Beating LM-cut with \( h_{\text{max}} \) (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 this new form of interaction. In hierarchical factored planning, at higher levels we only need to consider 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 \( h_{\text{max}} \) 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 \(\Pi = (V, A, I, G)\). \(V\) is a set of state variables, where each \(v \in V\) is associated with a finite domain \(D(v)\). We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to \(V\) is a state. \(I\) 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 \((\text{pre}(a), \text{eff}(a), \text{cost}(a))\) where the precondition \(\text{pre}(a)\) and effect \(\text{eff}(a)\) are partial assignments to \(V\), and \(\text{cost}(a) \in \mathbb{R}^{+}\) is the action’s non-negative cost.

For a partial assignment \(p\), \(V(p) \subseteq V\) denotes the subset of state variables instantiated by \(p\). For any \(V' \subseteq V(p)\), by \(p[V']\) we denote the value of \(V'\) in \(p\). An action \(a\) is applicable in a state \(s\) if \(\text{pre}(a) \subseteq s\), i.e., if \(s[v] = \text{pre}(a)[v]\) for all \(v \in V(\text{pre}(a))\). Applying \(a\) in \(s\) changes the value of each \(v \in V(\text{eff}(a))\) to \(\text{eff}(a)[v]\), 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 \(s[(a_1, \ldots, a_n)]\). A plan for \(\Pi\) is an action sequence s.t. \(G \subseteq I[(a_1, \ldots, a_n)]\). A plan is optimal if its summed-up cost is minimal among all plans for \(\Pi\).

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 \((v, v') \in [V(\text{eff}(a)) \cup V(\text{pre}(a))] \times V(\text{eff}(a))\).

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 \(\Pi\) be an FDR task with variables \(V\). A factoring \(F\) is a partition of \(V\) into non-empty subsets \(F\), called factors. The interaction graph \(IG(F)\) of \(F\) 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 \(CG\).

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

The only global interactions in a fork factoring consist in the root factor establishing preconditions for actions moving individual leaf factors. But when do factorings exist, and how to find one? Denote by \(F^{\text{SCC}}\) the factoring whose factors are the strongly connected components of \(CG\). Clearly, any fork factoring \(F\) must be coarser than \(F^{\text{SCC}}\), i.e., for every \(F \in F^{\text{SCC}}\) we must have \(F' \in F\) with \(F \subseteq F'\). In other words, we must perform the factoring at the level of SCCs in the causal graph. In particular, if \(|F^{\text{SCC}}| = 1\) (the causal graph consists of a single SCC), then the only fork factorings are the trivial and pathological ones.

For the case where \(|F^{\text{SCC}}| > 1\), consider Figure 1. We view the interaction graph \(IG(F^{\text{SCC}})\) 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 \(F \in F^{\text{SCC}}\) is fully contained in either of \(T\) or \(B\), and where the only arc in \(IG(\{T, B\})\) is \((T \rightarrow B)\). Let \(W\) be the set of weakly connected components of \(F^{\text{SCC}}\) within \(B\). Then we obtain a fork factoring \(F\) by setting \(F^T := T\) and \(F^L := W\). Vice versa, any non-trivial fork factoring can be obtained in this manner, except the redundant ones where some \(F^L \in F^L\) contains several weakly connected components from \(W\). 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).

Fork-Decoupled State Spaces

We assume an FDR task \(\Pi = (V, A, I, G)\) and a fork factoring \(F\) with root \(F^R\) and leaves \(F^L \in F^L\). 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 $F_L \in F_L$, 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 $J$ when ignoring preconditions on the root. Value assignments to $F_R$ are root states, notated $s^R$ and value assignments to any $F_L \in F_L$ are leaf states, notated $s^L$. For the leaf states of one particular $F_L \in F_L$, 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 $\text{pre}(a^L_{|F_L}) \subseteq s^R$.

