Decoupled Strong Stubborn Sets (Technical Report)

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 π^C, the leaf moves *compliant* with π^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; Helmert, 2006]).

In a fork factoring, the leaves are "conditionally independent", in the sense that, given a fixed center path π^C, the *compliant* leaf moves – those leaf moves enabled by the preconditions supplied along π^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 Helmert, 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.

Some proofs are replaced with proof sketches in the main text. The full proofs are in Appendix 8.

2 Background

We employ a finite-domain state variable formalization of planning (e.g. [Bäckström and Nebel, 1995; Helmert, 2006]). A *finite-domain representation* planning task, short FDR task, is a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star \rangle$. $\mathcal{V}$ is a set of *state variables*, each $v \in \mathcal{V}$ associated with a finite domain $\mathcal{D}(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to $\mathcal{V}$ is a *state*. s_0 is the *initial state*, and the *goal* $s_\star$ is a partial assignment to V. $\mathcal{A}$

is a finite set of *actions*. Each action $a \in \mathcal{A}$ is a triple $\langle pre(a), \mathit{eff}(a), cost(a)\rangle$ where the *precondition* $pre(a)$ and *effect* $\mathit{eff}(a)$ are partial assignments to $\mathcal{V}$, with $\mathit{eff}(a) \neq \emptyset$; $cost(a) \in \mathbb{R}^{0+}$ is a's non-negative *cost*.

Given a partial assignment p, by $vars(p) \subseteq \mathcal{V}$ we denote the subset of state variables on which p is defined. For $V \subseteq vars(p)$, by $p[V]$ we denote the assignment to V made by p. Action a is *applicable* in state s if $s \models pre(a)$, i.e., $pre(a) \subseteq s$. Applying a in s changes the value of $v \in vars(\mathit{eff}(a))$ to $\mathit{eff}(a)[v]$, and leaves s unchanged elsewhere. A *plan* for Π is an action sequence π applicable in s_0 and ending in a state s such that $s \models s_\star$. The plan is *optimal* if its summed-up cost, denoted $cost(\pi)$, is minimal among all plans for Π.

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

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

As a running example, consider a Logistics-style planning task with 1 truck variable t, N package 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_i, A) requires both t and p_i to be at A, and moves p_i into the truck). Setting $\{t\}$ as the center and each $\{p_i\}$ as a leaf, we obtain a fork factoring.

Not every task Π has a fork factoring. We assume GH's approach of analyzing Π's causal graph in a pre-process, identifying a fork factoring if one exists, else *abstaining* from solving Π. In what follows, assume a fork factoring $\mathcal{F}$.

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., $vars(pre(a)) \subseteq F^C \cup F^L$.

Due to these action behaviors, a fork factoring encapsulates a particular form of "conditional independence" between leaves. Assume a *center path* π^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 π^L *complies* with π^C if it uses only the center preconditions supplied along π^C, i.e., if π^L can be scheduled alongside π^C so that the combined action sequence is applicable in s_0. Intuitively, fixing π^C, the compliant leaf paths are the possible leaf moves given π^C. Observe that these possible moves are independent across leaf factors F^L, i.e., for each F^L we can choose a compliant π^L independently from that choice for any other leaf factor. Hence we can search over center paths π^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* $s^\mathcal{F}$ is given by a center path $\pi^C(s^\mathcal{F})$. It is associated with its *center state* $ct(s^\mathcal{F})$, simply the outcome of applying $\pi^C(s^\mathcal{F})$ to $s_0[F^C]$; and with its *pricing function* $prices(s^\mathcal{F})$. 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 $\pi^C(s^\mathcal{F})$ and ends in s^L (or ∞ 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": $prices(s^\mathcal{F})[s^L]$ 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 $s^\mathcal{F}$ later on.

