graph, every action affects (touches in its effect) either only \( F^C \), or only one leaf \( F^L \). We refer to the former kind as center actions, and to the latter kind as leaf actions. Observe that center actions do not have any preconditions on leaves. Furthermore, if leaf action \( a \) affects leaf \( F^L \), then it can be preconditioned only on \( F^C \) and \( F^L \), i.e., \( \text{vars}(\text{pre}(a)) \subseteq F^C \cup F^L \).

Due to these action behaviors, a fork factoring encapsulates a particular form of “conditional independency” between leaves. Assume a center path \( \pi^C \), i.e., a sequence of center actions applicable to \( s_0 \). A leaf path is a sequence of leaf actions applicable to \( s_0 \) when ignoring preconditions on \( F^C \). A leaf path \( \pi^L \) complies with \( \pi^C \) if it uses only the center preconditions supplied along \( \pi^C \), i.e., if \( \pi^L \) can be scheduled alongside \( \pi^C \) so that the combined action sequence is applicable in \( s_0 \). Intuitively, fixing \( \pi^C \), the compliant leaf paths are the possible leaf moves given \( \pi^C \). Observe that these possible moves are independent across leaf factors \( F^L \), i.e., for each \( F^L \) we can choose a compliant \( \pi^L \) independently from that choice for any other leaf factor. Hence we can search over center paths \( \pi^C \) only, maintaining all possible compliant paths separately for each leaf. We commit to the actual choices of compliant leaf paths only when the goal is reached.

Concretely, a decoupled state \( s^D \) is given by a center path \( \pi^C(s^D) \). It is associated with its center state \( c t(s^D) \), simply the outcome of applying \( \pi^C(s^D) \) to \( s_0[F^C] \); and with its pricing function \( \text{prices}(s^D) \). The latter maps each leaf state \( s^L \), i.e., each value assignment to some leaf \( F^L \), to its price, defined as the cost of a cheapest leaf path that complies with \( \pi^C(s^D) \) and ends in \( s^L \) (or \( \infty \) if no such path exists).

Pricing functions can be maintained in time low-order polynomial in the size of the individual \( F^L \) state spaces; we omit the details for space reasons. Note the word “price”: \( \text{prices}(s^D)[s^L] \) is not a cost we have already paid; rather, it is the cost we will have to pay in case we commit to \( s^L \) in \( s^D \) later on.

The initial decoupled state \( s^D_0 \) results from the empty center path \( \pi^C(s^D_0) = \emptyset \). We denote by \( \text{Reached}(s^D) \) the set of leaf states \( s^L \) reachable in \( s^D \), i.e., where \( \text{prices}(s^D)[s^L] < \infty \). A goal decoupled state \( s^D_g \) is one with a goal center state \( \pi^C(s^D_g) \models s^*_A[F^C] \) and where, for every leaf factor \( F^L \in F^L \), there exists a reachable goal leaf state \( s^L \), i.e., \( s^L \in \text{Reached}(s^D_g) \) such that \( s^L \models \pi^C(s^L) \). The actions applicable in \( s^D \) are those center actions whose precondition is satisfied in \( ct(s^D) \) (recall here that we do not branch over leaf actions). Applying \( a \) to \( s^D \) results in \( t^D \) where \( \pi^C(t^D) := \pi^C(s^D) \circ (a) \), and \( ct(t^D) \) as well as \( \text{prices}(t^D) \) arise from \( \pi^C(t^D) \) as defined above.

In our example, \( ct(s^D_0) = \{(t, A)\} \), and for each \( p_i \) the price of \( (p_i, A) \) is 0, that of \( (p_i, t) \) is 1, and that of \( (p_i, B) \) is \( \infty \). Observe here that the prices represent possible package moves given the initial center state, rather than moves we have already committed to. The only action applicable to \( s^D_0 \) in the decoupled search is the center action \( \text{drive}(t, A, B) \), leading to the goal decoupled state \( s^D_0 \) where \( ct(s^D_0) = \{(t, B)\} \) and the prices are as before except that \( (p_i, B) \) now has price 2, i.e., the package goals are reachable.