The fork-decoupled state space augments root paths with maintenance of cheapest compliant leaf paths. A leaf path $\pi^L = (a^L_{i_1}, \ldots, a^L_{i_n})$ complies with a root path $\pi^R$ (also: is $\pi^R$-compliant) if we can schedule the $a^L_{i_j}$ monotonically increasing points alongside $\pi^R$ so that each $a^L_{i_j}$ 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 $(r(t), \text{prices}(s))$ with $r(t)$ being a root state, and prices$(s) : S^L \mapsto \mathbb{R}^R \cup \{\infty\}$ being a function which assigns every leaf state a non-negative price. The fork-decoupled state space is a labeled transition system $\Theta^F = (S^F, A^R, T^F, I^F, S^F_0)$ as follows:

(i) $S^F$ is the set of all fork-decoupled states $s$.
(ii) $A^R$, the set of root actions, gives the transition labels.
(iii) $T^F$ is the set of transitions, with $(s \xrightarrow{a} t) \in T^F$ if $a \in A^R$, $r(t)[a] = r(t)$, and, for every leaf state $t_L$, prices$(t)[s^L] = \min_{c(s^L)} \{\text{prices}(s)[s^L] + c(s^L)\}$ where $c(s^L)$ is the cost of a cheapest path of $A^L_{\text{in}(t)}$ actions from $s^L$ to $t^L$ if such a path exists, else $c(s^L) = \infty$.
(iv) $I^F$ is the fork-decoupled initial state, where $r(I^F) = I[F_R]$, and $\text{prices}(I^F)[s^L_{\mid F_L}]$ for every leaf state $s^L \in S^L_{\mid F_L}$, for some $F_L \in F_L^C$, is the cost of a cheapest path of $A^L_{\text{in}(I^F)}$ actions from $I[F^L]$ to $s^L$ if such a path exists, else $\text{prices}(I^F)[s^L] = \infty$.
(v) $S^F_0$ are the fork-decoupled goal states $s_G$, where $r(s_G)$ is a goal root state and, for every $F_L \in F_L^C$, there exists a goal leaf state $s^L \in S^L_{\mid F_L}$ s.t. prices$(s_G)[s^L] < \infty$.

The cost of a path $\pi^F = s_0 \xrightarrow{a_1} s_1 \cdots s_{n-1} \xrightarrow{a_n} s_n$ in $\Theta^F$ is $\text{cost}(\pi^F) := \sum_{i=1}^n \text{cost}(a_i)$. A fork-decoupled plan for $\Pi$ is a path $\pi^F$ from $I^F$ to an element of $S^F_0$.

Note that, as stated, $S^F$ is infinite. The reachable part of $S^F$, however, is finite because it corresponds to root paths leading to different end states and/or pricing functions.

¹That is, denoting by $(s^R, \ldots, s^R_n)$ the root states traversed by $\pi^R$, there is a monotonically increasing function $f : \{1, \ldots, n\} \rightarrow \{0, \ldots, n\}$ s.t., for every $i \in \{1, \ldots, n\}$, $a^L_{i} \in A^L_{\text{in}(f(i))}$, where prices decrease monotonically along any $\Theta^F$ path. The latter is because prices$(s)[|t^L]$ forms part of the minimization in (iii), namely when setting $s^L := t^L$ and using the empty path of $A^L_{\text{in}(t)}$ actions.

Intuitively, $\Theta^F$ 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 $\text{cost}(\pi^F)$ of a fork-decoupled plan $\pi^F$ 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 $\text{prices}(s_G)[s^L]$ for a cheapest goal leaf state $s^L \in S^L_{\mid F_L}$. 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 $F$ has $F_R = \{t_A, t_B\}$ and the only leaf $F_L = \{p\}$. Definition 2 (i): The fork-decoupled states $S^F$ 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 $I^F$ has $r(I^F) = \{t_A = l_1, t_B = l_3\}$; prices$(I^F)$ assigns 0 to $p = l_1$, 1 to $p = A$, and $\infty$ to all other leaf states, because $A^L_{\text{in}(I^F)}$ 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 $I^F$, 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 $\Theta^F$. Say we obtain $s_1$ from $I^F$ by moving $t_A$ to $l_2$. Then $r(s_1) = \{t_A = l_2, t_B = l_3\}$. The price tags prices$(s_1)$ are still 0 for $p = l_1$ and 1 for $p = A$, as these were the prices in $I^F$, no cheaper compliant paths become available in $s_1$, and $\text{prices}(s_1)[s^L]_{F_L}$ 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 $I^F$. The only change in prices$(s_1)$ is that $p = l_2$ gets price 2, accounting for "upload($p, t_A, l_2$)" which is compliant at $s_1$ (but not at $I^F$).

