Beyond Forks: Finding and Ranking Star Factorings for Decoupled Search

Daniel Gnad, Valerie Poser and **Jörg Hoffmann**
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de; s9vapose@stud.uni-saarland.de

Abstract

Star-topology decoupling is a recent search reduction method for forward state space search. The idea basically is to automatically identify a *star factoring*, then search only over the center component in the star, avoiding interleavings across leaf components. The framework can handle complex star topologies, yet prior work on decoupled search considered only factoring strategies identifying fork and inverted-fork topologies. Here, we introduce factoring strategies able to detect general star topologies, thereby extending the reach of decoupled search to new factorings and to new domains, sometimes resulting in significant performance improvements. Furthermore, we introduce a predictive portfolio method that reliably selects the most suitable factoring for a given planning task, leading to superior overall performance.

1 Introduction

In classical 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. The states, goal condition, and actions are described relative to a vector of state variables, and the size of the resulting state space is exponential in the number of state variables. Numerous techniques have been proposed to tackle this state space explosion problem. Star-topology decoupled search, short *decoupled search*, is a recent addition to this arsenal [Gnad and Hoffmann, 2015; Gnad *et al.*, 2015].

Decoupled search is a form of factored planning (e. g. Amir and Engelhardt [2003], Kelareva *et al.* [2007], Fabre *et al.* [2010], Brafman and Domshlak [2013]), where the state variables are automatically partitioned into *factors* (components), and the planning process distinguishes between local per-factor planning vs. global across-factors planning. In contrast to previous approaches, star-topology decoupled search assumes that the interactions across factors take a particular shape, namely that of a star topology. Such a topology has a single *center* factor that can arbitrarily interact with possibly many *leaf* factors, but any interaction across leaf factors must be via the center. The decoupled search then branches only over those actions that affect the center, handling the possible moves for each leaf factor separately.

Intuitively, one can think of this as exploiting a form of "conditional independence" between the leaf factors: given a fixed transition path π^C for the center, the possible (*center-compliant*) transition paths for each leaf are independent across leaves. We can therefore avoid the multiplication of leaf states across leaves. Instead, decoupled search accumulates all possible leaf states given π^C into a so-called *decoupled state*, which compactly represents the set of all states that can be reached using π^C. The number of decoupled states can be, and is often in practice, exponentially smaller than the number of states in the standard state space.

While decoupled search is able to handle star topologies in general, existing factoring strategies so far only identified fork and inverted-fork topologies. Such structures are well known in planning [Katz and Domshlak, 2008; Katz and Keyder, 2012; Aghighi *et al.*, 2015] and can be easily detected by analyzing the *causal graph* of a planning task (e. g. Knoblock [1994], Jonsson and Bäckström [1995], Brafman and Domshlak [2003], Helmert [2003]). Yet fork and inverted-fork topologies are quite limited. In particular, they cannot identify any factorization within strongly connected components of the causal graph.

Here we extensively widen the scope of star factorings, introducing two new strategies that, in particular, do not suffer from this limitation. We aim at maximizing the number of leaf components, as the potential reductions are exponential in that number. Our first strategy is based on *maximum independent sets* of the causal graph, which yield the maximum possible number of leaves. The leaves are then post-processed, as larger leaves are more beneficial: we design a greedy strategy maximizing leaf *flexibility*, derived from the number of actions that affect only a leaf (and which the search hence doesn't need to branch over). Our second strategy is simpler. It employs greedy variable selection using a measure of connectivity in the causal graph, essentially moving the most densely connected variables into the center.

Both strategies extend the reach of decoupled search to new factorings and to new domains, and sometimes result in significant performance improvements. On the other hand, it turns out that the two new factoring strategies, as well as the previous ones, are often complementary (lead to good results in different cases). So how to automatically select the best factoring? We devise a *predictive portfolio* – a per-instance self-configuration method – to answer that question.

Sequential portfolios, running a set of component planners for a fixed allotted time each, have been used widely in planning (e. g. Howe *et al.* [1999], Gerevini *et al.* [2009], Helmert *et al.* [2011], Seipp *et al.* [2015]). Yet, to our awareness, the only known predictive planning portfolio is IBACOP [Cenamor *et al.*, 2016], which predicts the performance of a set of component planners based on a wide range of input-syntax features. Such features are also more generally used for performance prediction in planning [Roberts and Howe, 2009; Hoffmann, 2011; Fawcett *et al.*, 2014]. We go beyond this here by a ranking method based on *sample searches*, running the factoring candidates with a short time limit, extracting features from the searches. This is different from IBACOP, whose key to success is the component selection, not the per-instance ranking. Our approach works well. It reliably selects the best factoring, leading to superior overall performance.

2 Preliminaries

We consider a classical planning framework with finite-domain state variables [Bäckström and Nebel, 1995; Helmert, 2006], often referred to as *FDR* (finite-domain representation) planning. A *planning task* in this framework is a tuple $\Pi = \langle V, A, I, G\rangle$. Here, V is a finite set of *state variables*, short *variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. A is a finite set of *actions*, each $a \in A$ being a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{c}(a)\rangle$ of *precondition*, *effect*, and *cost*, where $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ are partial assignments to V, and the cost is a non-negative real number $\mathsf{c}(a) \in \mathbb{R}^{0+}$. A *state* is a complete assignment to V. I is the *initial state*. The *goal* G is a partial assignment. For a partial assignment p, we denote by $\mathcal{V}(p) \subseteq V$ the subset of variables on which p is defined. For $V' \subseteq \mathcal{V}(p)$, by $p[V']$ we denote the restriction of p onto V', i. e., the assignment to V' made by p. We identify (partial) variable assignments with sets of variable/value pairs.

