Beating LM-cut with $h^{\mathbf{max}}$ (Sometimes): Fork-Decoupled State Space Search

Daniel Gnad and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de

Abstract

Factored planning decouples planning tasks into subsets (*factors*) of state variables. The traditional focus is on handling complex cross-factor interactions. Departing from this, we introduce a form of *target-profile factoring*, forcing the cross-factor interactions to take the form of a *fork*, with several *leaf factors* and one potentially very large *root factor*. We show that forward state space search gracefully extends to such structure, by augmenting regular search on the root factor with maintenance of the cheapest compliant paths within each leaf factor. We analyze how to guarantee optimality. Connecting to standard heuristics, the performance improvements relative to A* are substantial, and sometimes dramatic: In four IPC benchmark domains, fork-decoupled state space search outperforms standard state space search even when using $h^{\max}$ in the former vs. LM-cut in the latter.

Introduction

Factored planning decouples planning tasks into subsets, *factors*, of state variables. In *localized* factored planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006; 2008; 2013; Fabre et al. 2010), two factors interact if they are affected by common actions, and a global plan needs to comply with these *cross-factor interactions*. In *hierarchical* factored planning (e. g. (Knoblock 1994; Kelareva et al. 2007)), the factors are used within a hierarchy of increasingly more detailed abstraction levels, accumulating the factors processed so far as search proceeds down the hierarchy.

In localized factored planning, much of the complexity stems from having to resolve complex cross-factor interactions. In hierarchical factored planning, at higher levels we don't know whether the abstract plan will be realizable at lower levels, leading to backtracking across levels. So what both localized and hierarchical factored planning algorithms share is *the ability to deal with very complex forms of cross-factor interactions*, and the difficulties entailed.

But do we actually need that ability? Provided we are willing to handle very large "factors", we can force the cross-factor interactions to be simple. We baptize this approach *target-profile factoring*: fixing a particular *profile* which we want the factoring to induce. Here, we instantiate this with *fork* (Katz and Domshlak 2008) profiles, where a single *root factor* provides preconditions required for moving several *leaf factors*. The root factor may be arbitrarily large, as we will tackle it by (heuristic) search. To obtain such a *fork factoring*, we view the causal graph as a DAG of strongly connected components (SCC) in which the root SCCs are at the top and the leaf SCCs at the bottom. We draw a horizontal line anywhere through that DAG, perceive the (entire) top part as the root factor, and perceive each weakly connected component inside the bottom part as a leaf factor. Of course, not every planning task has a useful fork factoring (e. g. if the causal graph is a single SCC). We will see this within negligible runtime at factoring time, and if so, simply *abstain* from solving the task.

The key advantage of fork factorings is a particular form of *conditional independence: for any fixed root path, the leaves can be moved independently of each other*. We exploit this by augmenting regular search on the root factor with maintenance of the cheapest *compliant paths* (paths that comply with the moves of the root) independently within each leaf factor. Compared to standard state space search, such *fork-decoupled state space search* can drastically reduce search space size, and in contrast to hierarchical factoring it avoids backtracking across levels.

We show how to naturally extend standard concepts such as A* and admissible heuristics, how to connect to standard heuristics, and how to guarantee (global cost-)optimality. Empirically, for satisficing planning the decoupling helps dramatically in transportation with fuel consumption (IPC NoMystery), but only there, because other fork-like domains are already easy for existing heuristic search techniques. For optimal planning, the empirical results are almost consistently good, and, in some domains, quite amazing.

It is worth pointing out at this early stage already that target-profile factoring, beyond fork profiles, suggests an entirely new way of exploiting structure:

Instead of relaxing *the planning task into a (structurally defined) fragment to obtain a heuristic function, try to* factorize *the task into the fragment to obtain a plan.*

This suggests a new direction for causal graph research, designing fragments suited to specialized combinatorial search algorithms, as opposed to tractability analysis. In the long term, this could lead to an entire portfolio of target profiles.

For space reasons, we need to be concise. See Gnad and Hoffmann (2015) for full proofs and additional examples.

Preliminaries

We use the finite-domain state variable setting (e.g. (Bäckström and Nebel 1995; Helmert 2006)). A *finite-domain representation* planning task, short FDR task, is a quadruple $\Pi = \langle V, A, I, G\rangle$. V is a set of *state variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to V is a *state*. I is the *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*. Each action $a \in A$ is a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a)\rangle$ where the *precondition* $\mathsf{pre}(a)$ and *effect* $\mathsf{eff}(a)$ are partial assignments to V, and $\mathsf{cost}(a) \in \mathbb{R}^{0+}$ is the action's non-negative *cost*.

For a partial assignment p, $\mathcal{V}(p) \subseteq V$ denotes the subset of state variables instantiated by p. For any $V' \subseteq \mathcal{V}(p)$, by $p[V']$ we denote the value of V' in p. An action a is *applicable* in a state s if $\mathsf{pre}(a) \subseteq s$, i.e., if $s[v] = \mathsf{pre}(a)[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. Applying a in s changes the value of each $v \in \mathcal{V}(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$, and leaves s unchanged elsewhere; the outcome state is denoted $s[\![a]\!]$. The outcome state of applying a sequence of (respectively applicable) actions is denoted $s[\![\langle a_1, \dots, a_n\rangle]\!]$. A *plan* for Π is an action sequence s.t. $G \subseteq I[\![\langle a_1, \dots, a_n\rangle]\!]$. The plan is *optimal* if its summed-up cost is minimal among all plans for Π.

The *causal graph* of a planning task captures state variable dependencies incurred by co-occurrences in actions (e.g. (Knoblock 1994; Jonsson and Bäckström 1995; Brafman and Domshlak 2003; Helmert 2006)). We use the commonly employed definition in the FDR context, where the causal graph *CG* is a directed graph over vertices V, with an arc from v to v', which we denote $(v \rightarrow v')$, if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$.

Fork Factoring

Our definitions of factorings and forks follow those by Brafman and Domshlak (2006) and Katz and Domshlak (2008):

Definition 1 (Fork Factoring) *Let* Π *be an FDR task with variables* V. *A* factoring $\mathcal{F}$ *is a partition of* V *into non-empty subsets* F, *called* factors. *The* interaction graph $IG(\mathcal{F})$ *of* $\mathcal{F}$ *is the directed graph whose vertices are the factors, and with an arc* $(F \rightarrow F')$ *if* $F \neq F'$ *and there exist* $v \in F$ *and* $v' \in F'$ *such that* $(v \rightarrow v')$ *is an arc in CG.*

$\mathcal{F}$ *is a* fork factoring *if* $|\mathcal{F}| > 1$ *and there exists* $F^R \in \mathcal{F}$ *s.t. the arcs in* $IG(\mathcal{F})$ *are exactly* $\{(F^R \rightarrow F^L) \mid F^L \in \mathcal{F} \setminus \{F^R\}\}$. F^R *is the* root *of* $\mathcal{F}$, *and all other factors* $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^R\}$ *are* leaves. *We also consider the* trivial fork factoring *where* $F^R = V$ *and* $\mathcal{F}^L = \emptyset$, *and the* pathological fork factoring *where* $F^R = \emptyset$ *and* $\mathcal{F}^L = \{V\}$.

The only global interactions in a fork factoring consist in *the root factor establishing preconditions for actions moving individual leaf factors*. But when do fork factorings exist, and how to find one? Denote by $\mathcal{F}^{\text{SCC}}$ the factoring whose factors are the strongly connected components of *CG*. Clearly, any fork factoring $\mathcal{F}$ must be coarser than $\mathcal{F}^{\text{SCC}}$, i.e., for every $F \in \mathcal{F}^{\text{SCC}}$ we must have $F' \in \mathcal{F}$ with $F \subseteq F'$. In other words, we must perform the factoring at the level of SCCs in the causal graph. In particular, if

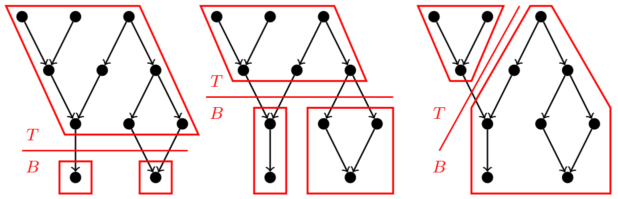
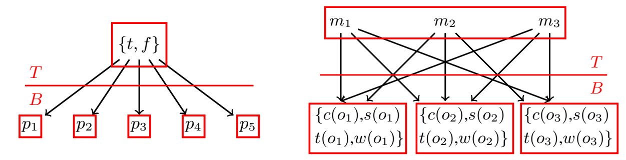


Figure 1: Examples of "horizontal lines" $\{T, B\}$ through a DAG of SCCs, and the corresponding fork factorings.