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 prices$(s_3)[p = t_B] = 3$ and 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 $t_2$ 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($p, t_A, l_1$), upload($p, t_A, l_3$)" yielding 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
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 \( \pi^R \), denoting by \( s \) the fork-decoupled state reached by \( \pi^R \) in \( \Theta^F \), and for every leaf state \( s^L \), \( \text{prices}(s)[s^L] \) is exactly the cost of a cheapest leaf path \( \pi^L \) that complies with \( \pi^R \) and achieves \( s^L \).

**Proof Sketch:** Showing this by induction, it is clearly true in \( T^F \). Presuming it is true in \( s \), let \( (s \xrightarrow{t} s') \in T^F \) and let \( tL \) be a leaf state. Observe that any cheapest leaf path \( \pi^L[t] \) compliant at \( t \) and achieving \( tL \) must consist of two parts, a cheapest leaf path \( \pi^L[s] \) compliant at \( t \) achieving some leaf state \( sL \), and a cheapest path of \( A^L|_{\pi^{L}} \) actions from \( sL \) to \( tL \). By induction, \( \text{prices}(s)[s^L] \) 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 goal leaf states:

**Definition 3 (Goal Price)** The goal price of \( s_G \in S^F \) is

\[
\text{GlobalCost}(s_G) := \min_{s \in S^F} \text{GlobalCost}(s) \quad \text{such that} \quad s_G[s] = \min_{s \in S^F} \text{GlobalCost}(s).
\]

The global cost of a \( \theta^F \) path \( \pi^F \) ending in \( s_G \in S^F \) is \( \text{GlobalCost}(\pi^F) := \text{cost}(\pi^F) + \text{GlobalCost}(s_G) \).

A fork-decoupled plan is fork-optimal if its global cost is minimal among all fork-decoupled plans.

For a fork-decoupled plan \( \pi^F \) ending in \( s_G \in S^F \), define \( \text{GlobalPlan}(\pi^F) \) as \( \pi^F \) augmented, for every \( F^L \in F^L \), with a cheapest compliant leaf path \( \pi^L \) achieving \( \min \{ \text{prices}(s_G)[s^L] \mid s^L \in S^L[F^L], s^L \subseteq \text{GlobalCost}(s_G) \} \). Vice versa, given a plan \( \pi \) for \( \Pi \), let \( \text{ForkPlan}(\pi) \) be the \( \Theta^F \) path \( \pi^F \) induced by the root actions in \( \pi \). We have:

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

**Proof Sketch:** As \( F \) is a factoring, any plan for \( \Pi \) can be perceived as a root path \( \pi^R \) achieving the goal for \( F^R \), augmented for every leaf \( F^L \) with a compliant leaf path \( \pi^L \) achieving \( G[F^L] \). The claim follows with Lemma 1. \( \square \)

What can we say about the size of \( \Theta^F \)? Consider the reachable states in \( \Theta^F \), denoted \( S^F R \), and those in the standard state space, denoted \( S^R \). A fork-decoupled state \( s \in S^F R \) can be viewed as a “condensed” set of (priced) worlds 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 \( \text{prices}(s)[s^L] < \infty \). 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 \( |S^F R| = 1 \). 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 line of \( m \) locations, and all packages and the truck start in the same location, then \( |S^R| = m \times (m + 1)^n \), whereas \( |S^F R| = \frac{m^{(m+1)n}}{2} \) because the leaf state prices are the same for each package as a function of the truck moves executed. However, there also are bad cases where \( S^F R \) enumerates combinations of leaf states for the same leaf, and \( S^F R \) is exponentially larger than \( S^R \). Our examples showing this are contrived though.