We say that action a is *applicable* in state s if $\mathsf{pre}(a) \subseteq s$. Applying a in s changes the value of all $v \in \mathcal{V}(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$, and leaves s unchanged elsewhere. The outcome state is denoted $s[\![a]\!]$. A *plan* for Π is an action sequence π iteratively applicable in I, and resulting in a state s_G where $G \subseteq s_G$. The plan π is *optimal* if the summed-up cost of its actions, denoted $\mathsf{c}(\pi)$, is minimal among all plans for Π.

The factoring strategies we will consider heavily employ the task's *causal graph*, a well known concept in planning capturing some of a task's structure, in terms of pairwise state-variable dependencies (e. g. Knoblock [1994], Jonsson and Bäckström [1995], Brafman and Domshlak [2003], Helmert [2006]). Specifically, the causal graph CG_Π of a planning task Π is a directed graph whose vertices are the variables V. The graph contains an arc $v \to v'$ if $v \neq v'$, and there exists an action $a \in A$ such that $v \in \mathcal{V}(\mathsf{pre}(a)) \cup \mathcal{V}(\mathsf{eff}(a))$ and $v' \in \mathcal{V}(\mathsf{eff}(a))$. Intuitively, an arc from v to v' indicates that, either, in order to move v' ($v' \in \mathcal{V}(\mathsf{eff}(a))$) we may have to move v first ($v \in \mathcal{V}(\mathsf{pre}(a))$); or, when we move v', we may have to move v as a side effect ($v \in \mathcal{V}(\mathsf{eff}(a))$).

We assume that CG_Π is weakly connected, which makes some factoring topologies more convenient to define. If that is not so, then each weakly connected component can be cast as a separate sub-task, and can be solved independently.

3 Decoupled Search Background

To understand our contribution, a recap of the previous work by Gnad and Hoffmann [2015] and Gnad *et al.* [2015] is required. We define what star factorings are, give a brief summary of decoupled search, and discuss previous methods for finding star factorings automatically.

3.1 Star Factorings

A *factoring* $\mathcal{F}$ is a partitioning of V into non-empty subsets $F \subseteq V$. These sets are called *factors*. We say that a factoring $\mathcal{F}$ is *trivial* if it contains only a single factor, i. e., $|\mathcal{F}| = 1$. A non-trivial factoring $\mathcal{F}$ is called a *star factoring* if there exists a factor $F^C \in \mathcal{F}$ where, denoting the remaining factors by $\mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$, for every action $a \in A$ where $\mathcal{V}(\mathsf{eff}(a)) \cap F^C = \emptyset$ there exists $F^L \in \mathcal{F}^L$ with $\mathcal{V}(\mathsf{eff}(a)) \subseteq F^L$ and $\mathcal{V}(\mathsf{pre}(a)) \subseteq F^L \cup F^C$. In other words, every action either affects (has an effect on) F^C, or it affects only a single $F^L \in \mathcal{F}^L$ and has preconditions only on that same F^L and/or on F^C. In this situation, we refer to F^C as the *center*, and to the factors in $\mathcal{F}^L$ as the *leaves*, in $\mathcal{F}$.

Intuitively, in a star factoring, the center dominates the task structure, as fixing a transition path for the center also fixes what any one of the leaves can or cannot do. Decoupled search exploits this by searching only over the actions affecting the center.

3.2 Decoupled Search

We need a few basic notations. Given a factoring $\mathcal{F}$, variable assignments to F^C are called *center states*, and assignments to an $F^L \in \mathcal{F}^L$ are called *leaf states*. We refer to actions affecting F^C as *center actions*, denoted A^C, and to actions affecting a leaf F^L as *leaf actions*, denoted $A^L[F^L]$. The set of all leaf actions is denoted A^L. Observe that A^C and $A^L[F^L]$ may overlap: this happens if there is a center action that affects both, F^C and F^L. We call leaf actions in $A^L \setminus A^C$ *leaf-only* actions. In a star factoring, center actions can have arbitrary preconditions and effects on all factors. On the other hand, as pointed out above already, leaf-only actions affect only F^L, and have preconditions only on $F^L \cup F^C$.

Given a path π^C of center actions (a *center path*, applicable to $I[F^C]$ in the task's projection onto F^C), decoupled search maintains – for each leaf separately – what is referred to as the *compliant-path graph*. The compliant-path graph compactly captures the set of leaf paths (sequences of $A^L[F^L]$ actions) that *comply* with the current center path. Here, a leaf path π^L of leaf F^L complies with a center path π^C if their $A^C \cap A^L[F^L]$ subsequences agree, and the leaf-only actions in π^L can be embedded into π^C such that the resulting action sequence is applicable to $I[F^C \cup F^L]$ when ignoring preconditions on the remaining leaves $\mathcal{F}^L \setminus \{F^L\}$.

A *decoupled state* $s^{\mathcal{F}}$ then consists of a center path π^C, along with the compliant-path graph for each leaf $F^L \in \mathcal{F}^L$. For each leaf state s^L, the *price* of s^L in $s^{\mathcal{F}}$ is the cost of a cheapest compliant leaf path ending in s^L, or ∞ if no such path exists. The search stops when reaching a goal decoupled state: a state whose center assignment satisfies the center goal $G[F^C]$, and where for each leaf $F^L \in \mathcal{F}^L$ there exists a finite-price leaf state that satisfies the leaf goal $G[F^L]$.