The *initial decoupled state* $s_0^\mathcal{F}$ results from the empty center path $\pi^C(s_0^\mathcal{F}) = \langle\rangle$. We denote by $ReachedL(s^\mathcal{F})$ the set of leaf states s^L *reachable* in $s^\mathcal{F}$, i.e., where $prices(s^\mathcal{F})[s^L] < \infty$. A *goal decoupled state* $s_\star^\mathcal{F}$ is one with a *goal center state* $ct(s_\star^\mathcal{F}) \models s_\star[F^C]$ and where, for every leaf factor $F^L \in \mathcal{F}^L$, there exists a reachable *goal leaf state* s^L, i.e., $s^L \in ReachedL(s_\star^\mathcal{F})$ such that $s^L \models s_\star[F^L]$. The actions applicable in $s^\mathcal{F}$ are those center actions whose precondition is satisfied in $ct(s^\mathcal{F})$ (recall here that we do not branch over leaf actions). Applying a to $s^\mathcal{F}$ results in $t^\mathcal{F}$ where $\pi^C(t^\mathcal{F}) := \pi^C(s^\mathcal{F}) \circ \langle a\rangle$, and $ct(t^\mathcal{F})$ as well as $prices(t^\mathcal{F})$ arise from $\pi^C(t^\mathcal{F})$ as defined above.

In our example, $ct(s_0^\mathcal{F}) = \{(t, A)\}$, 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 ∞. 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 $s_0^\mathcal{F}$ in the decoupled search is the center action *drive*(t, A, B), leading to the goal decoupled state $s_\star^\mathcal{F}$ where $ct(s_\star^\mathcal{F}) = \{(t, B)\}$ 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 $s_\star^\mathcal{F}$ is reached, a plan π for the input task Π can be constructed by augmenting the center path $\pi^C(s_\star^\mathcal{F})$ with compliant leaf paths ending in goal leaf states $s_\star^L$ (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 $\langle$*load*(t, p_i, A),*unload*$(t, p_i, B)\rangle$.

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

$LocalCost(s_\star^\mathcal{F})$ is optimal for $s_\star^\mathcal{F}$ (locally optimal) but not necessarily optimal for Π (globally optimal). Indeed, it can happen that, from $s_\star^\mathcal{F}$, a *better* plan can be obtained from a descendant of $s_\star^\mathcal{F}$. This is because, with additional center actions, cheaper leaf paths may become available. For example, say in $s^\mathcal{F}$ the leaf goal *have-car* has price 1000 via the applicable leaf action *buy-car*. But if we apply a center action *get-manager-job*, then the leaf action *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 [Alkhazraji *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 $T_s \subseteq \mathcal{A}$ constructed so that, for every plan π for s, at least one permutation of π 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" $s^{\mathcal{F}}$ is more complex. GH did not require, so did not introduce, such a concept. For our purposes, the following notions will suffice.

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

Clearly, to preserve optimality, it suffices for T_s to contain at least one center action starting an optimal decoupled plan for $s^{\mathcal{F}}$. Towards identifying sets T_s qualifying for this, we will need to focus exclusively on the part of the completion plan "behind" $s^{\mathcal{F}}$. We denote this by $PostPlan(\pi^{\mathcal{F}})$, the *postfix plan*. The center action subsequence in $PostPlan(\pi^{\mathcal{F}})$ is $\pi^C(\pi^{\mathcal{F}})$. For any leaf factor F^L, say $\pi^L = \langle a_1^L, \dots, a_n^L\rangle$ is the goal leaf path for F^L in $ComPlan(\pi^{\mathcal{F}})$, traversing leaf states $\langle s_0^L, \dots, s_n^L\rangle$. Then the leaf action subsequence for F^L in $PostPlan(\pi^C)$ is defined as $\langle a_{i+1}^L, \dots, a_n^L\rangle$, where i is the highest index for which $s_i^L \in ReachedL(s^{\mathcal{F}})$. In other words, we consider the postfix of π^L not contained in $ReachedL(s^{\mathcal{F}})$.

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 $s^{\mathcal{F}}$ leaf states it starts from.[1]

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 $s^{\mathcal{F}}$, the concept of action interference remains the same, but all other concepts must be extended. We start with applicability:

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

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

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

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

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

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 $s^{\mathcal{F}}$. 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

[1]For example, assume trucks t_1 and t_2 in our running example, both starting with all N packages at A, where all unloading actions have cost 1, loading a package into t_1 has cost 1 while loading it into t_2 has cost 2, and driving t_1 has cost 2 while driving t_2 has cost 1. Then the optimal decoupled plan for $s_0^{\mathcal{F}}$ is $drive(t_1, A, B)$, to a plan cost of $2 + 2N$. But the postfix plan of $drive(t_1, A, B)$ has cost $2 + N$, and a cheaper postfix plan, of cost $1 + N$, is obtained by the decoupled plan $drive(t_2, A, B)$.