On IPC benchmarks, with our current factoring strategy, \( S^F R \) 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 \( \Pi \) be an FDR task, and \( F \) a factoring. A heuristic is a function \( h : S^F \rightarrow \mathbb{R}^+ \cup \{\infty\} \). The root-perfect heuristic, \( h^{R^*} \), is that where \( h^{R^*}(s) \) for any \( s \) is the minimum over cost(\( \pi^F \)) for all \( \Theta^F \) paths \( \pi^F \) from \( s \) to some \( s_G \in S^F \). The fork-perfect heuristic, \( h^{F^*} \), is that where \( h^{F^*}(s) \) for any \( s \) is the minimum over GlobalCost(\( \pi^F \)) for all such paths. We say that \( h \) is root-admissible if \( h \leq h^{R^*} \), and fork-admissible if \( h \leq h^{F^*} \).

The conceptual distinction between \( h^{R^*} \) and \( h^{F^*} \) 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, \( h^{F^*} \) 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 \( \Pi \), cf. Theorem 1). We will refer to heuristics attempting to estimate \( h^{R^*} \) as root heuristics, and to heuristics attempting to estimate \( h^{F^*} \) as fork heuristics, and distinguish them notionally by superscripts “\( R^* \)” respectively “\( F^* \)”.

Observe that \( h^{F^*} \) 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 GlobalCost(\( \pi^F \)). 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, \( h^{F^*} \) 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 \( h^{F^*} \): We can compute \( h^{R^*} \) as \( h^{F^*} \) 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 \( h^{F^*} \) 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 \( h^{F^*} \) will draw search towards plans (b).

Turning our attention to the computation of practical heuristics, we consider \( h^{F^*} \) first. Note that, to estimate \( h^{F^*} \), 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 \( \Theta^F \):
Definition 5 (From Classical $h$ to $h^{F^{*}}$ Estimation) Let $h$ be any FDR planning heuristic, $\Pi = (V, A, I, G)$ an FDR task, $F$ a fork factoring, and $s$ a fork-decoupled state. The leaves-for-pay fork heuristic estimate is $h^{PF}_{PLS}(s)$ using the FDR task $\Pi_{PLS} = (V, A_{PLS}, s_{PLS}, G)$ obtained from $\Pi$ by setting $s_{PLS} := \mathcal{R}(s) \cup \{V \setminus F^{R}\}$ and including for every leaf state $s^{L} \in S^{L}_{PLS}$, where $s^{L} \neq I[F^{L}]$ and $\text{prices}(s)[s^{L}] < \infty$ a new leaf action $a[s^{L}]$ with empty precondition, effect $s^{L}$, and cost $\text{cost}(a[s^{L}]) := \text{prices}(s)[s^{L}]$.

This construction caters for “leaf actions to be scheduled before and up to $s^{P}$, but not by allowing to explicitly insert such actions. We instead allow the planner on $\Pi_{PLS}$ 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 $h^{PF}_{PLS}$ is fork-admissible. If $h = h^{*}$, then $h^{PF}_{PLS} = h^{F^{*}}$.

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

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

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

Theorem 3 If $h$ is admissible, then $h^{R}_{PLS}$ is root-admissible. If $h = h^{*}$, then $h^{R}_{PLS} = h^{R^{*}}$.

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

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 $\Pi$ be an FDR task, $F$ a fork factoring, and $\Theta^{F}$ the fork factoring state space. The explicit goal-price state space $\Theta^{G}_{GS}$ is like $\Theta^{F}$ but with a new state $G'$ and, for every $s_{C} \in S_{C}^{F}$, a new transition $s_{C} \rightarrow G'$ with cost $\text{GPrice}(s_{C})$; 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 $\Theta^{F}$, which is non-standard due to the goal state price tags, in terms of standard search on $\Theta^{G}_{GS}$; via this correspondence: (direct from construction)

Algorithm Fork-Decoupled $X$ (FDX):

Input: FDR planning task $\Pi$, fork factoring $F$
Heuristic search algorithm $X$
Fork heuristic $h^{F}$