$|\mathcal{F}^{\text{SCC}}| = 1$ (the causal graph consists of a single SCC), then the only fork factorings are the trivial and pathological ones.

For the case where $|\mathcal{F}^{\text{SCC}}| > 1$, consider Figure 1. We view the interaction graph $IG(\mathcal{F}^{\text{SCC}})$ over SCCs as a DAG where the root SCCs are at the top and the leaf SCCs at the bottom. Let $\{T, B\}$ (top, bottom) be a "horizontal line" through that DAG, i.e., a partition of V where every $F \in \mathcal{F}^{\text{SCC}}$ is fully contained in either of T or B, and where the only arc in $IG(\{T, B\})$ is $(T \rightarrow B)$. Let $\mathcal{W}$ be the set of weakly connected components of $\mathcal{F}^{\text{SCC}}$ within B. Then we obtain a fork factoring $\mathcal{F}$ by setting $F^R := T$ and $\mathcal{F}^L := \mathcal{W}$. Vice versa, any non-trivial fork factoring can be obtained in this manner, except the *redundant* ones where some $F^L \in \mathcal{F}^L$ contains several weakly connected components from $\mathcal{W}$. Thus finding a (non-redundant) fork factoring is equivalent to finding the "horizontal line" $\{T, B\}$.

Our concrete strategy for finding that line is quite simple, and will be described in the experiments section. The strategy tries to obtain a factoring with many small leaf factors, which is when our method typically works well: a lot of conditional independence, not much overhead. In particular, if we don't find a factoring with at least two leaves (which we will know after negligible runtime), we *abstain* from solving the input task. We can then pass the task on to standard techniques (or, in the long term, to other target profiles).


$\{t, f\}$
T
B
p_1
p_2
p_3
p_4
p_5
m_1
m_2
m_3
T
B
$\{c(o_1),s(o_1)$ $t(o_1),w(o_1)\}$
$\{c(o_2),s(o_2)$ $t(o_2),w(o_2)\}$
$\{c(o_3),s(o_3)$ $t(o_3),w(o_3)\}$
Figure 2: Possible fork factorings in transportation with fuel consumption (left), and job-planning problems (right).

To illustrate the kind of structure we can exploit, consider Figure 2. On the left (as in IPC NoMystery), a truck t with fuel supply f transports packages $p_1, \dots, p_n$; t and f form an SCC in the causal graph, each p_i is a leaf SCC on its own. On the right (similar to IPC Woodworking), individual objects o_i are mutually independent except for sharing the machines, so that each leaf SCC contains the properties pertaining to one o_i. In both cases, our factoring strategy will come up with the fork factoring shown in the figure.

Fork-Decoupled State Spaces

We assume an FDR task $\Pi = \langle V, A, I, G\rangle$ and a fork factoring $\mathcal{F}$ with root F^R and leaves $F^L \in \mathcal{F}^L$. We refer to the actions A^R affecting the root as *root actions*, notation convention a^R, and to the actions A^L affecting $V \setminus F^R$ (i.e. any of the leaves) as *leaf actions*, notation convention a^L. For the set of actions affecting one particular $F^L \in \mathcal{F}^L$, we write $A^L|_{F^L}$. Bear in mind that the root actions A^R have preconditions and effects only on F^R, and the leaf actions $A^L|_{F^L}$ have preconditions only on $F^R \cup F^L$, and effects only on F^L. In particular, the sets A^R and $A^L|_{F^L}$ form a partition of the original action set A. A *root path* is a sequence of root actions applicable to I; a *leaf path* is a sequence of leaf actions applicable to I when ignoring preconditions on the root. Value assignments to F^R are *root states*, notated s^R, and value assignments to any $F^L \in \mathcal{F}^L$ are *leaf states*, notated s^L. For the leaf states of one particular $F^L \in \mathcal{F}^L$, we write $S^L|_{F^L}$, and for the set of all leaf states we write S^L. A root state s^R is a *goal root state* if $s^R \supseteq G[F^R]$, and a leaf state $s^L \in S^L|_{F^L}$ is a *goal leaf state* if $s^L \supseteq G[F^L]$. Finally, for a root state s^R, $A^L|_{s^R}$ denotes the leaf actions enabled by s^R, i. e., those where $\mathsf{pre}(a^L)[F^R] \subseteq s^R$.

The fork-decoupled state space augments root paths with maintenance of cheapest compliant leaf paths. A leaf path $\pi^L = \langle a_1^L, \dots, a_n^L \rangle$ *complies* with a root path π^R (also: is π^R*-compliant*) if we can schedule the a_i^L at monotonically increasing points alongside π^R so that each a_i^L is enabled in the respective root state.[1] Our states store the F^R part as usual, and for each leaf state s^L store the *price* we would have to pay for a cheapest compliant leaf path achieving s^L:

Definition 2 (Fork-Decoupled State Space) *A* fork-decoupled state s *is a pair* $\langle \mathsf{rt}(s), \mathsf{prices}(s) \rangle$ *with* $\mathsf{rt}(s)$ *being a root state, and* $\mathsf{prices}(s) : S^L \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ *being a function which assigns every leaf state a non-negative* price. *The* fork-decoupled state space *is a labeled transition system* $\Theta^{\mathcal{F}} = \langle S^{\mathcal{F}}, A^R, T^{\mathcal{F}}, I^{\mathcal{F}}, S_G^{\mathcal{F}} \rangle$ *as follows:*

(i) $S^{\mathcal{F}}$ *is the set of all fork-decoupled states* s.

(ii) A^R*, the set of root actions, gives the transition labels.*

(iii) $T^{\mathcal{F}}$ *is the set of transitions, with* $(s \xrightarrow{a} t) \in T^{\mathcal{F}}$ *if* $a \in A^R$, $\mathsf{rt}(s)[\![a]\!] = \mathsf{rt}(t)$*, and, for every leaf state* t^L, $\mathsf{prices}(t)[t^L] = \min_{s^L}(\mathsf{prices}(s)[s^L] + c(s^L))$ *where* $c(s^L)$ *is the cost of a cheapest path of* $A^L|_{\mathsf{rt}(t)}$ *actions from* s^L *to* t^L *if such a path exists, else* $c(s^L) := \infty$.

(iv) $I^{\mathcal{F}}$ *is the* fork-decoupled initial state*, where* $\mathsf{rt}(I^{\mathcal{F}}) := I[F^R]$*, and* $\mathsf{prices}(I^{\mathcal{F}})[s^L]$*, for every leaf state* $s^L \in S^L|_{F^L}$ *for some* $F^L \in \mathcal{F}^L$*, is the cost of a cheapest path of* $A^L|_{\mathsf{rt}(I^{\mathcal{F}})}$ *actions from* $I[F^L]$ *to* s^L *if such a path exists, else* $\mathsf{prices}(I^{\mathcal{F}})[s^L] := \infty$.

(v) $S_G^{\mathcal{F}}$ *are the* fork-decoupled goal states s_G*, where* $\mathsf{rt}(s_G)$ *is a goal root state and, for every* $F^L \in \mathcal{F}^L$*, there exists a goal leaf state* $s^L \in S^L|_{F^L}$ *s.t.* $\mathsf{prices}(s_G)[s^L] < \infty$.

The cost *of a path* $\pi^{\mathcal{F}} = s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_n$ *in* $\Theta^{\mathcal{F}}$ *is* $\mathsf{cost}(\pi^{\mathcal{F}}) := \sum_{i=1}^{n} \mathsf{cost}(a_i)$*. A* fork-decoupled plan *for* Π *is a path* $\pi^{\mathcal{F}}$ *from* $I^{\mathcal{F}}$ *to an element of* $S_G^{\mathcal{F}}$.