Once a goal decoupled state \( s^D_g \) is reached, a plan \( \pi \) for the input task \( II \) can be constructed by augmenting the center path \( \pi^C(s^D_g) \) with compliant leaf paths ending in goal leaf states \( s^L \) (i.e., we now select such leaf paths, and commit to them). In our example, for each \( p_i \) we may select the compliant leaf path \( \text{load}(t, p, A), \text{unload}(t, p, B) \).

Selecting, for the plan \( \pi \), the cheapest compliant paths ending in the goal leaf states \( s^L \), by construction we have \( \text{cost}(\pi) = \text{cost}(\pi^C(s^D_g)) + \sum_{F^L \in F^L} \text{prices}(s^D_g)[s^L] \). If we select \( s^L \) with minimal \( \text{prices}(s^D_g)[s^L] \), such \( \pi \) is optimal among the plans for \( II \) whose center action subsequence is \( \pi^C(s^D_g) \). Given this, we refer to the cost of such \( \pi \) as the local cost of \( s^L \), denoted \( \text{LocalCost}(s^L) \). We set \( \text{LocalCost}(s^D_g) := \infty \) for non-goal decoupled states \( s^D \).

\( \text{LocalCost}(s^D_g) \) is optimal for \( s^D_g \) (locally optimal) but not necessarily optimal for \( II \) (globally optimal). Indeed, it can happen that, from \( s^D_g \), a better plan can be obtained from a descendant of \( s^D_g \). This is because, with additional center actions, cheaper leaf paths may become available. For example, say in \( s^D_g \) the leaf goal \( \text{have-car} \) has price 1000 via the applicable leaf action \( \text{buy-car} \). But if we apply a center action \( \text{get-manager-job} \), then the leaf action \( \text{get-company-car} \) becomes applicable, reducing the leaf goal price to 0.

In contrast to the standard setting, to guarantee optimality one must therefore continue the search on goal decoupled states (GH show that standard search algorithms are easy to adapt to this situation). The purpose of such search, trying to
decrease leaf prices, differs from that of non-goal decoupled states, trying to reach the goal in the first place. Our design of strong stubborn sets for decoupled search distinguishes between the two cases.

3 SSS for Non-Goal Decoupled States

We show that, for non-goal decoupled states, the definition of strong stubborn sets (SSS) for planning [Alkhzraji et al., 2012] can be extended to decoupled search by suitable extensions of its basic components.

A SSS for a given state $s$ is a set $T_s \subseteq A$ constructed so that, for every plan $\pi$ for $s$, at least one permutation of $\pi$ starts with an action $a \in T_s$. Hence SSS are fundamentally based on the concept of “plans for a given state”. That concept is trivial for classical state spaces. But in decoupled state spaces the structure of “states” $s^F$ is more complex. GH did not require, so did not introduce, such a concept. For our purposes, the following notions will suffice.

A path $\pi^F$ in the decoupled state space is a decoupled plan for $s^F$ if it leads from $s^F$ to a goal decoupled state. We say that $s^F$ is solvable if at least one such $\pi^F$ exists. We denote the center-action sequence underlying $\pi^F$ by $\pi^C(\pi^F)$. The completion plan given $\pi^F$, denoted $\text{ComPlan}(\pi^F)$, consists of $\pi^C(\pi^F)$ together with cheapest goal leaf paths $\pi^L$ compliant with $\pi^F(s^F) \circ \pi^C(\pi^F)$, ending in cheapest goal leaf states. In other words, $\text{ComPlan}(\pi^F)$ collects the postfix path for the center, and the complete path for each leaf. Observe that $\text{ComPlan}(\pi^F)$ is not uniquely defined, as there may be multiple suitable $\pi^L$. For our purposes, this does not matter and we assume any suitable choice of $\pi^L$. We say that $\pi^F$ is optimal if $\text{cost}(\text{ComPlan}(\pi^F))$ is minimal among all decoupled plans for $s^F$. In our running example, assume a third location C and the road map $A \rightarrow B \rightarrow C$. Say we apply drive$(t, A, B)$ to $s_0^F$ to obtain $s^F$. Then $\pi^C := \{\text{drive}(t, B, C)\}$ yields a decoupled plan $\pi^F$ for $s^F$, and $\text{ComPlan}(\pi^F)$ consists of all load$(t, p_i, A)$ actions, then $\pi^C$, and then unload$(t, p_i, C)$ actions.