3.3 Existing Factoring Strategies

Gnad *et al.* [2015] introduced the theoretical concept of star factorings, yet explored only a tiny fragment of that rich factoring space. They considered only extremely simple sub-classes arising from well-known structures called *forks* and *inverted-forks*, as well as a combination thereof, called *Xshape*. These structures can be derived from simple causal graph analyses, viewing the factoring as an equivalence relation over the variables, i. e., over the causal graph vertices.

The *interaction graph* $\mathsf{IG}_\Pi(\mathcal{F})$ given a factoring $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $F \to F'$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $v \to v'$ is an arc in CG_Π. A factoring $\mathcal{F}$ is a *fork factoring* if there exists $F^C \in \mathcal{F}$ such that the arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are exactly $\{F^C \to F^L \mid F^L \in \mathcal{F}^L\}$. $\mathcal{F}$ is an *inverted-fork factoring* if there exists $F^C \in \mathcal{F}$ such that the arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are exactly $\{F^L \to F^C \mid F^L \in \mathcal{F}^L\}$. $\mathcal{F}$ is an *Xshape factoring* if there exists $F^C \in \mathcal{F}$ such that, for every $F^L \in \mathcal{F}^L$, exactly one of $F^C \to F^L$ and $F^L \to F^C$ is an arc in $\mathsf{IG}_\Pi(\mathcal{F})$.

Every fork/inverted-fork/Xshape factoring is, in particular, a star factoring. The advantage of these simple special cases is that they are very easy to identify. Observe that, in any one of these factoring types, as the dependency between each pair of factors can only be in one direction, *(*) every strongly connected component (SCC) of the causal graph must be fully contained in a single factor*. One can hence identify such factorings from the DAG over causal-graph SCCs, which in practice tends to be very small. Gnad *et al.* [2015] devise simple greedy strategies attempting to (though not guaranteeing to) maximize the number of leaf factors.[1]

However, (*) is of course a stark limitation. For example, if the causal graph is strongly connected, then no factoring can be identified – this despite the fact that *every* planning task has exponentially many star factorings, namely for example all those partitioning V into two subsets (where the role of "center" vs. "leaf" can be attributed arbitrarily).

4 Finding Star Topologies

We design factoring strategies making more comprehensive use of the possibilities at hand. The strategies identify *strict-star factorings*. A factoring $\mathcal{F}$ is a strict-star factoring if there exists $F^C \in \mathcal{F}$ s.t. all arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are contained in $\{F^C \to F^L, F^L \to F^C \mid F^L \in \mathcal{F}^L\}$. In other words, we now allow arbitrary (including bidirectional) causal-graph dependencies between the center and the leaves. This definition has been stated by Gnad *et al.* [2015] before, but its power has never been explored.[2]

We introduce two factoring strategies. Both aim at maximizing the number of leaf factors in a strict-star factoring. The latter is **NP**-complete [Gnad *et al.*, 2015], due to a simple reduction from finding a *maximum independent set* (MIS): leaf factors are independent in the causal graph, and vice versa independent causal graph variables can be made leaves. Since maximizing the number of leaves is already hard, we focus on optimizing only this measure. In the end, the potential gain of decoupled search is exponential in that number. Many other features could also be considered, though. We leave the conclusive exploration of such features for future work, instead concentrating on how to detect star topologies.

Our first strategy uses a causal-graph MIS as a seed factoring, which is then post-processed. Our second strategy is simpler, using a greedy variable selection that moves the most densely connected variables into the center.

4.1 Maximizing the Number of Leaves

Given the correspondence between causal-graph maximum independent sets and strict-star factorings with maximum number of leaves, a natural approach to find the latter is by starting from a causal graph MIS $V_{\text{MIS}} \subset V$. In our implementation, we use standard methods to find such a MIS [Fomin *et al.*, 2009]; precisely, we consider all cardinality-maximal independent sets produced up to a time-out of 10 seconds. Each MIS then spawns a separate instance of our factoring strategy. The strategy starts with a factoring $\mathcal{F}_{\text{MIS}}$ where every leaf contains exactly one variable $v \in V_{\text{MIS}}$. It applies a post-process designed to maximize leaf *flexibility*. It may also *abstain* (see below) depending on the outcome.

The flexibility of a leaf F^L is the ratio of leaf-only actions over all actions affecting F^L: $|A^L[F^L] \setminus A^C|/|A^L[F^L]|$. We say that a leaf is *frozen* if its flexibility is 0. A leaf that is not frozen is called *mobile*, and a factoring $\mathcal{F}$ is mobile if all its leaves are mobile. Flexibility measures the "amount of work" a leaf can do on its own. In particular, it is 1 for fork or inverted-fork leaves. A frozen leaf cannot lead to a reduction of the search space, in the sense that all its leaf actions also affect the center, so must be branched over. We hence remove frozen leaves, using only mobile factorings in the search.