Note that, as stated, $S^{\mathcal{F}}$ is infinite. The *reachable* part of $S^{\mathcal{F}}$, however, is finite because it corresponds to root paths leading to different end states and/or pricing functions, where *prices decrease monotonically along any* $\Theta^{\mathcal{F}}$ *path*. The latter is because $\mathsf{prices}(s)[t^L]$ forms part of the minimization in (iii), namely when setting $s^L := t^L$ and using the empty path of $A^L|_{\mathsf{rt}(t)}$ actions.

Intuitively, $\Theta^{\mathcal{F}}$ encapsulates the idea of searching only in the root part, keeping track of what the leaves *could* do via the pricing functions, and selecting the *actually used* leaf paths as needed once we can achieve the goal. Observe in particular that the cost $\mathsf{cost}(\pi^{\mathcal{F}})$ of a fork-decoupled plan $\pi^{\mathcal{F}}$ accounts only for the root moves, not for the leaves. In that sense, fork-decoupled goal states s_G are *goal states with price tags*: We still have to "buy" the leaf goals, i. e., for each leaf factor F^L, we must pay $\mathsf{prices}(s_G)[s^L]$ for a cheapest goal leaf state $s^L \in S^L|_{F^L}$. We will get back to this below; for now let us illustrate Definition 2 with an example.

Say we have a transportation task with one package p, and two trucks t_A, t_B moving along three locations l_1, l_2, l_3 arranged in a line. Initially, $t_A = l_1$, $t_B = l_3$, and $p = l_1$. The goal is $p = l_3$. The actions are the usual truck moves and load/unload, with uniform costs. Our fork factoring $\mathcal{F}$ has $F^R = \{t_A, t_B\}$ and the only leaf $F^L = \{p\}$. Definition 2 (i): The fork-decoupled states $S^{\mathcal{F}}$ are assignments to $\{t_A, t_B\}$ along with a price for every leaf state, i. e., for every value of the leaf variable p. Definition 2 (ii): The transition labels are the truck moves. Definition 2 (iv): The fork-decoupled initial state $I^{\mathcal{F}}$ has $\mathsf{rt}(I^{\mathcal{F}}) = \{t_A = l_1, t_B = l_3\}$; $\mathsf{prices}(I^{\mathcal{F}})$ assigns 0 to $p = l_1$, 1 to $p = A$, and ∞ to all other leaf states, because $A^L|_{\mathsf{rt}(I^{\mathcal{F}})}$ contains exactly the load/unload actions for t_A at l_1, and for t_B at l_3. Note here that the "price" of $p = A$ is *not* a cost we have already spent. It is the cost we *would* have to spend if we decided to "buy" $p = A$ in $I^{\mathcal{F}}$, i. e., if we decided to augment the empty root path with a compliant leaf path achieving $p = A$.

For Definition 2 (iii) and (v), let us consider some transitions in $\Theta^{\mathcal{F}}$. Say we obtain s_1 from $I^{\mathcal{F}}$ by moving t_A to l_2. Then $\mathsf{rt}(s_1) = \{t_A = l_2, t_B = l_3\}$. The price tags $\mathsf{prices}(s_1)$ are still 0 for $p = l_1$ and 1 for $p = A$, as these were the prices in $I^{\mathcal{F}}$, no cheaper compliant paths become available in s_1, and $\mathsf{prices}(s)[t^L]$ forms part of the minimization in (iii) as pointed out above. In other words, the best option for these leaf states is to *not* move at s_1, instead using the compliant paths that were already available at $I^{\mathcal{F}}$. The only change in $\mathsf{prices}(s_1)$ is that $p = l_2$ gets price 2, accounting for "unload(p, t_A, l_2)" which is compliant at s_1 (but not at $I^{\mathcal{F}}$). Now say we obtain s_2 from s_1 by moving t_B to l_2, and s_3 from s_2 by moving t_B back to l_3. Then s_3 is a fork-decoupled goal state, because $\mathsf{prices}(s_2)[p = t_B] = 3$ and $\mathsf{prices}(s_3)[p = l_3] = 4$. Tracing back the compliant leaf path supporting $p = l_3$, we get the following 7-step global plan: load p onto t_A, move t_A to l_2, unload p, move t_B to l_2, load p onto t_2, move t_B to l_3, unload p.

Finally, say we obtain s_4 by moving t_A to l_3 in s_3. The compliant leaf path supporting $p = l_3$ in s_3 to yield the price tag 4 is then superseded by the new path "load(p, t_A, l_1), unload(p, t_A, l_3)" yielding $\mathsf{prices}(s_4)[p = l_3] = 2$. Observe that we now get a 6-step global plan, i. e., a plan *better* than that of the fork-decoupled goal state s_3 we passed through on the way to s_4. This is because, as indicated, *fork-decoupled goal states have price tags*. The price tag of s_4 is

[1]That is, denoting by $\langle s_0^R, \dots, s_m^R \rangle$ the root states traversed by π^R, there is a monotonically increasing function $f : \{1, \dots, n\} \mapsto \{0, \dots, m\}$ s.t., for every $i \in \{1, \dots, n\}$, $a_i^L \in A^L|_{s_{f(i)}^R}$.

cheaper than that of s_3, hence continuing search behind the goal state s_3 leads to a strictly better solution. We will see later how to obtain a standard structure for search.

To show correctness of the fork-decoupled state space relative to the original planning task, the central observation is:

Lemma 1 *For every root path π^R, denoting by s the fork-decoupled state reached by π^R in $\Theta^{\mathcal{F}}$, and for every leaf state s^L, $\mathsf{prices}(s)[s^L]$ is exactly the cost of a cheapest leaf path π^L that complies with π^R and achieves s^L.*

Proof Sketch: Showing this by induction, it is clearly true in $I^{\mathcal{F}}$. Presuming it is true in s, let $(s \xrightarrow{a} t) \in T^{\mathcal{F}}$ and let t^L be a leaf state. Observe that any cheapest leaf path $\pi^L[t]$ compliant at t and achieving t^L must consist of two parts, a cheapest leaf path $\pi^L[s]$ compliant at s achieving some leaf state s^L, and a cheapest path of $A^L|_{\mathsf{rt}(t)}$ actions from s^L to t^L. By induction, $\mathsf{prices}(s)[s^L]$ equals the cost of $\pi^L[s]$. By construction (Definition 2 (iii)), this shows the claim. $\square$

To formalize the "goal state price tag", recall that each leaf factor may be the cross-product of several state variables, and hence may have more than one goal leaf state. So we need to minimize over these leaf goal states:

Definition 3 (Goal Price) *The* goal price *of $s_G \in S^{\mathcal{F}}_G$ is $\mathsf{Gprice}(s_G) := \sum_{F^L \in \mathcal{F}^L} \min\{\mathsf{prices}(s_G)[s^L] \mid s^L \in S^L|_{F^L}, s^L \supseteq G[F^L]\}$. The* global cost *of a $\Theta^{\mathcal{F}}$ path $\pi^{\mathcal{F}}$ ending in $s_G \in S^{\mathcal{F}}_G$ is $\mathsf{GlobalCost}(\pi^{\mathcal{F}}) := \mathsf{cost}(\pi^{\mathcal{F}}) + \mathsf{Gprice}(s_G)$. A fork-decoupled plan is* fork-optimal *if its global cost is minimal among all fork-decoupled plans.*

For a fork-decoupled plan $\pi^{\mathcal{F}}$ ending in $s_G \in S^{\mathcal{F}}_G$, define $\mathsf{GlobalPlan}(\pi^{\mathcal{F}})$ as $\pi^{\mathcal{F}}$ augmented, for every $F^L \in \mathcal{F}^L$, with a cheapest compliant leaf path π^L achieving $\arg\min\{\mathsf{prices}(s_G)[s^L] \mid s^L \in S^L|_{F^L}, s^L \supseteq G[F^L]\}$. Vice versa, given a plan π for Π, let $\mathsf{ForkPlan}(\pi)$ be the $\Theta^{\mathcal{F}}$ path $\pi^{\mathcal{F}}$ induced by the root actions in π. We have:

Theorem 1 *For any (fork-optimal) fork-decoupled plan $\pi^{\mathcal{F}}$ for Π, $\mathsf{GlobalPlan}(\pi^{\mathcal{F}})$ is an (optimal) plan for Π. Vice versa, for any (optimal) plan π for Π, $\mathsf{ForkPlan}(\pi)$ is a (fork-optimal) fork-decoupled plan for Π.*

Proof Sketch: As $\mathcal{F}$ is a fork factoring, any plan for Π can be perceived as a root path π^R achieving the goal for F^R, augmented for every leaf F^L with a compliant leaf path π^L achieving $G[F^L]$. The claim follows with Lemma 1. $\square$

What can we say about the *size* of $\Theta^{\mathcal{F}}$? Consider the reachable states in $\Theta^{\mathcal{F}}$, denoted $S^{\mathcal{F}}_R$, and those in the standard state space, denoted S_R. A fork-decoupled state $s \in S^{\mathcal{F}}_R$ can be viewed as a "condensed" set of (priced) world states $w \in S_R$, namely all those w consisting of the same root state and, for each leaf factor, any leaf state s^L reachable in s i. e. where $\mathsf{prices}(s)[s^L] < \infty$. Condensing world states in this way can dramatically reduce state space size. A pathological case is the pathological fork factoring (hence the name), where the entire task is a single "leaf" and $|S^{\mathcal{F}}_R| = 1$. Genuinely good cases have exponential reductions at no blow-up in leaf size. For example, if a single truck (root) moves n packages (leaves) on a map with m locations, and all packages and the truck start in the same location, then $|S_R| = m * (m+1)^n$, whereas $|S^{\mathcal{F}}_R| = \frac{m(m+1)}{2}$ because the leaf state prizes are the same for each package as a function of the truck moves executed. However, there also are bad cases where $S^{\mathcal{F}}_R$ enumerates combinations of leaf states for the same leaf, and $S^{\mathcal{F}}_R$ is exponentially larger than S_R. Our examples showing this are contrived though. On IPC benchmarks, with our current factoring strategy, $S^{\mathcal{F}}_R$ is never larger than S_R, and is typically *much* smaller. The curious reader is invited to forward to Table 1 for a moment.

Heuristic Functions

Two *different kinds* of heuristic functions are of interest:

Definition 4 (Heuristics) *Let Π be an FDR task, and $\mathcal{F}$ a fork factoring. A* heuristic *is a function $h : S^{\mathcal{F}} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$. The* root-perfect *heuristic, h^{R*}, is that where $h^{R*}(s)$ for any s is the minimum over $\mathsf{cost}(\pi^{\mathcal{F}})$ for all $\Theta^{\mathcal{F}}$ paths $\pi^{\mathcal{F}}$ from s to some $s_G \in S^{\mathcal{F}}_G$. The* fork-perfect *heuristic, $h^{\mathcal{F}*}$, is that where $h^{\mathcal{F}*}(s)$ for any s is the minimum over $\mathsf{GlobalCost}(\pi^{\mathcal{F}})$ for all such paths. We say that h is* root-admissible *if $h \leq h^{R*}$, and* fork-admissible *if $h \leq h^{\mathcal{F}*}$.*

The conceptual distinction between h^{R*} and $h^{\mathcal{F}*}$ lies in that h^{R*} cares only about how much work is left for the root factor, i. e., the cost of a root path sufficient to enable every leaf to reach its goal *somehow*. In contrast, $h^{\mathcal{F}*}$ accounts for the combined cost of root and leaves, i. e. for the best extension of our current root path into an overall fork-decoupled plan (and thus a plan for the original input task Π, cf. Theorem 1). We will refer to heuristics attempting to estimate h^{R*} as *root heuristics*, and to heuristics attempting to estimate $h^{\mathcal{F}*}$ as *fork heuristics*, and distinguish them notationally by superscripts "R" respectively "$\mathcal{F}$".

Observe that $h^{\mathcal{F}*}$ is notably different from typical remaining cost estimators (including, e. g., h^{R*}) due to the goal state price tags, captured in Definition 4 by the goal price component of $\mathsf{GlobalCost}(\pi^{\mathcal{F}})$. These price tags account for the entire cost of moving the leaves, including actions that will be scheduled *before* (and up to) the state s in question. In particular, $h^{\mathcal{F}*}$ is *not* 0 on fork-decoupled goal states: we still have to pay the goal prize, moving the leaves into place.

Observe furthermore that h^{R*} is a special case of $h^{\mathcal{F}*}$: We can compute h^{R*} as $h^{\mathcal{F}*}$ in a modified planning task where the cost of all leaf actions is set to 0. h^{R*} keeps track only of which leaf states are reachable, not of the associated cost. This may lead to qualitatively different decisions, i. e., h^{R*} and $h^{\mathcal{F}*}$ may *disagree*. Using a transportation example again, say that there are two alternative kinds of plans, (a) ones that pass the packages through several trucks, loading/unloading every time, vs. (b) ones that make more truck moves but have to load/unload each package only once and thus are better globally. Then h^{R*} will draw search towards plans (a), whereas $h^{\mathcal{F}*}$ will draw search towards plans (b).

Turning our attention to the computation of practical heuristics, we consider $h^{\mathcal{F}*}$ first. Note that, to estimate $h^{\mathcal{F}*}$, we have to account for the option to schedule leaf actions *before* (and up to) the state s in question. This is both, a threat to informativity (we cannot simply discount the cost of such actions) and to admissibility (we cannot simply pretend that this is not possible). It turns out that both can be tackled using the pricing information maintained in $\Theta^{\mathcal{F}}$:

Definition 5 (From Classical h to $h^{\mathcal{F}*}$ Estimation) *Let h be any FDR planning heuristic, $\Pi = \langle V, A, I, G\rangle$ an FDR task, $\mathcal{F}$ a fork factoring, and s a fork-decoupled state. The* leaves-for-pay *fork heuristic estimate is $h^{\mathcal{F}}_{L\$}(s) := h(s_{L\$})$ using the FDR task $\Pi_{L\$} = \langle V, A_{L\$}, s_{L\$}, G\rangle$ obtained from Π by setting $s_{L\$} := \mathsf{rt}(s) \cup I[V \setminus F^R]$, and including for every leaf state $s^L \in S^L|_{F^L}$ where $s^L \neq I[F^L]$ and $\mathsf{prices}(s)[s^L] < \infty$ a new leaf action $a[s^L]$ with empty precondition, effect s^L, and cost $\mathsf{cost}(a[s^L]) := \mathsf{prices}(s)[s^L]$.*

This construction caters for "leaf actions to be scheduled before and up to s", but not by allowing to explicitly insert such actions. We instead allow the planner on $\Pi_{L\$}$ to pay the prize that *would* be incurred by doing so, using the new actions $a[s^L]$. This simple approach preserves admissibility, and preserves informativity in a basic sense:

Theorem 2 *If h is admissible, then $h^{\mathcal{F}}_{L\$}$ is fork-admissible. If $h = h^*$, then $h^{\mathcal{F}}_{L\$} = h^{\mathcal{F}*}$.*

Proof Sketch: Let $s \in S^{\mathcal{F}}$. Any $\Theta^{\mathcal{F}}$ path $\pi^{\mathcal{F}}$ from s to $s_G \in S^{\mathcal{F}}_G$ corresponds to a plan for $\Pi_{L\$}$ of cost $\mathsf{GlobalCost}(\pi^{\mathcal{F}})$. Vice versa, as $\mathcal{F}$ is a fork factoring for $\Pi_{L\$}$, any optimal plan π for $\Pi_{L\$}$ uses cheapest compliant leaf paths, hence corresponds to a $\Theta^{\mathcal{F}}$ path $\pi^{\mathcal{F}}$ from s to $s_G \in S^{\mathcal{F}}_G$ where $\mathsf{GlobalCost}(\pi^{\mathcal{F}})$ equals the cost of π. Hence $h^*(s_{L\$}) = h^{\mathcal{F}*}(s)$, from which the claims follow. □