$a' \in A$ must be used by any postfix plan for $s^{\mathcal{F}}$ that contains a. At least one of the cases must apply as $a \notin app^{dec}(s^{\mathcal{F}})$. 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 $s^{\mathcal{F}}$.

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 $s^{\mathcal{F}}$ be a non-goal decoupled state. An action set L is a* decoupled disjunctive action landmark *for $s^{\mathcal{F}}$ if, for all decoupled plans π^C for $s^{\mathcal{F}}$, we have $PostPlan(\pi^C) \cap L \neq \emptyset$.*

In our implementation, we find decoupled disjunctive action landmarks simply in terms of a necessary enabling set for the goal condition $s_\star$, i.e., exactly as in Definition 2 but using $s_\star$ 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 ex. v s.t. $\mathit{eff}(a')[v] \neq \mathit{eff}(a)[v]$ or $\mathit{eff}(a)[v] \neq pre(a')[v]$ or $\mathit{eff}(a')[v] \neq pre(a)[v]$. Decoupled strong stubborn sets are now defined as follows:

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

(i) T_s contains a decoupled disjunctive action landmark.

(ii) For all actions $a \in T_s$ and $a \notin app^{dec}(s^{\mathcal{F}})$, T_s contains a decoupled necessary enabling set for a.

(iii) For all center actions $a \in T_s$ and $a \in app^{dec}(s^{\mathcal{F}})$, 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 $s^{\mathcal{F}}$.

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

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

The proof considers any decoupled plan $\pi^{\mathcal{F}}$ for $s^{\mathcal{F}}$. Denote $\pi = \langle a_1, \dots, a_m \rangle = PostPlan(\pi^{\mathcal{F}})$, 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 $PostPlan(s^{\mathcal{F}})$ does not contain any applicable leaf actions – can be moved to the front of π.

Concluding this section, observe that SSS is the special case of DSSS where no decoupling takes place. Namely, consider the *trivial fork factoring* where all state variables are in the center. This is not actually a fork factoring according to the definition, as it has only a single component. But it is useful as a theoretical border case, where decoupled search defaults to standard search. Using the trivial fork factoring, DSSS as specified simplifies to SSS. This is easy to verify based on the adapted definitions, all of which trivialize to the standard ones in this case.

4 SSS for Goal Decoupled States

Say we are facing a goal decoupled state $s^{\mathcal{F}}_\star$. 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 $t^{\mathcal{F}}_\star$ where $LocalCost(t^{\mathcal{F}}_\star) < LocalCost(s^{\mathcal{F}}_\star)$, 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 *buy-car* (cost 1000) applicable to $s^{\mathcal{F}}$, and a center action *get-manager-job* which enables leaf action *get-company-car* (cost 0). However, now the leaf goal is not *have-car*, but *be-at-NYC* for which another leaf action *drive-car* is needed. Then $\{$*drive-car*$\}$ is a landmark: Any optimal completion plan for $s^{\mathcal{F}}$ has to use this action behind $s^{\mathcal{F}}$, i.e., after applying another center action. But *drive-car* is applicable in $s^{\mathcal{F}}$, so Definition 4 would stop here, and T_s would not contain *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 $s^{\mathcal{F}}_\star$ 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 $s^{\mathcal{F}}_\star$ if (i) $a \notin app^{dec}(s^{\mathcal{F}}_\star)$; and (ii) there exists a leaf state $s^L \in ReachedL(s^{\mathcal{F}}_\star)$ such that $s^L \models pre(a)[F^L]$, and, denoting the outcome of applying a to s^L with t^L, $prices(s^{\mathcal{F}}_\star)[s^L] + cost(a) < prices(s^{\mathcal{F}}_\star)[t^L]$.*

The frontier *of $s^{\mathcal{F}}_\star$ is the set of all frontier actions in $s^{\mathcal{F}}_\star$.*

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 $s^{\mathcal{F}}_\star$ be a goal decoupled state. An action set T_s is a* decoupled strong stubborn set (DSSS) *for $s^{\mathcal{F}}_\star$ if the following conditions hold:*

(i) T_s contains the frontier of $s^{\mathcal{F}}_\star$.