Clearly, to preserve optimality, it suffices for $T_s$ to contain at least one center action starting an optimal decoupled plan for $s^F$. Towards identifying sets $T_s$ qualifying for this, we will need to focus exclusively on the part of the completion plan “behind” $s^F$. We denote this by $\text{PostPlan}(\pi^F)$, the postfix plan. The center action subsequence in $\text{PostPlan}(\pi^F)$ is $\pi^C(\pi^F)$. For any leaf factor $F^L$, say $\pi^L = \langle a^L_1, \ldots, a^L_\alpha \rangle$ is the goal leaf path for $F^L$ in $\text{ComPlan}(\pi^F)$, traversing leaf states $\langle s^L_0^F, \ldots, s^L_\mu^F \rangle$. Then the leaf action subsequence $a^L_i$ in $\text{PostPlan}(\pi^F)$ is defined as $\langle a^L_i, \ldots, a^L_\alpha \rangle$, where $i$ is the highest index for which $s^L_i \in \text{ReachedL}(s^F)$. In other words, we consider the postfix of $\pi^F$ not contained in $\text{ReachedL}(s^F)$.

Two notions, of completion plan and postfix plan, are required because postfix plans are (in contrast to the standard setting) not suited to define optimality. The decoupled plan leading to the cheapest postfix plan may differ from that leading to the cheapest completion plan. This is because the postfix plan ignores the price of the $s^F$ leaf states it starts from.

The original definition of SSS in states $s$ relies on the basic concepts of disjunctive action landmarks, action interference, necessary enabling sets, and action applicability. For a corresponding definition for decoupled states $s^F$, the concept of action interference remains the same, but all other concepts must be extended. We start with applicability:

**Definition 1 (Action Applicability).** Let $s^F$ be a decoupled state. A center action $a$ is applicable in $s^F$ if $\text{ct}(s^F) = \text{pre}(a)$. A leaf action $a$ affecting leaf $F^L$ is applicable in $s^F$ if $\text{ct}(s^F) = \text{pre}(a)[F^C]$, and there exists a leaf state $s^L \in \text{ReachedL}(s^F)$ such that $s^L \models \text{pre}(a)[F^L]$. The set of actions applicable in $s^F$ is denoted with $\text{app}^{dec}(s^F)$.

Note that this definition encompasses both, center actions and leaf actions. This is in contrast to the decoupled search which branches only over (applicable) center actions. Thus the notion of “applicability” as per Definition 1 is different from the notion used in decoupled search. It is better suited for the definition of strong stubborn sets, lending itself to a direct extension of the original definition.

Let us next focus on the concept of necessary enabling sets. Given an action $a$ whose preconditions are not true, a necessary enabling set should be a set of actions one of which must necessarily be applied in order to enable $a$. In the standard setting, such a set is trivial to obtain, by picking a precondition value not currently true and selecting all actions achieving that value. In decoupled search, this is not as easy because decoupled states do not assign unique values to leaf-factor state variables. We adjust the concept as follows:

**Definition 2 (Decoupled Necessary Enabling Set).** Let $s^F$ be a decoupled state, and let $a$ be an inapplicable action $a \notin \text{app}^{dec}(s^F)$. An action set $A$ is a decoupled necessary enabling set for $a$ in $s^F$ if either of the following cases holds:

(i) $A = \{a' \in A | \text{eff}(a')[v] = \text{pre}(a)[v]\}$ where $v \in \text{vars(pre}(a)) \cap F^C$ s.t. $\text{ct}(s^F)[v] \neq \text{pre}(a)[v]$.

(ii) $A = \{a' \in A | \text{eff}(a')[v] = \text{pre}(a)[v]\}$ where $v \in \text{vars(pre}(a)) \setminus F^C$ s.t., for all $s^L \in \text{ReachedL}(s^F)$, we have $s^L \not\models \text{pre}(a)[v]$.

(iii) $A = \bigcup_{v \in V} \{a' \in A | \text{eff}(a')[v] = \text{pre}(a)[v]\}$ where $V \neq \emptyset$ is the set of all $v \in \text{vars(pre}(a))$ s.t. exists $s^L \in \text{ReachedL}(s^F)$ with $s^L \not\models \text{pre}(a)[v]$.