We now get an h^{R*} estimate as a corollary via the aforementioned relation to $h^{\mathcal{F}*}$:

Definition 6 (From Classical h to h^{R*} Estimation) *Let h be any FDR planning heuristic, $\Pi = \langle V, A, I, G\rangle$ an FDR task, $\mathcal{F}$ a fork factoring, and s a fork-decoupled state. The* leaves-for-free *root heuristic estimate is $h^R_{L0\$}(s) := h(s_{L0\$})$ in the FDR task $\Pi_{L0\$} = \langle V, A_{L0\$}, s_{L0\$}, G\rangle$ identical to $\Pi_{L\$}$ as in Definition 5, except that we set $\mathsf{cost}(a^L) := 0$ for all leaf actions $a^L \in A^L$, and define $\mathsf{cost}(a[s^L]) := 0$.*

Theorem 3 *If h is admissible, then $h^R_{L0\$}$ is root-admissible. If $h = h^*$, then $h^R_{L0\$} = h^{R*}$.*

Proof Sketch: This follows from Theorem 2 and as h^{R*} equals $h^{\mathcal{F}*}$ when setting the cost of all leaf actions to 0. □

Search Algorithms

Disregarding optimality, we can run any search algorithm on the fork-decoupled state space, stopping at the first fork-decoupled goal state. For optimal planning, matters are more subtle. One of our methods is formulated on a modified state space where the goal price tags are explicit:

Definition 7 (Explicit Goal Prices) *Let Π be an FDR task, $\mathcal{F}$ a fork factoring, and $\Theta^{\mathcal{F}} = \langle S^{\mathcal{F}}, A^R, T^{\mathcal{F}}, I^{\mathcal{F}}, S^{\mathcal{F}}_G\rangle$ the fork-decoupled state space. The* explicit goal-price *state space $\Theta^{\mathcal{F}}_{G\$}$ is like $\Theta^{\mathcal{F}}$ but with a new state G' and, for every $s_G \in S^{\mathcal{F}}_G$, a new transition $s_G \rightarrow G'$ with cost $\mathsf{Gprice}(s_G)$; the set of goal states is set to be $\{G'\}$.*

The explicit goal-price state space is useful because it allows us to formulate search on $\Theta^{\mathcal{F}}$, which is non-standard due to the goal state price tags, in terms of standard search on $\Theta^{\mathcal{F}}_{G\$}$, via this correspondence: (direct from construction)

```
Algorithm Fork-Decoupled X (FDX):
  Input: FDR planning task $\Pi$, fork factoring $\mathcal{F}$
         Heuristic search algorithm $X$
         Fork heuristic $h^{\mathcal{F}}$
  Output: A plan for $\Pi$, or "failed"
Let $h^{\mathcal{F}}_{G\$} := \begin{cases} h^{\mathcal{F}}(s) & s \in S^{\mathcal{F}} \\ 0 & s = G' \end{cases}$
Run $X$ with $h^{\mathcal{F}}_{G\$}$ on $\Theta^{\mathcal{F}}_{G\$}$
If $X$ found a solution path $\pi = \pi^{\mathcal{F}} \circ \langle s_G \rightarrow G'\rangle$
  return $\mathsf{GlobalPlan}(\pi^{\mathcal{F}})$
else return "failed"
```

Figure 3: Exploiting any known search algorithm X.

Lemma 2 *The solution paths in $\Theta^{\mathcal{F}}_{G\$}$ are in one-to-one correspondence with the fork-decoupled plans for Π.*

Consider Figure 3. FDX just runs any search algorithm X on $\Theta^{\mathcal{F}}_{G\$}$. We have:

Theorem 4 *If X is complete, then FDX is complete. If X is optimal for admissible heuristics, then FDX is optimal for fork-admissible heuristics.*

Proof: Completeness follows directly from Lemma 2 and Theorem 1. Optimality follows in the same manner, as, for fork-admissible $h^{\mathcal{F}}$, $h^{\mathcal{F}}_{G\$}$ is admissible in $\Theta^{\mathcal{F}}_{G\$}$. □

Consider now Figure 4. AFRA* guides A* by a *root heuristic*, and uses a fork heuristic merely for upper-bound pruning. The search is drawn to cheap root paths, disregarding leaf costs, so to guarantee optimality we must exhaust the open list. Without early termination, this would be dominated by FDA* because AFRA* would then have to expand at least all $N[s]$ where $g(N) + h^{\mathcal{F}}(s)$ is less than optimal solution cost. *With* early termination, that is not so because in the best case we have to exhaust only those $N[s]$ where $g(N) + h^R(s)$ is less than optimal *root* solution cost.

Theorem 5 *AFRA* is complete, and is optimal for root-admissible h^R and fork-admissible $h^{\mathcal{F}}$. FRA* is optimal for root-admissible h^R.*

Proof Sketch: Completeness of AFRA* is obvious with Theorem 1. To see optimality of AFRA*, say that early termination fires for $N[s_G]$, let $\pi^{\mathcal{F}}_1$ be the fork-decoupled plan leading to N, and let $\pi^{\mathcal{F}}_2$ be any fork-decoupled plan that would be generated later if we continued the search. As $\mathsf{Gprice}(s_G) = \mathsf{MINGprice}$, $\pi^{\mathcal{F}}_2$ pays at least as high a goal price as $\pi^{\mathcal{F}}_1$. For root-admissible h^R, because we use A*, $\mathsf{cost}(\pi^{\mathcal{F}}_1) \leq cost(\pi^{\mathcal{F}}_2)$. So $\pi^{\mathcal{F}}_2$ cannot have better global cost than $\pi^{\mathcal{F}}_1$. But then, as AFRA* generates every fork-optimal fork-decoupled plan eventually, it must already have found such a plan, which must be stored in $\pi^{\mathcal{F}}_U$. Optimality of FRA* is a special case. □

FRA* is not complete because we force it to prove optimality. In practice, one could return the plan anyway, flagging it as "proved optimal" only if $\mathsf{Gprice}(s_G) = \mathsf{MINGprice}$. From a theory perspective, being *optimal but not complete* is a rather unique profile. It can be viewed as an effect of target-profile factoring, which is intrinsically geared at special-case structure (in our case, the leaf costs).

```
**Algorithm** *Anytime Fork-Root A$^*$(AFRA$^*$)*:
  **Input:** FDR planning task $\Pi$, fork factoring $\mathcal{F}$
       Root heuristic $h^R$, fork heuristic $h^{\mathcal{F}}$
  **Output:** An optimal plan for $\Pi$, or "unsolvable"
Let $U := \infty$ /* best known upper bound */
Let $\pi_U^{\mathcal{F}} := \bot$ /* corresponding plan */
Run A$^*$ with $h^R$ on $\Theta^{\mathcal{F}}$, with these modifications:
   Continue search until the open list is empty
   Whenever a goal vertex node $N[s_G]$ is expanded:
      If $g(N) + \mathsf{Gprice}(s_G) < U$
         let $U := g(N) + \mathsf{Gprice}(s_G)$
         let $\pi_U^{\mathcal{F}}$ := the fork-decoupled plan leading to $N$
      If $\mathsf{Gprice}(s_G) = \mathsf{MINGprice}$
         return $\mathsf{GlobalPlan}(\pi_U^{\mathcal{F}})$ /* early termination */
   Whenever a node $N[s]$ is generated, and $U \neq \infty$:
      If $g(N) + h^{\mathcal{F}}(s) \geq U$
         discard $N$ /* upper-bound pruning */
If $\pi_U^{\mathcal{F}} \neq \bot$ return $\mathsf{GlobalPlan}(\pi_U^{\mathcal{F}})$ else return "unsolvable"
**Algorithm** *Fork-Root A$^*$(FRA$^*$)*:
  **Input:** FDR planning task $\Pi$, fork factoring $\mathcal{F}$
       Root heuristic $h^R$
  **Output:** An optimal plan for $\Pi$, or "sorry no luck"
Run AFRA$^*$ with $h^R$ on $\Theta^{\mathcal{F}}$, with these modifications:
   Do not use upper-bound pruning
   When the first goal vertex node $N[s_G]$ is expanded, do:
      If $\mathsf{Gprice}(s_G) = \mathsf{MINGprice}$
         let $\pi^{\mathcal{F}}$ := the fork-decoupled plan leading to $N$
         return $\mathsf{GlobalPlan}(\pi^{\mathcal{F}})$
      else return "sorry no luck"
```

Figure 4: New search algorithms. Search nodes are notated $N[s]$ where s is the state and N the node itself. $\mathsf{MINGprice}$ is the sum, over the leaf factors $F^L \in \mathcal{F}^L$, of optimal plan cost for the *projection* of Π onto F^L.