(ii) For all actions $a \in T_s$ and $a \notin app^{dec}(s^{\mathcal{F}}_\star)$, T_s contains a decoupled necessary enabling set for a.

(iii) For all center actions $a \in T_s$ and $a \in app^{dec}(s^{\mathcal{F}}_\star)$, T_s contains all actions that interfere with a.

Consider now a state $s^{\mathcal{F}}_\star$ where $\langle\rangle$ is not an optimal decoupled plan, i.e., we can find a better plan below $s^{\mathcal{F}}_\star$. Consider any decoupled plan $\pi^{\mathcal{F}}$ leading to $t^{\mathcal{F}}_\star$ where $LocalCost(t^{\mathcal{F}}_\star) < LocalCost(s^{\mathcal{F}}_\star)$. Then $ComPlan(\pi^{\mathcal{F}})$

contains at least one frontier action a_F, intuitively because these actions are needed to decrease prices relative to $s^{\mathcal{F}}_\star$. By construction, a_F has a center precondition not satisfied in $s^{\mathcal{F}}_\star$. Therefore, with the inclusion of necessary enabling sets, we get that T_s must contain an applicable center action a of $\pi^{\mathcal{F}}$. For the same reasons as before we can move a to the front, proving that DSSS as per Definition 6 preserve optimality:

Theorem 2. *Let $s^{\mathcal{F}}_\star$ be a goal decoupled state for which $\langle\rangle$ is not an optimal decoupled plan. Let T_s be a decoupled strong stubborn set for $s^{\mathcal{F}}_\star$. Then T_s contains a center action that starts an optimal decoupled plan for $s^{\mathcal{F}}_\star$.*

Observe that $Frontier(s^{\mathcal{F}})$ may be empty. In that case, the DSSS will be empty, too. This is valid because, in this case, necessarily $\langle\rangle$ is an optimal decoupled plan for $s^{\mathcal{F}}_\star$, i.e., no better plan can be found below $s^{\mathcal{F}}_\star$ 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 $\mathcal{F}$ such that, on $\mathcal{F}$, 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 $\mathcal{F}$ for the first claim. There are only 3 reachable decoupled states ($s^{\mathcal{F}}_0$; 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 $\mathcal{F}$ 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 $\mathcal{F}$ such that, on $\mathcal{F}$, DSSS yields an exponentially stronger reduction than both, fork-decoupled search and SSS.*

Two suitable families $\mathcal{F}$ 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 $s^{\mathcal{F}}_0$. 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 abstains 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