The MIS post-process is detailed in Figure 1. Starting with a MIS-factoring $\mathcal{F}_{\text{MIS}}$, we (1) move variables from the center into the leaves, (2) remove the frozen leaves, and (3) maximize the number of variables in each leaf. For (1) and (3), we use a hill-climbing approach to select the variables (*maximizeFlexibility*). We do so by computing a set of *candidate* variables that are connected to exactly one leaf factor, and that can hence be moved into that leaf factor without introducing leaf-leaf dependencies. For each candidate c, a candidate factoring is generated, where c has been moved into the respective leaf. Out of the candidate factorings resulting from the last for-loop execution, we pick the one with the highest number of mobile leaves (in the argmax), then we iterate.

Frozen leaves are removed in step (2), because any leaf that is still frozen at this point cannot become mobile later

[1] We remark that, in fact, one can easily guarantee to find fork (respectively inverted-fork) factorings with the maximal number of leaves. Namely, that holds when simply setting $\mathcal{F}^L$ to the leaf (root) causal-graph SCCs. This is because adding a non-leaf (non-root) component as a new leaf factor necessarily introduces a dependency across leaf factors. Adding a component to an existing leaf can only increase its size, but can never lead to more leaf factors. We will use these enhanced fork/inverted-fork strategies in our experiments.

[2] We shun the complexity of general star factorings because (a) they are defined based on individual actions and cannot be captured by a compact structural representation like the causal graph; and (b) their usefulness over strict-star factorings is unclear as the only additional possibility are center actions affecting several leaves, an implicit form of dependency across leaves.

IndependentSetFactoring($\Pi = \langle V, A, I, G \rangle$)**:**
$\mathcal{F}^L := \{\{v\} \mid v \in \texttt{MIS}(\mathsf{CG}_\Pi)\}$
$\mathcal{F}^L :=$ *maximizeFlexibility*($\mathcal{F}^L$) (1)
$\mathcal{F}^L := \{F^L \in \mathcal{F}^L \mid F^L$ is mobile$\}$ (2)
$\mathcal{F}^L :=$ *maximizeFlexibility*($\mathcal{F}^L$) (3)
if $|\{F^L \in \mathcal{F}^L \mid F^L$ *is mobile*$\}| < 2$ **then**
  return *abstain*
return $\mathcal{F} := \{F^C := \{v \in V \mid \forall F^L \in \mathcal{F}^L : v \notin F^L\}\} \cup \mathcal{F}^L$

Function *maximizeFlexibility* ($\mathcal{F}_0^L$)**:**
  $F^C := \{v \in V \mid \forall F^L \in \mathcal{F}_0^L : v \notin F^L\}$
  $V' := \{v \in F^C \mid |neighbourLeaves(v, \mathcal{F}_0^L)| = 1\}$
  $\mathcal{F}_{\text{max}}^L := \mathcal{F}_0^L$
  $i := 1$
  while $V' \neq \emptyset$ **do**
    for $v \in V'$ **do**
      $\{F_v^L\} := neighbourLeaves(v, \mathcal{F}_{\text{max}}^L)$
      $\mathcal{F}_i^L := \mathcal{F}_{\text{max}}^L \cup \{F_v^L \cup \{v'\}\} \setminus \{F_v^L\}$
      $i := i + 1$
    end
    $\mathcal{F}_{\text{max}}^L :=$ argmax$(|\mathcal{F}_j^L|)$, for $j \in [i - |V'|, i - 1]$
    $F^C := \{v \in V \mid \forall F^L \in \mathcal{F}_{\text{max}}^L : v \notin F^L\}$
    $V' := \{v \in F^C \mid |neighbourLeaves(v, \mathcal{F}_{\text{max}}^L)| = 1\}$
  end
  return $\mathcal{F}_j^L$ with max # of mobile leaves, $j \in [0, i-1]$

Function *neighbourLeaves* ($v, \mathcal{F}^L$)**:**
  $N := \{F^L \in \mathcal{F}^L \mid \exists v' \in F^L : v \to v' \in \mathsf{CG}_\Pi \vee v' \to v \in \mathsf{CG}_\Pi\}$
  return N

Figure 1: Factoring strategy based on a maximum independent set (MIS) of the causal graph.

on. Step (3) aims at further increasing flexibility. Note that additional iterations cannot improve the flexibility, since the set of candidates would be empty. The factoring with the highest number of mobile leaves is returned.

Although increasing flexibility – by moving variables into the leaves – is desirable, having very large leaf state spaces can lead to a prohibitive computational overhead when updating the compliant-path graphs. To prevent this, like Gnad and Hoffmann [2015], we use 2^{32} as an upper bound on the domain-size product of the variables in a leaf. We remark that there might be more suitable choices for the upper bound. Experimenting with this is out of the scope of this work, though.

The algorithm may fail to find a non-trivial factoring. In that case, our factoring strategy *abstains*, i. e., does not suggest a factoring for this input task. Indeed, as done in previous work on decoupled search, we abstain – here as well as in the factoring strategy described in the next subsection – whenever the algorithm returns a factoring with at most one mobile leaf. This makes sense because the main advantage of decoupled search is to avoid interleaving across *several* leaf factors. If the factoring strategy abstains, one can in principle use any arbitrary other planner; the decision to abstain is typically taken very quickly (details in the experiments).

4.2 A Greedy Approximation

Although the causal graph is usually small, it turns out that there exist planning instances for which computing a maximum independent set is infeasible. Therefore, we propose an alternative to the independent-set factoring, based on the connectivity of the variables in the causal graph. We count the number of *incident arcs*, the number of CG-arcs a variable participates in, and move highly connected variables to the center. This method computes a factoring (or fails to do so) very quickly. Details are given in Figure 2.