Experiments

Our implementation is in FD (Helmert 2006). The pricing function is maintained enumeratively for each leaf factor. For plan extraction, each leaf state stores a backward pointer to its predecessor on a cheapest compliant path. We employ *subsumption pruning*, where s subsumes t if $\mathsf{rt}(s) = \mathsf{rt}(t)$ and, for every leaf state s^L, $\mathsf{prices}(s)[s^L] \leq \mathsf{prices}(t)[s^L]$. We implemented fork and root heuristics via Definitions 5 and 6 for $h^{\max}$, LM-cut, and h^{FF} (Bonet and Geffner 2001; Helmert and Domshlak 2009; Hoffmann and Nebel 2001). We also experiment with FD's "blind heuristic", returning 0 on goal states and $\min_{a \in A} \mathsf{cost}(a)$ elsewhere.

We ran all IPC STRIPS benchmarks (1998 – 2014), on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

Factoring Strategy

We proceed as follows. If the input task has causal graph leaf variables, define $\mathcal{F}_0$ as the fork factoring whose leaf factors are exactly those variables; else $\mathcal{F}_0$ is undefined. Initialize $D := \mathit{IG}(\mathcal{F}^{\mathrm{SCC}})$ to be the DAG interaction graph over the input task's SCCs. Set $B_0 := \emptyset$ and $i := 1$. Then iterate the following steps: Take the *candidates* to be those leaf SCCs L in D where, with $B'_L := B_{i-1} \cup L$ and $\mathcal{W}'_L$ being the weakly connected components within B'_L, all $W' \in \mathcal{W}'_L$ satisfy $\prod_{v \in W'} |\mathcal{D}(v)| \leq 2^{32}$. If there are no candidates, terminate the iteration. Else, select a candidate L with minimal $\prod_{v \in L} |\mathcal{D}(v)|$. Set $B_i := B'_L$, define $\mathcal{F}_i$ by $\mathcal{F}_i^R := V \setminus B_i$ and $\mathcal{F}_i^L := \mathcal{W}'_L$, remove L from D, increment i, and iterate. Upon termination, consider the set of defined $\mathcal{F}_i$. If this set is empty, or no factoring in this set has more than a single leaf, then we *abstain* from solving Π. Otherwise, we select the factoring $\mathcal{F}_i$ whose number of leaf factors is maximal, and whose index i is minimal among these factorings.

	#	Std	Fork	Std $\lvert S_R\rvert$	Fork $\lvert S_R^{\mathcal{F}}\rvert$	Nr	MaxSize	Nr	MaxSize
		State spaces				Leaf factors (Mean stats)			
		Built		Mean size (Common)		Common		All	
Driverlog	20	5	8	3005640.0	27385.2	3.4	5.0	8.4	12.2
Logistics	63	12	23	422507.6	2614.2	4.9	7.5	11.2	71.2
Miconic	145	45	30	48544.0	6572.3	4.5	4.0	16.0	4.0
NoMystery	40	11	26	3561168.0	1290.4	5.0	6.7	9.0	10.5
Pathways	29	3	3	1167551.7	236471.7	3.0	2.0	21.1	2.0
Rovers	40	5	5	3925444.8	138813.4	3.8	2.0	20.9	2.0
Satellite	36	4	4	170808.0	109268.8	4.8	2.0	54.8	2.0
TPP	24	5	11	10029509.4	21.6	4.0	7.8	8.3	954.4
Woodwork	61	9	9	8204333.2	122433.9	2.8	160.1	3.9	43.6
Zenotravel	20	7	7	1965822.0	41887.1	3.7	5.1	10.0	11.7

Table 1: Reachable state space size and leaf factor statistics, on IPC instances where our factoring method does not abstain (see text). Where there are several IPC instance suites of a domain, we take their union here. "Built": reachable state space could be fully built; "Common": those instances where both state spaces were built; "Nr": (mean) number of leaf factors; "MaxSize": (mean) maximum leaf factor size, where the "size" of $F^L \in \mathcal{F}^L$ is the number of reachable states in the projection of Π onto F^L.

In words, we run a greedy pass through the lattice of fork factorings, imposing for sanity that no leaf factor has estimated size greater than MAXINT (for 32 bits). We select the factoring with the maximal number of leaves, keeping these as small as possible. This maximizes the amount of conditional independence, while minimizing the overhead of maintaining pricing functions. It takes negligible runtime effort as the graph $\mathit{IG}(\mathcal{F}^{\mathrm{SCC}})$ is reliably very small. The runtime is (rounded to) 0.00 in 97% of all IPC STRIPS tasks, and is ≤ 0.1 seconds in all but a single task, where we take 0.19 seconds but FD's pre-process takes several minutes.

We also ran alternative strategies where we selected the *last* factoring generated when imposing a bound M on the sum $\sum_{F^L \in \mathcal{F}_i^L} \prod_{v \in F^L} |\mathcal{D}(v)|$ of estimated leaf sizes. In almost all domains, this either hardly changed the factoring at all, or changed it only for large M (1 million and more), and then only on the smallest instances where we obtained a single "leaf" containing most of the state space, see next.

We abstain from single-leaf factorings because these nearly always arise from a factoring grouping most of the state variables into the leaf. In transportation domains with capacity constraints, for example, the "horizontal line" is drawn between the vehicles and the entire rest of the task. Such factorings *can* sometimes be useful; e. g., in IPC'08-optimal Transport instance p04, using LM-cut evaluations go down from 95281 for A* to 17606 for FDA*, and runtime goes down from 56.8 to 17.6 seconds. But the huge single leaf is typically manageable only on the smallest instances of a domain. An interesting topic for future work is to handle large leaves using a symbolic, rather than enumerative, representation. For now, we concentrate on exploiting conditional independence, across > 1 leaves.

Consider Table 1. One might wonder whether the radical state space size reduction is pathological (if the entire task is a single "leaf" then $|S_R^{\mathcal{F}}| = 1$ but "MaxSize" $= |S_R|$), yet the right-hand side of the table shows that this is not so. The only two domains where leaves do get "large" are TPP and Woodworking, but compared to the state space reduction they are still tiny. Observe, though, that the state space reduction does not pay off in Miconic, where it does not outweigh the runtime/memory overhead and more standard state spaces are built. The reduction tends to be less if the leaves are "too small", as in Miconic, Rovers, Satellite, and Pathways. This makes sense as the leaves then cannot "do much on their own", so, relatively speaking, less world states are condensed into each single fork-decoupled state.

In all except the domains from Table 1, we abstain from every instance. While this may be construed as a weakness of fork factoring, arguably it is, at least as much, a strength. Almost all known planning/search techniques work well only on a limited domain subset. For example, state-of-the-art partial-order reduction (POR) (Wehrle and Helmert 2014), a long-standing technique into which intense effort was invested, improves coverage with LM-cut on 7 IPC domains, substantially (> 1 instance) on 5. These numbers are almost the same, actually slightly better (7 and 6, Table 2), for fork factoring. However, in difference to POR and basically all other known techniques, fork factoring *abstains* outside its scope: The structure it requires to work well is explicit and easily testable. This is perfect for portfolios (e. g. (Xu et al. 2008; Cenamor, de la Rosa, and Fernández 2014)), combining the strengths of *different* basic solvers, which is necessary for effective general off-the-shelf tools anyhow.