Case (i) in this definition corresponds to the standard setting, where $A$ are the achievers of an open precondition on the center, whose assignment is fixed in $s^F$. Case (ii) captures the situation where a leaf precondition is false in all reachable leaf states. Case (iii) is relevant because a leaf action may have several preconditions, each satisfied by some reachable leaf state, but not all satisfied jointly in any reachable leaf state. We then collect the achievers of preconditions open in any reachable leaf state. Clearly, in every case at least one $a' \in A$ must be used by any postfix plan for $s^F$ that contains $a$. At least one of the cases must apply as $a \notin \text{app}^{dec}(s^F)$. For center actions, only case (i) is possible. For leaf actions, we first test (ii), then (iii), and finally (i), the motivation being to focus on the open center preconditions “closest” to $s^F$.

We finally need to adjust the concept of disjunctive action landmarks. Given our notion of postfix plans, this is direct:

**Definition 3 (Decoupled Disjunctive Action Landmark).** Let $s^F$ be a non-goal decoupled state. An action set $L$ is a decou-
pled disjunctive action landmark for \( s^F \) if, for all decoupled plans \( \pi^F \) for \( s^F \), we have \( \text{PostPlan}(\pi^C) \cap L \neq \emptyset \).

In our implementation, we find decoupled disjunctive action landmarks simply in terms of a necessary enabling set for the goal condition \( s^* \), i.e., exactly as in Definition 2 but using \( s^* \) as the precondition of a hypothetical action \( a \).

The last basic concept we need is the standard notion of interference between pairs of actions. We say that \( a \) and \( a' \) interfere if \( v \) s.t. \( \text{eff}(a') = [v] \neq \text{eff}(a) = [v] \) or \( \text{eff}(a') [v] \neq \text{pre}(a') [v] \) or \( \text{eff}(a) [v] \neq \text{pre}(a) [v] \). Decoupled strong stubborn sets are now defined as follows:

**Definition 4 (DSSS for Non-Goal Decoupled States).** Let \( s^F \) be a non-goal decoupled state. An action set \( T_s \) is a decoupled strong stubborn set (DSSS) for \( s^F \) if the following conditions hold:

(i) \( T_s \) contains a decoupled disjunctive action landmark.
(ii) For all actions \( a \in T_s \) and \( a \notin \text{app}^{dec}(s^F) \), \( T_s \) contains a decoupled necessary enabling set for \( a \).
(iii) For all center actions \( a \in T_s \) and \( a \in \text{app}^{dec}(s^F) \), \( T_s \) contains all actions that interfere with \( a \).

Thanks to the adapted basic concepts, this definition mirrors the original one [Alkhazraji et al., 2012]. Intuitively, condition (i) ensures that \( T_s \) makes progress to the goal; condition (ii) ensures that \( T_s \) backchains all the way to the current state; condition (iii) ensures that, if we branch over \( a \), then we also branch over all actions that may be in conflict with \( a \). All three conditions are identical to the respective original one, modulo the adapted basic concepts. The single exception is the restriction to center actions in condition (iii). We do not need to include interfering actions for applicable leaf actions. That is so because postfix plans do not contain applicable leaf actions anyhow: everything that can be done using such actions is already reachable in \( s^F \).

By adapting the proof arguments from the standard setting, one can show that DSSS preserve optimality:

**Theorem 1.** Let \( s^F \) be a solvable non-goal decoupled state. Let \( T_s \) be a DSSS in \( s^F \). Then \( T_s \) contains a center action that starts an optimal decoupled plan for \( s^F \).

The proof considers any decoupled plan \( \pi^F \) for \( s^F \). Denote \( \pi = (a_1, \ldots, a_m) = \text{PostPlan}(\pi^F) \), and let \( i \) be the smallest index so that \( a_i \in T_s \). For the same reasons as shown in the original proof for SSS [Alkhazraji et al., 2012], such \( a_i \) exists, must be applicable, and – together with the fact that \( a_i \) must be a center action, as \( \text{PostPlan}(\pi^F) \) does not contain any applicable leaf actions – can be moved to the front of \( \pi \).

**4 SSS for Goal Decoupled States**

Say we are facing a goal decoupled state \( s^F_\bullet \). Instead of actions required for reaching the goal, we need to capture actions required to reduce the leaf-goal prices. One may consider to define landmarks relative to the decoupled plans reaching states \( t^F_\bullet \) where \( \text{LocalCost}(t^F_\bullet) < \text{LocalCost}(s^F_\bullet) \), and then re-use the remainder of Definition 4 unchanged. Indeed, this was our first solution attempt. The problem is that the landmark actions may pertain to leaf states already reached, only at non-optimal prices; and then we may miss the actions required to reduce those prices.