Output: A plan for $\Pi$, or “failed”
Let $h^{G}_{GS} := \begin{cases} h^{F}(s) & s \in S^{F} \\ 0 & s = G' \end{cases}$
Run $X$ with $h^{G}_{GS}$ on $\Theta^{G}_{GS}$
If $X$ found a solution path $\pi = \pi^{F} \circ (s_{G} \rightarrow G')$
return GlobalPlan($\pi^{F}$)
else return “failed”

\begin{figure}[h]
    \centering
    \includegraphics[width=\textwidth]{figure3}
    \caption{Exploiting any known search algorithm $X$.}
    \label{fig:fdx}
\end{figure}

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

Consider Figure 3. FDX just runs any search algorithm $X$ on $\Theta^{G}_{GS}$. 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 $h^{F}, h^{F}_{GS}$ is admissible in $\Theta^{G}_{GS}$. \hfill $\square$

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 $g(N) + h^{F}(s)$ 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 $h^{F}$. 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 $N[s_{C}]$, let $\pi^{F}$ be the fork-decoupled plan leading to $N$, and let $\pi^{F}_{GS}$ be any fork-decoupled plan that would be generated later if we continued the search. As $\text{GPrice}(s_{C}) = \text{MINPrice}$, $\pi^{F}_{GS}$ pays at least as high a goal price as $\pi^{F}_{GS}$. For root-admissible $h^{R}$, because we use $A^{*}$, $\text{cost}(\pi^{F}) \leq \text{cost}(\pi^{F}_{GS})$. So $\pi^{F}$ cannot have better global cost than $\pi^{F}_{GS}$. 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 $\pi^{F}_{GS}$. Optimality of FRA* is a special case. \hfill $\square$

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 $\text{GPrice}(s_{C}) = \text{MINPrice}$. 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).
Figure 4: New search algorithms. Search nodes are notated $N[s]$ where $s$ is the state and $N$ the node itself. MINGprice is the sum, over the leaf factors $F^L \in \mathcal{F}^L$, of optimal plan cost for the projection of $\Pi$ onto $\mathcal{F}^L$.

**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 function is maintained enumeratively for each leaf factor.

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 $D := IG(\mathcal{F}^{SCC})$ 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 $B_i - B_{i-1} \cup L$ and $W_L$ being the weakly connected components within $B_i$, all $W^L \in W_L$ satisfy $\sum_{v \in \mathcal{F}^L} |D(v)| \leq 2^{|B_i|}$. If there are no candidates, terminate the iteration. Else, select a candidate $L$ with minimal $\prod_{v \in \mathcal{F}^L} |D(v)|$. Set $B_i := B^L_i - B_{i-1}$, define $\mathcal{F}_i$ by $\mathcal{F}^R_i := V \setminus B_i$ and $\mathcal{F}^L_i := W_{B_i}$, 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 $\Pi$. Otherwise, we select the factoring $\mathcal{F}_i$ whose number of leaf factors is maximal, and whose index $i$ is minimal among these factorings.

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 $IG(\mathcal{F}^{SCC})$ is reliably very small. The runtime is (rounded to) 0.00 in 97% of all IPC STRIPS tasks, and is 1.01 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 $\sum_{F^L \in \mathcal{F}^L} \prod_{v \in \mathcal{F}^L} |D(v)|$ 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 FRA*, 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 $|S_R| = 1$ 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).

---

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.

---

**Table 2**

<table>
<thead>
<tr>
<th>Domain</th>
<th>FDA*</th>
<th>AFRA*</th>
<th>FRA*</th>
<th>FDA*-LMcut</th>
<th>AFRA*-LMcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistics</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>NoMystery</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>Woodwork</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>109</td>
</tr>
<tr>
<td>Rovers</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>109</td>
</tr>
</tbody>
</table>

---

The figures show the percentage of instances covered for each planner on the NoMystery test suites. The coverage is categorized by domain type.
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; “SB” Symba. The scatter plots show runtime with LM-cut (A* on the x-axis). In (B) and (C), “∑D Gmea max” is the factor over the per-leaf sum, 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 F* uses our alternative factoring strategy with leaf size bound $M = 10$ million.

Future Research Directions

In summary, 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; “SB” Symba. The scatter plots show runtime with LM-cut (A* on the x-axis). In (B) and (C), “∑D Gmea max” is the factor over the per-leaf sum, 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 F* uses our alternative factoring strategy with leaf size bound $M = 10$ million.
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