	Blind Heuristic									LM-cut								
	Expansions				Runtime						Expansions				Runtime			
Inst	A*	SSS	DS	DSSS	A*	SSS	DS	DSSS	Inst	A*	SSS	DS	DSSS	A*	SSS	DS	DSSS
Logistics'00																		
p6-9	368109	368109	30	9	2.2	5.2	0.0	0.0	p12-0	116544	116544	149	98	132.3	141.8	0.2	0.1
p12-0	–	–	8101	882	–	–	15.1	0.2	p14-0	–	–	4130	2193	–	–	17.1	6.6
p12-1	–	–	22644	1338	–	–	46.4	0.3	p14-1	–	–	8263	4726	–	–	42.3	17.9
p14-0	–	–	–	197855	–	–	–	605.8	p15-0	–	–	41259	15977	–	–	280.3	62.5
p14-1	–	–	–	324152	–	–	–	1256.9	p15-1	–	–	11710	5978	–	–	59.6	21.7
Logistics'98																		
p1	–	–	75954	15404	–	–	325.48	14.2	p1	12634	12634	555	379	13.6	15.2	1.0	0.5
p5	–	–	–	20410	–	–	–	45.24	p31	56	56	12	12	0.0	0.0	0.0	0.0
p31	133855	133855	586	224	1.31	3.53	0.16	0.04	p32	108	47	20	15	0.0	0.0	0.0	0.0
p32	218003	218003	368	124	1.39	3.3	0.04	0.01	p33	92692	92692	388	104	85.8	94.2	0.4	0.1
p33	–	–	3550	592	–	–	1.43	0.2	p35	1636	1636	360	360	11.8	12.5	4.8	1.8
Pathways																		
p2	2916	531	2366	489	0.0	0.0	0.0	0.0	p3	98	64	98	66	0.0	0.0	0.0	0.0
p3	53603	2252	16030	609	0.3	0.0	0.7	0.0	p4	189	150	189	150	0.0	0.0	0.1	0.0
p4	300600	10331	31903	2131	3.8	0.2	2.6	0.1	p5	46402	6675	27346	3989	51.8	6.2	39.9	4.1
Rovers																		
p5	7.52M	213647	152871	6861	71.5	5.3	15.2	0.5	p5	71222	4562	9533	1154	9.7	0.5	2.2	0.2
p6	–	1.20M	6.28M	28693	–	21.0	763.8	1.9	p8	–	–	1.16M	1.00M	–	–	896.1	630.5
p7	32.78M	25.19M	522185	406676	301.1	489.6	43.8	35.2	p9	–	892779	–	8573	–	205.5	–	3.8
p9	–	–	–	397564	–	–	–	55.8	p12	19195	11788	8915	4915	9.9	5.1	6.1	2.6
p12	–	–	–	1.52M	–	–	–	278.8	p14	–	–	–	780716	–	–	–	481.7
Satellite																		
p2	1539	1471	303	249	0.0	0.0	0.0	0.0	p7	95606	3204	77253	10735	120.8	4.9	156.7	12.3
p3	13243	5839	1484	857	0.1	0.1	0.1	0.1	p9	–	3722	–	40514	–	35.1	–	194.1
p4	274070	14510	27706	16225	3.1	0.3	19.0	7.9	p10	–	172718	–	–	–	1399.1	–	–
p5	22.98M	3.01M	364513	217733	636.5	106.2	181.1	149.6	p11	–	0	–	0	–	9.2	–	16.2
p6	19.81M	142382	2.17M	121935	402.7	6.3	1358.1	40.2	p18	–	8366	–	4665	–	98.2	–	140.5
Woodworking'08																		
p1	9797	1002	1	1	0.1	0.0	0.0	0.0	p7	32418	177	224	46	601.0	1.7	5.4	1.2
p2	23287	70	0	0	0.2	0.0	0.0	0.0	p8	–	694	–	5268	–	10.8	–	61.8
p9	–	1.65M	–	30851	–	88.9	–	4.5	p9	–	5157	1103	43	–	5.7	9.3	0.3
p16	–	–	–	1.78M	–	–	–	223.1	p24	9868	425	615	168	19.8	0.4	1.3	0.3
p24	–	137867	1210202	21721	–	5.2	120.6	1.7	p30	–	308	525	62	–	50.7	463.6	18.1
Woodworking'11																		
p5	–	137867	1.21M	21721	–	5.4	132.4	1.7	p5	9868	425	615	168	19.9	0.4	1.2	0.3
p12	–	–	–	1.78M	–	–	–	220.7	p12	–	18317	22072	1920	–	30.3	118.9	5.5
									p13	–	0	0	0	–	0.1	0.2	0.1
									p16	32418	177	224	46	621.2	1.7	5.6	1.1
									p19	–	694	–	5268	–	10.4	–	61.5