To illustrate, say that, as before, we have a leaf action \( \text{buy-car} \) (cost 1000) applicable to \( s^F \), and a center action \( \text{get-manager-job} \) which enables leaf action \( \text{get-company-car} \) (cost 0). However, now the leaf goal is not \( \text{have-car} \), but \( \text{be-at-NYC} \) for which another leaf action \( \text{drive-car} \) is needed. Then \( \{\text{drive-car}\} \) is a landmark: Any optimal completion plan for \( s^F_\bullet \) has to use this action behind \( s^F_\bullet \), i.e., after applying another center action. But \( \text{drive-car} \) is applicable in \( s^F_\bullet \), so Definition 4 would stop here, and \( T_s \) would not contain \( \text{get-company-car} \). In other words, the notion of necessary enabling sets is suited to reachability but is not suited to capture what’s needed to decrease prices.

We tackle this situation through a notion of frontier actions, required to make any progress on the prices:

**Definition 5 (Frontier Action).** Let \( s^F \) be a decoupled goal state, and let \( a \) be a leaf action affecting leaf \( F^L \). We say that \( a \) is a frontier action in \( s^F \) if (i) \( a \notin \text{app}^{dec}(s^F) \); and (ii) there exists a leaf state \( s^F_\bullet \in \text{Reached}(s^*_{\bullet}) \) such that \( s^F = \text{pre}(a)[F^L] \), and, denoting the outcome of applying \( a \) to \( s^F \) with \( t^F \), \( \text{prices}(s^F_\bullet)[s^F_\bullet] + \text{cost}(a) < \text{prices}(s^F)[t^F] \).

The frontier of \( s^F_\bullet \) is the set of all frontier actions in \( s^F_\bullet \).

In words, the frontier consists of those leaf actions that are not currently applicable, but enabling whose center precondition would result in a reduced price for at least one leaf state. This set of actions now takes the role of the landmark:

**Definition 6 (DSSS for Goal Decoupled States).** Let \( s^F_\bullet \) be a goal decoupled state. An action set \( T_s \) is a decoupled strong stubborn set (DSSS) for \( s^F_\bullet \) if the following conditions hold:

(i) \( T_s \) contains the frontier of \( s^F_\bullet \).
(ii) For all actions \( a \in T_s \) and \( a \notin \text{app}^{dec}(s^F_\bullet) \), \( T_s \) contains a decoupled necessary enabling set for \( a \).
(iii) For all center actions \( a \in T_s \) and \( a \in \text{app}^{dec}(s^F_\bullet) \), \( T_s \) contains all actions that interfere with \( a \).

Consider now a state \( s^F_\bullet \) where \( \{\} \) is not an optimal decoupled plan, i.e., we can find a better plan below \( s^F_\bullet \). Consider any decoupled plan \( \pi^F \) leading to \( t^F_\bullet \) where \( \text{LocalCost}(t^F_\bullet) < \text{LocalCost}(s^F_\bullet) \). Then \( \text{ComPlan}(\pi^F) \) contains at least one frontier action \( \alpha_F \), intuitively because these actions are needed to decrease prices relative to \( s^F_\bullet \). By construction, \( \alpha_F \) has a center precondition not satisfied in \( s^F_\bullet \). Therefore, with the inclusion of necessary enabling sets, we get that \( T_s \) must contain an applicable center action \( \alpha \) of \( \pi^F \). For the same reasons as before we can move \( \alpha \) to the front, proving that DSSS as per Definition 6 preserve optimality:

**Theorem 2.** Let \( s^F_\bullet \) be a goal decoupled state for which \( \{\} \) is not an optimal decoupled plan. Let \( T_s \) be a decoupled strong stubborn set for \( s^F_\bullet \). Then \( T_s \) contains a center action that starts an optimal decoupled plan for \( s^F_\bullet \).