IncidentArcsFactoring($\Pi = \langle V, A, I, G \rangle$)**:**
$F^C := \emptyset$
$i := 1$
for $v \in V$ **do**
  // sorted by decreasing # of incident arcs in CG_Π
  $F^C := F^C \cup \{v\}$
  $\mathcal{F}_i^L :=$ *connectedComponents*($V \setminus F^C$)
  $i := i + 1$
end
$\mathcal{F}^L :=$ select $\mathcal{F}_i^L$ with max # of mobile leaves, $i \in [1, |V|]$
if $|\{F^L \in \mathcal{F}^L \mid F^L$ *is mobile*$\}| < 2$ **then**
  return *abstain*
return $\mathcal{F} := \{F^C := \{v \in V \mid \forall F^L \in \mathcal{F}^L : v \notin F^L\}\} \cup \mathcal{F}^L$

Figure 2: A greedy factoring strategy based on the number of incident arcs of a variable in the causal graph.

The algorithm starts with the trivial factoring $\mathcal{F} = \mathcal{F}^L = \{V\}$, where all variables are in a single leaf factor. We sort the variables by decreasing number of incident arcs and move variables from the leaf to the center F^C according to this ordering. This generates a sequence of center factors where in each step the size of the center is increased by one. For each such F^C, we set the leaves $\mathcal{F}^L$ to be the weakly connected components in CG projected on the non-center part $V \setminus F^C$ that fit the leaf size bound. Picking the weakly connected components in the leaf part ensures that there are no cross-leaf dependencies, and we get a strict-star factoring. Again, we choose the factoring that maximizes the number of mobile leaves, abstaining if there are less than 2 of them.

The idea behind this strategy is to have the highly connected variables in the center, because we assume those to otherwise introduce dependencies between the leaves.

5 Predictive Per-Instance Self Configuration

Now that we have an arsenal of factoring strategies at hand, the question arises if there exists a method that dominates the others. As we shall see in the experiments, this is not the case, and the factoring methods are typically complementary. A simple way to exploit such complementarity is to combine several planners in a sequential portfolio (e. g. Howe *et al.* [1999], Gerevini *et al.* [2009], Helmert *et al.* [2011], Seipp *et al.* [2015]), where each component planner gets a fixed allotted time slot based on a benchmark training phase.

Cenamor *et al.* [2016] devised a more flexible *predictive portfolio*, IBACOP, that assigns the times based on a per-instance analysis over syntactic features of the planning task input. Here, we select one of our factorings on a per-instance basis. In contrast to Cenamor *et al.*, we do not rely on syntactic features, but run a short *sample search* with each factoring.

Specifically, we generate the set of factorings using all factoring strategies: fork, inverted-fork, Xshape, MIS-based for