Planner Performance Results

For satisficing planning, the domains within our scope are often too easy for fork factoring to be beneficial. The most notable exception is transportation with fuel consumption. Denote with GBFS FD's lazy-greedy best-first search with a second open list for preferred operators. On the IPC'11 satisficing NoMystery benchmarks, while GBFS solves 9 instances with h^{FF}, FDGBFS solves 19. Figure 5 examines NoMystery more closely, on Nakhost et al.'s (2012) controlled benchmarks. We ran the respective best satisficing and optimal planners from Nakhost et al.'s experiments, as well as IPC'14 optimal-track winner *Symba* (Torralba and Alcázar 2013). FDGBFS solves *all* instances, vastly outperforming its satisficing competitors when resources are scarce. FDA* with LM-cut solves more than 90% of the instances in "small", vastly outperforming merge-and-shrink (M&S). It solves all instances in "large", in stark contrast to the optimal planners run by Nakhost et al., as well as M&S in our own runs here, none of which managed to solve *any* of these instances. Symba handles "large" much better, though clearly worse than FDA*; in "small", Symba has only 44% coverage in total (data not included for readability).

Table 2 gives full results for optimal planning. To clarify the competition: Almost all works on factored plan-

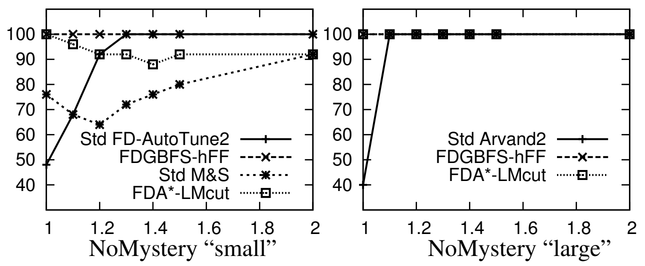


Figure 5: % instances covered in Nakhost et al.'s (2012) NoMystery test suites, over *resource constrainedness* (factor by which the available fuel exceeds the minimum required).

ning have notions of local- or step-optimality, and could be rendered globally optimal only by impractical searches over bounds on the number of (local) steps. Fabre et al. (2010) do guarantee global cost-optimality, but report themselves that their approach is impractical (except on manually reformulated versions of the IPC'04 Promela domains). We compare against partition pruning in A* (Nissim, Apsel, and Brafman 2012; Nissim and Brafman 2014), the only other work exploiting some form of decomposition in globally cost-optimal heuristic search. We compare against POR (Wehrle and Helmert 2012; Wehrle et al. 2013; Wehrle and Helmert 2014), in its best-performing (on our non-abstained benchmarks) variant, as a representation of the state of the art in admissible search space pruning.

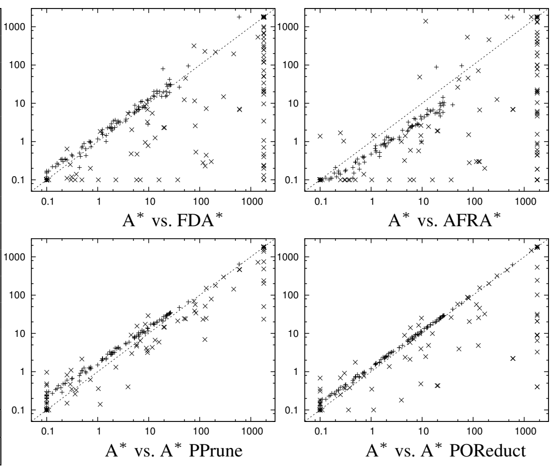
Fork-decoupled state space search clearly has the edge in overall coverage. For each of LM-cut, $h^{\max}$, and the blind heuristic, FDA* and AFRA* coverage dominates that of A* in all domains except Miconic (as well as 2 instances in Zenotravel for AFRA* with LM-cut). Our techniques yield amazing benefits in Logistics, NoMystery, and TPP; strong benefits in Woodworking and Rovers (the latter especially when using the alternative factoring strategy described above); and minor to significant benefits in various other cases. In Logistics, NoMystery, and TPP, as well as Rovers with the alternative factoring strategy, even $h^{\max}$ and blind search with fork-decoupled search have higher coverage than LM-cut with standard search. Indeed, *when ignoring the oversize Miconic domain,* $h^{\max}$ *and blind search outperform LM-cut in summed-up coverage* (109–112 vs. 97).

In almost all cases, as evidenced by the coverage of FRA*, the first plan found by AFRA* is proved optimal by early termination already. We need to continue search only sometimes in Driverlog, Woodworking, and Zenotravel.

The search space reduction obtained by fork decoupling is typically much larger than that for partition pruning and POR, although there are exceptions (Pathways, Satellite, Woodworking). Similarly for runtime. The runtime factors are sometimes skewed by the small size of commonly solved instances, as well as FD's rounding; e. g., in TPP, *all* runs of FDA* and AFRA* on commonly solved instances are rounded to 0.1. The scatter plots shed additional light on this, showing that FDA* and AFRA* are highly beneficial here, at a slightly higher risk in the case of AFRA*.

Of course, our observations must be qualified against the benchmark selection: fork factoring is great *within its scope*. Yet keep in mind that, outside this scope, we can after negligible runtime invoke whatever alternative planner we want.

		(A): Coverage																		
		A*			FDA*			AFRA*			FRA*			PPrune			POReduct			
	#	BL	HM	LM	BL	HM	LM	BL	HM	LM	BL	HM	LM	BL	HM	LM	BL	HM	LM	SB
Driver	20	7	9	13	*11*	*11*	13	*11*	*11*	13	*9*	*10*	10	7	9	13	7	9	13	**14**
Log00	28	10	12	20	*22*	*22*	**28**	*22*	*22*	*25*	*22*	*22*	*25*	*11*	12	20	10	12	20	19
Log98	35	2	2	**6**	*4*	*3*	**6**	*4*	*4*	**6**	*4*	*4*	**6**	*3*	*3*	**6**	2	*3*	**6**	5
Mico	145	50	50	**136**	36	36	135	36	36	135	36	36	135	45	45	**136**	45	45	**136**	102
NoMy	20	8	8	14	*17*	*18*	**20**	*17*	*19*	**20**	*17*	*18*	**20**	8	8	14	8	8	14	14
Pathw	29	3	3	4	3	3	4	3	3	4	3	3	4	3	3	4	3	3	4	**5**
Rovers	40	6	6	7	6	6	*8*	6	*7*	*9*	6	*7*	*9*	6	6	*10*	*7*	*8*	*9*	**14**
Satell	36	6	6	7	6	6	7	6	6	*8*	6	6	*8*	*7*	*7*	**12**	6	*7*	*11*	9
TPP	27	5	5	5	**23**	*22*	*18*	**23**	*22*	**23**	**23**	*22*	**23**	5	5	5	5	5	5	7
Woo08	13	4	5	6	*5*	*6*	*10*	*5*	*6*	**11**	1	2	*7*	4	5	6	*6*	*8*	**11**	**11**
Woo11	5	0	1	2	*1*	1	*4*	*1*	1	**5**	*1*	1	**5**	0	1	2	*1*	*3*	**5**	**5**
Zeno	20	8	8	**13**	*11*	*11*	**13**	*11*	*11*	11	*9*	*9*	8	8	8	**13**	8	8	**13**	11
$\sum$	418	109	115	233	*145*	*145*	*266*	*145*	*148*	**270**	*137*	*140*	*260*	107	112	*241*	108	*119*	*247*	216
Rovers	$\mathcal{F}$'				*9*	*9*	*9*	*9*	*9*	*9*	*7*	*7*	7							



