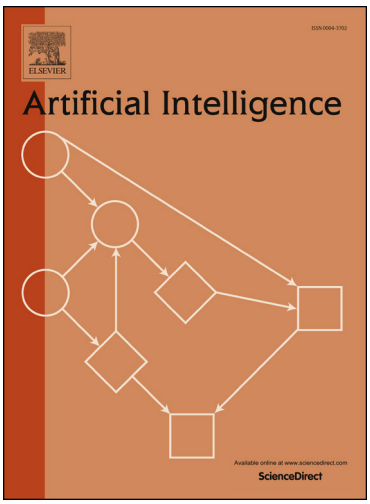
Accepted Manuscript



Star-Topology Decoupled State Space Search

Daniel Gnad, Jörg Hoffmann

PII: S0004-3702(17)30173-X
DOI: https://doi.org/10.1016/j.artint.2017.12.004
Reference: ARTINT 3049

To appear in: *Artificial Intelligence*

Received date: 3 May 2017
Revised date: 20 December 2017
Accepted date: 21 December 2017

Please cite this article in press as: D. Gnad, J. Hoffmann, Star-Topology Decoupled State Space Search, *Artif. Intell.* (2018), https://doi.org/10.1016/j.artint.2017.12.004

Star-Topology Decoupled State Space Search

Daniel Gnad

Saarland University, Saarbrücken, Germany

Jörg Hoffmann

Saarland University, Saarbrücken, Germany

Abstract

State space search is a basic method for analyzing reachability in discrete transition systems. To tackle large compactly described transition systems – the state space explosion – a wealth of techniques (e. g., partial-order reduction) have been developed that reduce the search space without affecting the existence of (optimal) solution paths. Focusing on classical AI planning, where the compact description is in terms of a vector of state variables, an initial state, a goal condition, and a set of actions, we add another technique, that we baptize *star-topology decoupling*, into this arsenal. A star topology partitions the state variables into components so that a single center component directly interacts with several leaf components, but the leaves interact only via the center. Many applications explicitly come with such structure; any classical planning task can be viewed in this way by selecting the center as a subset of state variables separating connected leaf components.

Our key observation is that, given such a star topology, the leaves are conditionally independent given the center, in the sense that, *given a fixed path of transitions by the center, the possible center-compliant paths are independent across the leaves.* Our decoupled search hence branches over center transitions only, and maintains the center-compliant paths for each leaf separately. As we show, this method has exponential separations to all previous search reduction techniques, i. e., examples where it results in exponentially less effort. One can, in principle, prune duplicates in a way so that the decoupled state space can never be larger than the original one. Standard search algorithms remain applicable using simple transformations. Our experiments exhibit large improvements on standard AI

Email addresses: gnad@cs.uni-saarland.de (Daniel Gnad), hoffmann@cs.uni-saarland.de (Jörg Hoffmann)

planning benchmarks with a pronounced star topology.[1]

Keywords: AI Planning, heuristic search, problem decomposition

1. Introduction

Reachability analysis in large discrete state transition systems arises in several areas of computer science. Examples are AI planning [4] and diagnosis [5], model checking [6], and multiple sequence alignment [7]. The question is whether, starting from a given state s, the system can reach a given state t, or can reach some state satisfying a given property, like a planning goal in AI, or the negation of a safety property in model checking. Answering this question is hard due to the *state explosion* problem [8]. The transition system is compactly described, in terms of state variables and transition rules, a network of synchronized automata, or a bounded Petri net. The size of the system itself – the compact description's *state space* – is exponential in the size of that description.

Forward state space search is one basic method for reachability analysis. A wealth of techniques have been developed that reduce the search space while preserving *completeness* (finding a solution if one exists) and, ideally, *optimality* (finding a solution with minimum summed-up transition cost). *Partial-order reduction* exploits permutable parts of the state space [8, 9, 10, 11, 12, 13, 14, 15, 16]. *Symmetry reduction* exploits symmetric parts of the state space [17, 18, 19, 20, 21]. In *Petri-net unfolding*, the search space is an acyclic Petri net (a DAG) over *conditions* (vertices annotated with state-variable values) and *events* (vertices annotated with transitions), where events are added based on which *markings* (combinations of state-variable values) are reachable [22, 23, 24, 25, 26, 27, 28].

Our contribution consists in a new search reduction method, *star-topology decoupling*, which is complementary to all previous methods, and can be configured to either preserve completeness and optimality, or to only preserve completeness (allowing stronger reductions). We introduce the method in AI planning, where a planning *task* is given in terms of finite-domain state variables, an *initial state*, a *goal* condition, and a set of *actions* describing the possible transitions.