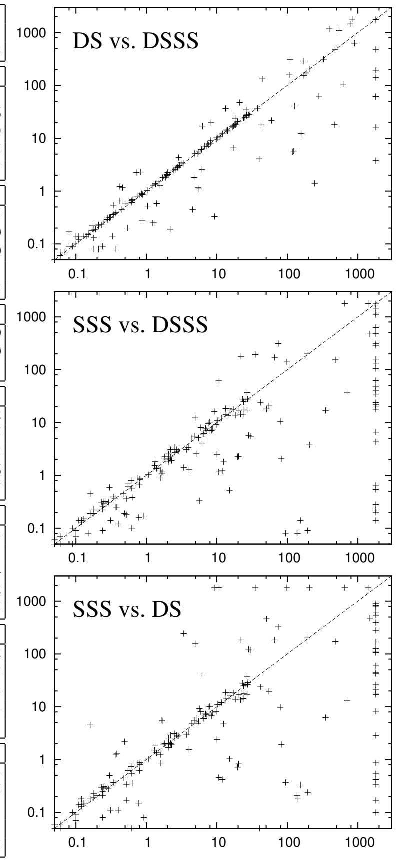

Figure 1: Runtime, and expansions to prove optimality (before last f-layer in A^*). Table: Per-instance data on selected IPC instances "Inst" (see text). "M": million, "–": out of time or memory. Scatter plots: Runtime with LM-cut, for all pairs "X vs. Y" of non-baseline configurations. X on x-axis, Y on y-axis, time-out 1800 seconds inserted for unsolved instances.

		Blind Heuristic				LM-cut			
Domain	#	A^*	SSS	DS	DSSS	A^*	SSS	DS	DSSS
Driverlog	20	7	7	11	11	13	13	13	13
Logistics'00	28	10	10	22	24	20	20	28	28
Logistics'98	35	2	2	4	5	6	6	6	6
Miconic	145	50	45	35	36	136	136	135	135
NoMystery	20	8	7	17	15	14	14	20	19
Pathways	29	3	3	3	3	4	4	4	4
Rovers	40	6	7	7	9	7	9	9	11
Satellite	36	6	6	6	6	7	11	7	11
TPP	27	5	5	23	22	5	5	18	18
Wood'08	13	4	6	5	7	6	11	10	11
Wood'11	5	0	1	1	2	2	5	4	5
Zenotravel	20	8	7	11	11	13	13	13	13
$\sum$	418	109	106	145	151	233	247	267	274

Table 1: Coverage (number of instances solved).

fully makes up for this advantage: the per-domain coverage of DSSS dominates that of SSS. The single exception to the latter is Miconic, where DSSS just inherits the weakness (runtime 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 slowdown 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 2.0 (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 Gnad 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

[Alkhazraji *et al.*, 2012] Yusra Alkhazraji, Martin Wehrle, Robert Mattmüller, and Malte Helmert. A stubborn set algorithm for optimal planning. In Luc De Raedt, Christian Bessiere, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter Lucas, editors, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 891–892. IOS Press, 2012.

[Amir and Engelhardt, 2003] Eyal Amir and Barbara Engelhardt. Factored planning. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 929–935. Morgan Kaufmann, 2003.

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[Brafman and Domshlak, 2003] Ronen I. Brafman and Carmel Domshlak. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, 18:315–349, 2003.

[Brafman and Domshlak, 2013] Ronen Brafman and Carmel Domshlak. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71, 2013.

[Fabre *et al.*, 2010] Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In Ronen Brafman, Héctor Geffner, Jörg Hoffmann, and Henry Kautz, editors, *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 65–72. AAAI Press, 2010.

[Gnad and Hoffmann, 2015a] Daniel Gnad and Jörg Hoffmann. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, pages 88–96. AAAI Press, 2015.

[Gnad and Hoffmann, 2015b] Daniel Gnad and Jörg Hoffmann. From fork decoupling to star-topology decoupling. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, pages 53–61. AAAI Press, 2015.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169. AAAI Press, 2009.

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

[Jonsson and Bäckström, 1995] Peter Jonsson and Christer Bäckström. Incremental planning. In Malik Ghallab and Alfredo Milani, editors, *New Directions in AI Planning: EWSP '95 — 3rd European Workshop on Planning*, volume 31 of *Frontiers in Artificial Intelligence and Applications*, pages 79–90, Amsterdam, 1995. IOS Press.

[Kelareva *et al.*, 2007] Elena Kelareva, Olivier Buffet, Jinbo Huang, and Sylvie Thiébaux. Factored planning using decomposition trees. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1942–1947, 2007.

[Knoblock, 1994] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[Valmari, 1989] Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (APN 1989)*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag, 1989.