	(B): Evaluations with LM-cut: Improvement factor relative to A*												
		FDA*			AFRA*			PPrune			POReduct		
	#	$\sum$D	Gmea	max	$\sum$D	Gmea	max	$\sum$D	Gmea	max	$\sum$D	Gmea	max
Driver	13	2.2	3.5	73.2	1.2	3.0	83.3	1.4	1.3	2.0	1.0	1.2	1.9
Log00	20	960.3	130.7	1407	562.3	106.5	1119	2.8	2.1	5.4	2.9	2.7	22.3
Log98	6	49.3	11.4	458.8	0.2	1.7	458.8	4.3	2.4	3.0	2.5	3.4	5.4
Mico	135	2.2	1.7	6.5	2.2	1.7	6.5	1.0	1.0	1.0	1.0	1.0	1.0
NoMy	14	20.4	8.8	22.2	8.4	5.0	9.3	1.0	1.0	1.0	1.0	1.0	1.0
Pathw	4	1.9	1.3	1.9	1.7	1.2	1.7	2.2	1.3	2.2	16.2	3.0	16.7
Rovers	7	4.8	2.5	8.9	7.8	3.4	15.5	2.4	1.6	3.2	4.4	2.3	19.9
Satell	7	1.0	1.1	1.4	2.2	1.6	5.1	16.3	3.6	16.9	21.5	5.6	27.8
TPP	5	2491	49.0	5657	2858	52.7	6869	1.1	0.9	1.6	1.0	1.0	1.0
Woo08	6	82.4	42.9	135.2	72.8	41.4	108.0	1.4	1.2	1.5	243.2	10.6	346.9
Woo11	2	82.4	55.4	135.2	72.8	49.6	108.0	1.4	1.4	1.5	250.3	173.7	346.9
Zeno	11	6.5	3.0	13.2	1.4	1.3	4.0	2.3	1.5	3.1	1.0	1.0	1.3

	(C): Runtime with LM-cut: Improvement factor relative to A*												
		FDA*			AFRA*			PPrune			POReduct		
	#	$\sum$D	Gmea	max	$\sum$D	Gmea	max	$\sum$D	Gmea	max	$\sum$D	Gmea	max
Driver	9	0.7	1.8	31.6	0.4	1.7	54.0	1.2	1.0	1.7	1.0	0.9	1.0
Log00	11	311.9	67.1	561.1	337.9	69.9	841.7	2.9	2.2	5.4	3.0	2.4	5.2
Log98	4	14.8	7.7	158.3	0.1	1.0	299.6	2.4	1.9	2.7	3.9	2.7	4.8
Mico	100	0.8	0.8	3.4	1.7	2.0	9.7	0.7	0.6	0.9	0.9	0.8	1.0
NoMy	12	17.9	4.5	19.7	20.5	4.8	23.4	0.9	0.4	1.0	0.8	0.4	0.9
Pathw	1	1.2	1.2	1.2	1.2	1.2	1.2	2.2	2.2	2.2	13.4	13.4	13.4
Rovers	3	1.9	2.1	2.9	4.3	4.3	6.5	2.4	2.3	3.0	3.5	4.5	19.6
Satell	3	0.6	0.6	0.7	1.8	1.8	3.9	17.6	11.3	18.2	25.7	18.6	26.2
TPP	1	68.2	68.2	68.2	68.2	68.2	68.2	1.1	1.1	1.1	1.0	1.0	1.0
Woo08	2	65.9	27.3	84.7	69.8	30.4	85.9	1.3	1.3	1.4	233.6	113.1	269.8
Woo11	2	68.7	27.5	89.1	71.6	30.3	88.5	1.3	1.3	1.4	237.7	112.1	275.1
Zeno	7	4.0	1.6	6.9	1.1	0.8	3.1	2.0	1.0	2.2	0.9	0.8	1.0

Table 2: Performance on IPC optimal-track instances where our factoring strategy does not abstain. Best coverage **bold**, better-than-baseline coverage *italic*. "BL" blind heuristic; "HM" $h^{\max}$; "LM" LM-cut; "PPrune" partition pruning; "POReduct" partial-order reduction; "SB" Symba. The scatter plots show runtime with LM-cut (A* on the x-axis). In (B) and (C), "$\sum$D" is the factor over the per-domain sums, and "Gmea" ("max") is the Geometric mean (the maximum) over the per-instance factors. Instances for (B) are commonly solved ones, for (C) the same but excluding ones commonly solved in ≤ 0.1 seconds (FD rounds runtimes ≤ 0.1 to 0.1). Rovers $\mathcal{F}$' uses our alternative factoring strategy with leaf size bound $M = 10$ million.

Future Research Directions

In preliminary work, we have determined that the scope of fork factoring can be extended to *star-topology factoring*: replace the "root" with a "center" and the leaves with "periphery factors", allowing arbitrary interactions between the center and each periphery factor. This corresponds to a much generalized notion of "conditional independence": For any fixed center path, the periphery factors can still be moved independently of each other, given that each complies with the preconditions requested and provided by the center, as well as co-occurring effects with the center. Note that this allows to address planning tasks whose causal graph is strongly connected. Indeed, if we are willing to tackle single-leaf factorings (see next), then *any* partition of the state variables into two subsets yields a star-topology factoring.

We currently enumerate leaf states explicitly, a major caveat when dealing with large leaf factors. But one could represent the pricing functions *symbolically* instead, via ADDs (Bahar et al. 1997). This is true also of star-topology factoring, which then becomes a highly configurable form of mixing explicit-state search with symbolic search, including in particular the option to freely divide the state variables into ones ("center") tackled by explicit search vs. ones ("single periphery factor") tackled by symbolic search.

Beyond this, the road opens up to target-profile factoring more generally. Are there interesting structures other than "conditional independence" to exploit? In chain structures, e. g., what about restrictions on the "domain transition graphs" of the factors to guarantee that backtracking won't be needed, abstaining if these restrictions are not satisfied? Can we combine different target profiles through hierarchical sub-structuring, where "factors" at a higher level are optimally solved via a target profile at a lower level? Viewing target-profile factoring as a highly parameterized way of reformulating the search, can we benefit from orthogonal enhancements like partial-order reduction, macro-actions (e. g. (Vidal 2004; Botea et al. 2005)), symmetries (e. g. (Domshlak, Katz, and Shleyfman 2012))? Can we understand under which circumstances a reformulation is beneficial or detrimental? Can we predict which alternative target profile best matches the input, and design self-configuring portfolios?

Last but not least, it is worth mentioning that star topology is a classical system design paradigm in many areas of CS. Hence star-topology decoupled state space search could turn out to be useful for control, synthesis, and verification problems far beyond the realm of AI Planning.

Acknowledgments. We thank the anonymous reviewers, whose comments helped to improve the paper. We thank Carmel Domshlak for discussions, which contributed to many of the ideas for future research.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In Gottlob, G., ed., *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 929–935.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebraic decision diagrams and their applications. *Formal Methods in System Design* 10(2/3):171–206.

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

Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds. 2012. *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12).*

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

Brafman, R., and Domshlak, C. 2003. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research* 18:315–349.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In Gil, Y., and Mooney, R. J., eds., *Proc. 21st National Conference of the American Association for Artificial Intelligence (AAAI-06)*, 809–814.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen et al. (2008), 28–35.

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

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2014. IBACOP and IBACOP2 planner. In *IPC 2014 planner abstracts*, 35–38.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In Bonet et al. (2012).

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-optimal factored planning: Promises and pitfalls. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 65–72.

Gnad, D., and Hoffmann, J. 2015. Fork-decoupled state space search. Technical report. Available at `http://fai.cs.uni-saarland.de/hoffmann/papers/icaps15a-tr.pdf`.

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

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

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

Jonsson, P., and Bäckström, C. 1995. Incremental planning. In *European Workshop on Planning*.

Katz, M., and Domshlak, C. 2008. Structural patterns heuristics via fork decomposition. In Rintanen et al. (2008), 182–189.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored planning using decomposition trees. In Veloso, M., ed., *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1942–1947.

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

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A monte carlo random walk approach. In Bonet et al. (2012), 181–189.

Nissim, R., and Brafman, R. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research* 51:293–332.

Nissim, R.; Apsel, U.; and Brafman, R. I. 2012. Tunneling and decomposition-based state reduction for optimal planning. In Raedt, L. D., ed., *Proc. 20th European Conference on Artificial Intelligence (ECAI'12)*, 624–629.

Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds. 2008. *Proc. 18th International Conference on Automated Planning and Scheduling (ICAPS'08).*

Torralba, A., and Alcázar, V. 2013. Constrained symbolic search: On mutexes, BDD minimization and more. In Helmert, M., and Röger, G., eds., *Proc. 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 175–183.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Koenig, S.; Zilberstein, S.; and Koehler, J., eds., *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 150–160.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In Bonet et al. (2012).

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

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

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research* 32:565–606.