	A* using LM-cut									GBFS using h^{FF}									GBFS using h^{FF} and preferred operator pruning								
Dom	#	B	F	IF	X	MIS	IA	SC	%	#	B	F	IF	X	MIS	IA	SC	%	#	B	F	IF	X	MIS	IA	SC	%
Air	17	13	-	-	-	13	-	13		17	**17**	-	-	-	16	-	16		17	17	-	-	-	17	-	17	
Csnac	20	0	-	0	0	-	0	0		20	0	-	0	0	-	0	0		20	3	-	**6**	**6**	-	**6**	**6**	
Depot	22	**7**	-	**7**	**7**	5 (11)	**7**	**7**		22	14	-	**19**	**19**	9 (11)	**19**	**19**		22	18	-	**20**	**20**	10 (11)	19	**20**	
Driv	20	13	13	-	13	13	13	13	100	20	18	**20**	-	**20**	**20**	**20**	**20**	86	20	20	20	-	20	20	20	20	100
Elev	50	40	-	**41**	**41**	9 (41)	40	**41**	83	50	48	-	**50**	**50**	-	10 (40)	**50**		50	**50**	-	**50**	**50**	-	10 (40)	**50**	
Floor	40	**13**	-	**13**	**13**	8	8	**13**	100	40	**8**	-	6	6	**8**	4	7	57	40	**8**	-	**8**	**8**	**8**	4	**8**	50
Free	42	**2**	-	-	-	-	1	1		42	41	-	-	-	-	**42**	**42**		42	42	-	-	-	-	42	42	
Log	63	26	34	27	34	**35**	**35**	**35**	100	63	54	**63**	**63**	**63**	**63**	**63**	**63**	93	63	63	63	63	63	63	63	63	98
Mico	145	**136**	135	-	135	135	135	135		145	145	145	-	145	145	145	145		145	145	145	-	145	145	145	145	
Mpri	6	**6**	-	-	-	-	4	4		6	6	-	-	-	-	6	6		6	6	-	-	-	-	6	6	
Myst	5	**1**	-	0 (1)	0 (1)	-	**1** (4)	**1**		5	1	-	1 (1)	1 (1)	-	1 (4)	**2**		5	**2**	-	1 (1)	1 (1)	-	1 (4)	**2**	
NoMy	20	14	**20**	-	**20**	**20**	**20**	**20**		20	9	**19**	-	**19**	**19**	**19**	**19**		20	10	**19**	-	**19**	**19**	**19**	**19**	
Open	70	**40**	-	-	-	38	35	38	84	70	69	-	-	-	69 (1)	**70**	**70**	46	70	**70**	-	-	-	69 (1)	**70**	69	71
Parc	23	7	-	-	-	-	**13**	**13**		25	**24**	-	-	-	-	23	23		25	24	-	-	-	-	24	24	
Pathw	30	**5**	4 (1)	-	4	**5**	**5**	**5**	100	30	11	12 (1)	-	13	15	13	**18**	100	30	20	19 (1)	-	20	23	20	**24**	100
Rover	40	7	**9**	5 (2)	**9**	**9**	**9**	**9**	88	40	23	22	**38 (2)**	22	21	21	32	90	40	**40**	**40**	38 (2)	**40**	**40**	**40**	**40**	87
Sat	36	7	7	10 (2)	7	8	9	**12**	90	36	30	33	**34 (2)**	33	33	28	33	87	36	**36**	**36**	34 (2)	**36**	**36**	31	35	84
Tetr	13	4	-	-	-	-	**5**	**5**		17	5	-	-	-	-	**6**	**6**		17	13	-	-	-	-	**14**	**14**	
Thoug	0									13	5	-	-	-	0 (10)	5	5	100	13	**10**	-	-	-	1 (10)	**10**	**10**	100
Tidy	40	22	-	-	-	8 (18)	23	**24**		20	14	-	-	-	14	14	14		20	**15**	-	-	-	13 (1)	13	13	
TPP	29	5	**18 (2)**	2 (3)	**18**	3 (26)	5	**18**	100	29	21	23 (2)	25 (3)	25	3 (26)	**26**	25	96	29	**29**	27 (2)	26 (3)	**29**	3 (26)	24	27	96
Trans	70	**23**	-	**23**	**23**	3 (67)	18 (34)	**23**	100	70	16	-	**70**	**70**	3 (67)	9 (61)	**70**	100	70	45	-	**70**	**70**	3 (67)	9 (61)	**70**	100
Truck	27	9	-	-	-	6 (13)	**10**	**10**	100	27	**16**	-	-	-	7 (13)	**16**	**16**	100	27	**18**	-	-	-	7 (13)	16	16	100
Wood	46	25	14 (28)	28 (7)	28 (7)	-	26 (3)	**33**		49	48	43 (6)	48 (1)	48 (1)	1 (48)	46 (3)	**49**	74	49	**49**	43 (6)	48 (1)	48 (1)	1 (48)	46 (3)	**49**	44
Zeno	20	**13**	**13**	11 (2)	**13**	12	12	**13**	61	20	**20**	**20**	18 (2)	**20**	**20**	**20**	**20**	100	20	**20**	**20**	18 (2)	**20**	**20**	**20**	**20**	100
Other	87	72	3 (84)	0 (87)	3 (84)	72	72	72	76	90	87	3 (87)	0 (90)	3 (87)	87 (2)	87	87	94	90	88	3 (87)	0 (90)	3 (87)	88 (50)	88	80	94
All	981	510	270	167	368	402	506	**558**		986	750	403	372	557	553	713	**857**		986	861	435	382	598	586	760	**897**	
			(560)	(557)	(330)	(331)	(58)					(540)	(553)	(326)	(343)	(125)					(540)	(553)	(326)	(344)	(125)		
MIS	620	390	241	83	271	**402**				643	525	341	172	360	**553**				642	577	369	176	392	**586**			
			(271)	(434)	(207)							(263)	(435)	(208)							(262)	(434)	(207)				
IA	923	490	270	160	361		**506**			861	685	403	267	452		**713**			861	**764**	435	277	493		760		
			(502)	(540)	(313)							(415)	(536)	(309)							(415)	(536)	(309)				

Table 1: Coverage data (number of solved instances) for the standard search baseline (B), the revised F/IF/X factorings, our new MIS and IA-based factorings, and self-configuration (SC). Best results highlighted in **bold**. # is the number of instances where at least one factoring strategy did not abstain. Numbers in parentheses show the number of abstained tasks per domain. The three rows at the bottom show overall coverage (and overall abstained) on the subset of instances where any (All), the MIS-based (MIS), or the IA-based (IA) strategy does not abstain. "-" marks domains in which a strategy abstains on all instances. We summarize domains with identical coverage for B, MIS, IA, and SC in "Other". "%" indicates SC's accuracy (see text).

every MIS found, incident arc based. For each factoring, we run a sample search with a time limit of 1s. We additionally allow up to 10s per factoring to precompute the leaf state spaces, which is important for the efficiency of decoupled search. In case a sample search already solves the task, we return the solution. In domains where just a single method finds a factoring, we simply select that factoring. Given multiple factorings, we select one based on sample search features, namely *heuristic improvement* – the ratio between initial state heuristic value vs. the best heuristic value observed in the search – as well as the number of *expanded states*.

It turns out that, to reliably select the best-performing factoring, it suffices to rank the factorings based on just one of these features depending on the objective: heuristic improvement (higher is better) for satisficing planning (where better heuristic values lead to faster search); and expanded states (higher is better) for optimal planning. For the latter, we expect the search space to be rather large, independent of the factoring, so a fast expansion rate is important (the expansion time highly depends on the factoring).

6 Experiments

We implemented the factoring strategies – the two new ones, as well as the refined fork (F), inverted-fork (IF), and Xshape (X) strategies as mentioned in Section 3.3 – for Gnad *et al.* [2015] decoupled search planner built on top of the Fast Downward system (FD) [Helmert, 2006]. The experiments were conducted on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) limits of 30 minutes (4 GB). We run experiments on the sequential optimal and satisficing tracks of all international planning competitions.