[Wehrle and Helmert, 2012] Martin Wehrle and Malte Helmert. About partial order reduction in planning and computer aided verification. In Lee McCluskey, Brian Williams, José Reinaldo Silva, and Blai Bonet, editors, *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press, 2012.

[Wehrle and Helmert, 2014] Martin Wehrle and Malte Helmert. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning* *and Scheduling (ICAPS 2014)*, pages 323–331. AAAI Press, 2014.

8 Proofs

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

Proof. Let $\pi^{\mathcal{F}}$ be any decoupled plan for $s^{\mathcal{F}}$. Let $\pi := PostPlan(\pi^{\mathcal{F}})$ and denote $\pi = \langle a_1, \dots, a_n \rangle$.

As T_s contains a decoupled disjunctive action landmark, there is at least one action in π that is contained in T_s. Consider the action a_i in π and T_s with smallest index, i.e., $\{a_1, \dots, a_{i-1}\} \cap T_s = \emptyset$.

Observe first that a_i is applicable in $s^{\mathcal{F}}$. If this was not so, then a necessary enabling set A would have to be contained in T_s to enable a_i, and one action from A would necessarily be contained in π in front of a_i, contradicting that a_i is the shared action with smallest index.

Observe next that, therefore, a_i must be a center action. As a_i is applicable, if it was a leaf action then its outcome state would be contained in $ReachedL(s^{\mathcal{F}})$, and therefore $a_i \notin PostPlan(\pi^{\mathcal{F}})$ in contradiction.

Observe finally that a_i can be moved to the front of π. If this was not so, then a_i would have to interfere with an action in $\{a_1, \dots, a_{i-1}\}$. However, in that case, as per Definition 4 (iii), the interfering action would have to be contained in T_s as well, again contradicting the assumption on the smallest index.

As a_i can be moved to the front of π, a_i can be moved to the front of $\pi^C(\pi^{\mathcal{F}})$. Altogether, for any decoupled plan $\pi^{\mathcal{F}}$ for $s^{\mathcal{F}}$, T_s contains a center action starting a permutation of $\pi^{\mathcal{F}}$. The claim follows. □

Theorem 2. *Let $s^{\mathcal{F}}_\star$ be a goal decoupled state for which $\langle\rangle$ is not an optimal decoupled plan. Let T_s be a decoupled strong stubborn set for $s^{\mathcal{F}}_\star$. Then T_s contains a center action that starts an optimal decoupled plan for $s^{\mathcal{F}}_\star$.*

Proof. Let $\pi^{\mathcal{F}}$ be an optimal decoupled plan for $s^{\mathcal{F}}_\star$. As the empty plan, $\langle\rangle$, is not optimal for $s^{\mathcal{F}}_\star$, $\pi^{\mathcal{F}}$ must lead to $t^{\mathcal{F}}_\star$ where $LocalCost(t^{\mathcal{F}}_\star) < LocalCost(s^{\mathcal{F}}_\star)$.

We first prove that $ComPlan(\pi^{\mathcal{F}})$ contains at least one frontier action. Towards this, let F^L be any leaf factor on which a goal is defined. Say that $\langle a_1, \dots, a_n \rangle$ is the goal leaf path for F^L in $ComPlan(\pi^{\mathcal{F}})$, traversing F^L leaf states $\langle s^L_0, \dots, s^L_n \rangle$. Define $cost(s^L_i) := \sum_{j=1}^{i} cost(a_i)$ to be the cost of s^L_i on the given path. Define i as the highest index for which $cost(s^L_i) = prices(s^{\mathcal{F}}_\star)[s^L_i]$. At least one such index must exist as $cost(s^L_0) = 0 = prices(s^{\mathcal{F}}_\star)[s^L_0]$. Define π to be like $ComPlan(\pi^{\mathcal{F}})$, but containing, for every F^L, only the leaf-path postfix behind this index i.