Observe that \( \text{Frontier}(s^F_\bullet) \) may be empty. In that case, the DSSS will be empty, too. This is valid because, in this case, necessarily \( \{\} \) is an optimal decoupled plan for \( s^F_\bullet \), i.e., no better plan can be found below \( s^F_\bullet \) and the search can stop.
5 Exponential Separations

Before proceeding to the empirical part of our research, let us state some basic theoretical facts evaluating the power of DSSS. We say that a search space reduction method X is exponentially separated from a method Y if there exists a parameterized example family F such that, on F, X yields an exponentially stronger reduction than Y.

Decoupled search and SSS are complementary in that each is exponentially separated from the other:

**Theorem 3.** Fork-decoupled search is exponentially separated from SSS, and vice versa.

Our running example with locations A and B is a suitable family F for the first claim. There are only 3 reachable decoupled states (s_0^F; drive to B; drive back). But SSS do not yield any pruning because, in any state s, to make progress to the goal, T_s must include an applicable (un)load action; which interferes with the applicable drive action; which in turn interferes with all applicable (un)load actions. The opposite claim follows from examples, e.g. IPC Parcprinter, with no fork factoring but strong SSS pruning.

Trivially, DSSS is exponentially separated from each of fork-decoupled search and SSS, simply because DSSS naturally generalizes each of these components, so we can use the same families F as in Theorem 3. As a much stronger testimony to the power of DSSS, there are cases where it is exponentially separated from both its components:

**Theorem 4.** There exists a parameterized example family F such that, on F, DSSS yields an exponentially stronger reduction than both, fork-decoupled search and SSS.

Two suitable families F arise from simple modifications of our running example. First, say we have M trucks and N*M packages, where each truck t_i is associated with a group of N packages that only t_i can transport. The number of reachable decoupled states is exponential in M because all trucks must be in the center factor. The SSS-pruned reachable standard state space has size exponential in N because including an (un)load action into T_s necessitates, due to interference via the truck move as above, to include all applicable (un)load actions for the respective package group. However, in decoupled search with DSSS pruning, there are only M reachable states. This is because the two sources of pruning power combine gracefully. Decoupling gets rid of the blow-up in N (the packages within a group become independent leaves), while DSSS gets rid of the blow-up in M (only a single truck is committed to at a time).

In our second example, DSSS even is exponentially more than the sum of its components: stubborn sets have exponentially more impact on the decoupled search space than on the standard one. Say we have N packages and M trucks (where every truck may transport every package). Then decoupled search blows up in M, and SSS does not do anything because any package may require any truck. Applying DSSS to decoupled search, no truck move is pruned in s_0^F. However, after applying any one drive(t_i, A, B) action, all package prices are the cheapest possible ones, the frontier is empty, and DSSS stops the search. So, again, there are only M reachable states. As we shall see next, similar phenomena seem to occur in the standard IPC Logistics benchmarks.

6 Experiments

We extended GH’s implementation of fork-decoupled search in FD [Helmert, 2006]. To extract the fork factorings, we use GH’s method. It computes the strongly connected components (SCCs) of the causal graph, and, arranging the acyclic graph of SCCs with roots “at the top” and leaves “at the bottom”, greedily finds a “horizontal line” through that graph. The part above the line becomes the center, each weakly connected component below the line becomes a leaf. The technique abates if there is ≤ 1 leaf, the rationale being that decoupling pays off mainly through avoiding enumeration across > 1 leaves. We show results for those benchmarks on which the technique does not abstain. From the International Planning Competition (IPC) STRIPS benchmarks (’98–’14), this is the case for instances from 12 domains.

We focus here on optimal planning, the main purpose of the optimality-preserving pruning via strong stubborn sets. We run A+ with a blind heuristic as a measure of search space size, and with LM-cut [Helmert and Domshlak, 2009] as a representative of the state of the art, using GH’s method (Fork-Decoupled A+) to adopt these techniques for decoupled search. We compare decoupled search with DSSS pruning (simply referred to as “DSSS” in what follows) against decoupled search without that pruning (“DS” in what follows). We furthermore compare against A+ in the standard state space without pruning (“A” in what follows), and with SSS pruning (“SSS” in what follows). All experiments are run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