To evaluate our MIS factoring strategy (on its own, without per-instance self-configuration), in case several factorings with the same number of mobile leaves are found, we select an arbitrary one among those.

For all factoring strategies, the factoring process is fast. The factoring time is below 1s on 90% of the instances; the maximum is 30s. In instances with high factoring time, it is typically (though not always) still faster than FD's preprocess. Expectedly, the incident arcs based factoring (IA) tends to be significantly faster than the MIS-based strategy.

Our new strategies produce fork / inverted-fork / Xshape / strict-star factorings in 18% / 16% / 1% / 65% of the cases for IA, and in 32% / 6% / 0% / 62% for MIS. So IA and MIS are indeed able to detect many strict-star factorings.

In Table 1, we show coverage results of A* search with LM-cut [Helmert and Domshlak, 2009], and greedy best-first search (GBFS) with h^{FF} [Hoffmann and Nebel, 2001] with and without using preferred operators. Quite obviously, the factoring strategies differ a lot, and no strategy dominates all others. Comparing our MIS and IA strategies, it turns out that IA abstains significantly less. The reason for this are "ill-structured" maximum independent sets that, while having many independent variables, cannot be repaired to a mobile

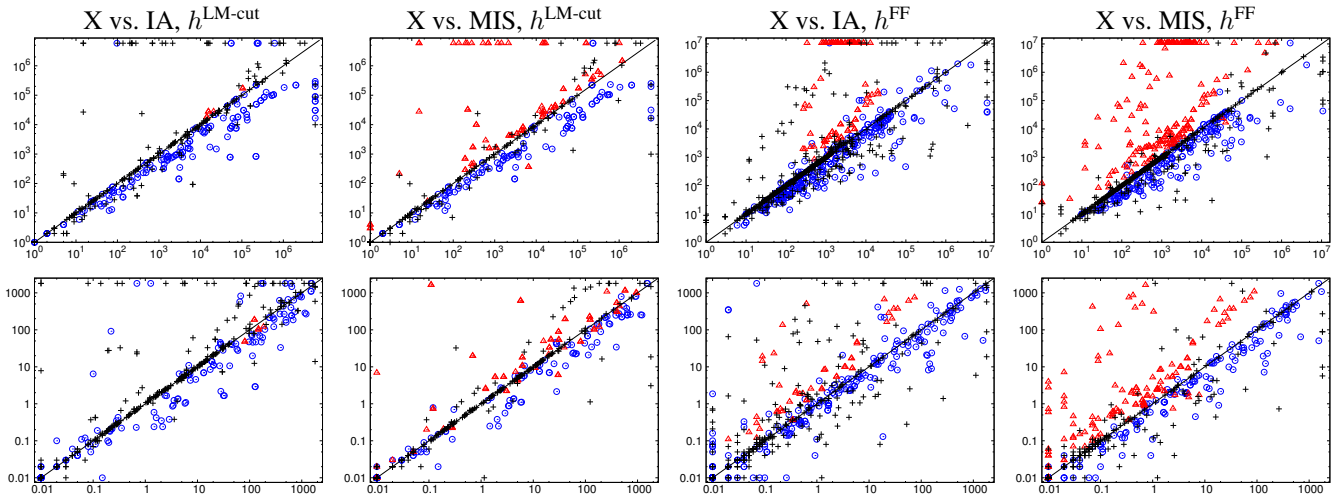

Figure 3: Scatter plots, with a data point per instance, showing the search space size (top), and runtime in seconds (bottom) of algorithms "A vs. B", with A on the x-axis and B on the y-axis. Black points indicate instances where both strategies did non abstain. Blue (red) color highlights instances for which X (IA/MIS) has abstained; we show data for B in this case.

factoring by our post-process. Furthermore, a bit unexpectedly, the IA factoring often results in a larger number of leaf factors, on instances were both strategies are successful.

In *newly tackled* domains, i. e., ones where only MIS and/or IA do not abstain, we see that optimal decoupled search with LM-cut usually solves about as many instances (± 2) as standard search. An outlier to the positive side is ParcPrinter ($+6$), to the negative side Openstacks (-3), both with IA. When using GBFS with h^{FF}, there is little coverage difference in the newly tackled domains. Most configurations solve most of the instances, so in this sense these benchmarks are just not sufficiently challenging to exhibit coverage differences. In the other domains though, where previous strategies find factorings, too, we often see big coverage differences, most notably in Rovers and Satellite.

Predictive self-configuration (SC) turns out to be very useful. Its accuracy (fraction of instances with at least 2 different factorings in which the best-performing – according to FD's search time – factoring is selected) is shown in the % columns. Clearly, accuracy is very good almost across the board, especially in optimal planning. In the optimal benchmark suite, SC chooses the F/IF/X/IA/MIS strategy 368/267/1/279/66 times, in the satisficing suite it is 350/246/2/335/53. Note that the high number of forks always includes 145 instances of Miconic. In terms of coverage, this leads to superior performance overall. This is either (1) due to combining methods that abstain on different instances, or (2) picking the right factoring on instances that can be solved when using one, but not using another factoring method.