Let π^L be a non-empty leaf path in π. Then a_{i+1} is a frontier action in $s^{\mathcal{F}}_\star$. To see this, observe first that a_{i+1} is applicable in s^L_i by construction, and s^L_i is reached in $s^{\mathcal{F}}_\star$ by construction. Therefore, a_{i+1} must have a center precondition not satisfied in $s^{\mathcal{F}}_\star$, or else we would have $cost(s^L_{i+1}) = prices(s^{\mathcal{F}}_\star)[s^L_{i+1}]$ in contradiction to i being the highest index with this property. In particular, $a_{i+1} \notin app^{dec}(s^{\mathcal{F}}_\star)$ as required for (i) in Definition 5. Furthermore, by construction we have that $cost(s^L_{i+1}) < prices(s^{\mathcal{F}}_\star)[s^L_{i+1}]$, which shows (ii) in Definition 5 because $cost(s^L_{i+1}) = cost(s^L_i) + cost(a_{i+1}) = prices(s^{\mathcal{F}}_\star)[s^L_i] + cost(a_{i+1})$.

Clearly, as $LocalCost(t^{\mathcal{F}}_\star) < LocalCost(s^{\mathcal{F}}_\star)$, we furthermore have that π contains at least one non-empty leaf path, because otherwise the leaf prices in $t^{\mathcal{F}}_\star$ could not be smaller than those in $s^{\mathcal{F}}_\star$.

Now we now, in particular, that π contains at least one leaf action a_{i+1} with an open center precondition, and that this same action a_{i+1} is contained in T_s. As T_s contains necessary enabling sets as per Definitions 6 (ii) and 2, it follows that π and T_s must share an applicable center action. Let a be the first such action in π.

Observe that a can be moved to the front of π. If this was not so, then a would have to interfere with an action preceding a in π. However, in that case, as per Definition 6 (iii), the interfering action would have to be contained in T_s as well, contradicting that a is the first one.

As a can be moved to the front of π, a can be moved to the front of $\pi^C(\pi^{\mathcal{F}})$. Altogether, for any decoupled plan $\pi^{\mathcal{F}}$ for $s^{\mathcal{F}}_\star$ leading to $t^{\mathcal{F}}_\star$ where $LocalCost(t^{\mathcal{F}}_\star) < LocalCost(s^{\mathcal{F}}_\star)$, T_s contains a center action starting a permutation of $\pi^{\mathcal{F}}$. The claim follows. □

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

Proof. Consider again our running example with locations A and B. There are only 3 reachable decoupled states: in $s^{\mathcal{F}}_0$, all packages can be at A or loaded (i.e., these are the reachable leaf states); driving to B, all packages can also be at B; driving back to A yields the same center state but a different pricing function. SSS, on the other hand, does not yield any pruning. In any state s, to make progress to the goal, T_s must include one applicable *load* or *unload* action; which interferes with the applicable *drive* action; which in turn interferes with all applicable *load*/*unload* actions.

Vice versa, consider a task family $\mathcal{F}$ where N binary variables are initially 0 and each must be set to 1, with an individual action. This task has no fork factoring at all. But, using SSS, every state has only a single non-pruned action. □

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

Proof. Consider our example but with 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 (all trucks and packages start at A, all packages must be transported to B). The number of reachable decoupled states is exponential in M, because all trucks must be in the center factor, and their move combinations are enumerated. For SSS, as soon as T_s contains a *load*/*unload* action for one group of N packages, the *load*/*unload* actions for all other packages in that group are present as well, due to interference as before. So the SSS-pruned reachable state space has size exponential in N.

Consider now decoupled search with DSSS pruning. In $s^{\mathcal{F}}_0$, all packages can be at A or loaded into their respective truck. The landmark will select one package, associated with some truck t_i; hence T_s includes *drive*(t_i, A, B). This does not interfere with the *drive* actions for the other trucks, so it is the *only* applicable center action in T_s, and we get a single successor state $s^{\mathcal{F}}$. In $s^{\mathcal{F}}$, the packages associated with t_i can all be at B. So the landmark for DSSS selects a package associated with another truck $t_j \neq t_i$. The only non-pruned action is *drive*(t_j, A, B); and so forth. Once all trucks are at B, we have a goal decoupled state $s^{\mathcal{F}}_{\star}$. $Frontier(s^{\mathcal{F}}_{\star}) = \emptyset$ as the package prices are already the cheapest possible ones. So there are exactly M reachable decoupled states. $\square$