Table 1 shows coverage results. The most important comparison for our purposes here is that between DSSS vs. DS, i.e., the direct benefit our pruning technique yields over the baseline search. DSSS is rarely worse (NoMystery -2 and TPP -1 for blind search, only NoMystery -1 for LM-cut), it is often better (6 domains for blind, 4 domains for LM-cut), and consequently is better, though not dramatically better, in the overall. Comparing to A+ and SSS, we see that DSSS improves DS whenever (i.e., in all domains where) SSS improves A+. Whenever SSS outperforms DS, DSSS fully makes up for this advantage: the per-domain coverage of DSSS dominates that of SSS. The single exception to the

<table>
<thead>
<tr>
<th>Domain</th>
<th>#</th>
<th>Blind Heuristic</th>
<th>LM-cut</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A+</td>
<td>SSS</td>
</tr>
<tr>
<td>Driverlog</td>
<td>20</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>Logistics’00</td>
<td>28</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Logistics’98</td>
<td>35</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Miconic</td>
<td>145</td>
<td>50</td>
<td>45</td>
</tr>
<tr>
<td>NoMystery</td>
<td>20</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Pathways</td>
<td>29</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Rovers</td>
<td>40</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Satellite</td>
<td>36</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>TPP</td>
<td>27</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Wood’08</td>
<td>13</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Wood’11</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Zenotravel</td>
<td>20</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td><strong>Σ</strong></td>
<td>418</td>
<td>109</td>
<td>106</td>
</tr>
</tbody>
</table>

Table 1: Coverage (number of instances solved).
latter is Miconic, where DSSS just inherits the weakness (run-time overhead at not much search gain) of decoupled search.

Figure 1 shows fine-grained performance data. Consider first the scatter plots. The plot at the top reveals that DSSS often improves over DS, up to 2 orders of magnitude on commonly solved instances, while bad cases are consistently limited to a moderate overhead. The plot SSS vs. DS shows that, without pruning, decoupled search is in the advantage yet also incurs several bad cases. We see in the plot SSS vs. DSSS that, with DSSS pruning, this risk mostly disappears.

The table in Figure 1 shows data for those domains where DSSS sometimes reduces expansions relative to DS (we discuss the other domains below). For each of blind search and LM-cut, from the instances solved by at least one method, we selected at most 5, namely the most challenging ones (largest expansions under standard A*). Where these did not include an instance solved by all methods, to exemplify the cross-comparison we included the most challenging such instance.

As the table shows, on those domains where DSSS does yield pruning, it consistently improves over DS, both in expansions and runtime, for both blind search and LM-cut. The behavior in Logistics is especially remarkable. On the standard state space, SSS yields little or no reduction, while in the decoupled state space, DSSS yields strong reductions. This establishes a practical case of DSSS being more than the sum of its components. Compared to SSS, decoupled search with DSSS is superior in Logistics, Pathways, and Rovers, and is inferior in Satellite; the picture in Woodworking is mixed.

On the domains where DSSS does not reduce expansions (Driverlog, Miconic, NoMystery, TPP, and Zenotravel), a runtime overhead is incurred. For blind search, we get slow-down factors up to 218.5 in NoMystery, 26.9 in TPP, and 4.8 in the other 3 domains. This is due to the small per-state search effort in blind search, relative to which computing a DSSS can consume substantial runtime. For the state-of-the-art search using LM-cut, where per-state effort is much higher, the overhead is small. The maximum (geometric mean) slow-down factor is 1.3 (1.1) for Driverlog, 2.0 (1.0) for Miconic, 3.1 (2.2) for NoMystery, 2.0 (1.3) for TPP, and 2.1 (1.1) for Zenotravel. Using a simple “safety belt” which switches DSSS off after 1000 expansions if no action was pruned, the slow-down disappears in almost all cases.

7 Conclusion

We have shown that fork-decoupled search and strong stubborn sets combine gracefully in theory, and that the combination can yield good results in practice. Our next step will be to extend decoupled strong stubborn sets to star-topology decoupling as per Gnäd and Hoffmann [2015b]. More generally, decoupled search is a new paradigm that, presumably,
can be fruitfully combined not only with (heuristic search and) strong stubborn sets, but also with other search techniques like symmetry reduction or symbolic representations.

Acknowledgments

Daniel Gnad was partially supported by the German Research Foundation (DFG), as part of project grant HO 2169/6-1, “Star-Topology Decoupled State Space Search”. Martin Wehrle was supported by the Swiss National Science Foundation (SNSF) as part of the project “Automated Reformulation and Pruning in Factored State Spaces (ARAP)”.

References