Runtime and search space size scatter plots, shown in Figure 3, allow a more fine-grained view on the performance of different factorings. We show data for MIS and IA, comparing to X as a baseline. The first row of plots shows the per-instance comparison of the search space size (# expanded nodes until last f-layer for $h^{\text{LM-cut}}$, # evaluated states for h^{FF}), the second row shows runtime. If both factoring strategies succeed (black points), we see that they often result in the same factoring – there are many black points on the diagonal. When running $h^{\text{LM-cut}}$, it turns out that some of the instances solved by X, cannot be solved by IA. This risk is less pronounced for MIS, so the MIS-based factorings indeed seem to be better, although abstaining more often. On commonly solved instances with different factorings X is mostly faster than IA/MIS, favoring the simpler strategy if it succeeds. A positive outlier is, e. g., the Satellite domain, where MIS with $h^{\text{LM-cut}}$ achieves an average speed-up factor over B of around 27, compared to no speed-up with the X strategy. Other good cases are, e. g., Logistics and Pathways with h^{FF}, where IA gets a speed-up of 55 (32 for X), resp. 107 (13 for X) over B. In case X abstains (blue dots) we see that our new strategies very consistently outperform the baseline B, as we have already seen in the coverage table. If IA/MIS abstain (red dots) we see the same picture for X vs. B. So, although mostly invisible in the coverage table, there are cases where the new strategies perform significantly better than standard search, and sometimes even better than Xshape factorings.

7 Conclusion

Decoupled search can tackle a large set of star factorings, yet has previously been applied to fork and inverted-fork structures only. Our work begins to close this gap, with more general strict-star factorings found through maximum independent sets and greedy optimizations/approximations. The empirical results, especially with per-instance self-configuration, are reasonably good. Major improvements are rare though. The question remains whether better factoring strategies yet exist, or whether the observed limitations are simply due to the inherent structure ("we can only exploit star topologies where they are present") of these benchmarks.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/6-1, "Star-Topology Decoupled State Space Search".

References

[Aghighi *et al.*, 2015] Meysam Aghighi, Peter Jonsson, and Simon Ståhlberg. Tractable cost-optimal planning over restricted polytree causal graphs. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3225–3231. AAAI Press, January 2015.

[Amir and Engelhardt, 2003] Eyal Amir and Barbara Engelhardt. Factored planning. In *Proceedings of the 18th IJCAI*, pages 929–935, Acapulco, Mexico, August 2003. Morgan Kaufmann.

[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 Brafman and Carmel Domshlak. Structure and complexity in planning with unary operators. *JAIR*, 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. *AI*, 198:52–71, 2013.

[Cenamor *et al.*, 2016] Isabel Cenamor, Tomás de la Rosa, and Fernando Fernández. The ibacop planning system: Instance-based configured portfolios. *JAIR*, 56:657–691, 2016.

[Fabre *et al.*, 2010] Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 65–72. AAAI Press, 2010.

[Fawcett *et al.*, 2014] Chris Fawcett, Mauro Vallati, Frank Hutter, Jörg Hoffmann, Holger Hoos, and Kevin Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press, 2014.

[Fomin *et al.*, 2009] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *JACM*, 56(5), 2009.

[Gerevini *et al.*, 2009] Alfonso Gerevini, Alessandro Saetti, and Mauro Vallati. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 350–353. AAAI Press, 2009.

[Gnad and Hoffmann, 2015] Daniel Gnad and Jörg Hoffmann. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 88–96. AAAI Press, 2015.

[Gnad *et al.*, 2015] Daniel Gnad, Jörg Hoffmann, and Carmel Domshlak. From fork decoupling to star-topology decoupling. In *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, 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 *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

[Helmert *et al.*, 2011] Malte Helmert, Gabriele Röger, Jendrik Seipp, Erez Karpas, Jörg Hoffmann, Emil Keyder, Raz Nissim, Silvia Richter, and Matthias Westphal. Fast Downward Stone Soup. In *IPC 2011 planner abstracts*, pages 38–45, 2011.

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

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Hoffmann, 2011] Jörg Hoffmann. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *JAIR*, 41:155–229, 2011.

[Howe *et al.*, 1999] Adele Howe, Eric Dahlman, Chistopher Hansen, Anneliese Von Mayrhauser, and Michael Scheetz. Exploiting competitive planner performance. In *Proceedings of the 5th European Conference on Planning (ECP'99)*, pages 62–72. Springer-Verlag, 1999.

[Jonsson and Bäckström, 1995] Peter Jonsson and Christer Bäckström. Incremental planning. In *European Workshop on Planning*, 1995.

[Katz and Domshlak, 2008] Michael Katz and Carmel Domshlak. Structural patterns heuristics via fork decomposition. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 182–189. AAAI Press, 2008.

[Katz and Keyder, 2012] Michael Katz and Emil Keyder. Structural patterns beyond forks: Extending the complexity boundaries of classical planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 1779–1785, Toronto, ON, Canada, July 2012. AAAI Press.

[Kelareva *et al.*, 2007] Elena Kelareva, Olivier Buffet, Jinbo Huang, and Sylvie Thiébaux. Factored planning using decomposition trees. In *Proceedings of the 20th IJCAI*, pages 1942–1947, Hyderabad, India, January 2007. Morgan Kaufmann.

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

[Roberts and Howe, 2009] Mark Roberts and Adele Howe. Learning from planner performance. *AI*, 173(5-6):536–561, 2009.

[Seipp *et al.*, 2015] Jendrik Seipp, Silvan Sievers, Malte Helmert, and Frank Hutter. Automatic configuration of sequential planning portfolios. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3364–3370. AAAI Press, January 2015.