[1] Parts of the presented material were previously published [1, 2, 3]. This article introduces, analyses, and evaluates our techniques much more comprehensively. In particular, it adds new theoretical results pertaining to the space of star-topology factorings, to pruning methods avoiding state space blow-ups, and to exponential separations from previous techniques.

1.1. Star-Topology Decoupling

The distinguishing feature of star-topology decoupling is the assumption of a particular structural profile, a *star topology*. Viewing disjoint subsets of state variables as *components*, in a star topology a single center component interacts with multiple leaf components, but the leaves interact with each other only via the center. Many applications explicitly come with such structure. For example, distributed systems are often synchronized via a central component (client-server architectures, shared-memory computing systems), and cooperative agents are synchronized via their shared (commonly affected/queried) state variables. Arbitrary AI planning tasks can be viewed in this way by selecting the center as some subset of state variables breaking the dependencies between connected (leaf) components of the remaining state variables.

The key to star-topology decoupling is a particular form of "conditional independence": *given a fixed path of transitions by the center, the possible center-compliant paths are independent across the leaves*. For example, say the center C is a state variable encoding the position of a vehicle v, and each leaf L is a state variable encoding the position of a transportable object o. Given a fixed path π^C of vehicle moves, the compliant moves for any object o, alongside π^C, are those which load/unload o at those points on π^C where v is currently at the required location. Any sequence of such load/unload actions for o – any π^C*-compliant path* for o – can be committed to for o, independently of what any other transportable object o' is committed to. *Decoupled search* exploits this property by searching over center paths π^C only; it maintains, alongside each π^C, the leaf states reachable on π^C-compliant paths. This way, the component (local) state spaces are searched separately, avoiding the enumeration of combined (global) states across leaves. In catchy (though imprecise) analogy to conditional independence in graphical models, star-topology decoupling "instantiates" the center to break the dependencies between the leaves.

The search is least-commitment in the sense that leaf moves are committed to only at the end, when the goal is reached. During the search, the leaf states reachable on π^C-compliant paths are maintained exhaustively. Namely, the end point of each center path π^C in the search is associated with a *decoupled state* s, which for every leaf component L stores those leaf states s^L of L reached by π^C-compliant paths. One can view s as a compact representation of a set of global states, its *hypercube* $[s]$: all states formed from the center state reached by π^C, and any combination of π^C-compliant s^L. These are exactly those global states that can be reached on a transition path whose center sub-path is π^C.

Optionally, each s^L is annotated with the cost of a cheapest π^C-compliant path achieving s^L. We refer to that cost as s^L's *price*: it is not a cost we have already committed to paying, but a cost we *will* pay if, at the end, we commit to using s^L. Maintaining leaf state prices allows to preserve optimality; maintaining only leaf state reachability suffices to preserve completeness.

In the example above, for any transportable object o, the object's initial location l_0 is reachable on a π^C-compliant path, namely the empty path, at cost 0. Every other location l for o is reachable on a π^C-compliant path of length 2 – load at l_0, unload at l – iff π^C visits l_0 and afterwards visits l. Once the goal location of o is reachable, we can commit to a compliant path, i. e., a suitable load/unload pair, moving o to its goal location. In case different loads/unloads have different cost, distinguishing leaf state prices allows to select a cheapest such pair for each o. Searching over center paths enabling different-cost pairs allows to guarantee global optimality.

Star-topology decoupling has been inspired by *factored planning* methods, which also partition the state variables into components [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. *Hierarchical* factored planning is remotely related; the search for a plan proceeds top-down in a hierarchy of increasingly more detailed levels identified by the factors. *Localized* factored planning is more closely related; the search for a plan proceeds by first planning locally on individual components, followed by global cross-component constraint resolution. In comparison to both, the key feature of star-topology decoupling is the focus on star topologies, which limits the possible interactions across factors, facilitating specialized search algorithms (as outlined above).

Star-topology decoupling also relates to Petri net unfolding, specifically to *contextual unfolding* [27] as planning actions typically have non-consumed preconditions (typically called *prevail conditions*). For example, a load action requires, but does not consume, a particular vehicle position. One can view star-topology decoupling as a new form of unfolding where the "conditions" are component states, and the star topology is exploited (a) to avoid the enumeration of exponentially many event "histories", keeping track of which prevail conditions may have been consumed in the past; as well as (b) to get rid of the **NP**-hard problem of testing the reachability of a marking in an unfolding prefix (for decoupled search, this is testable in linear time). As we shall see, these theoretical advantages often translate into empirical ones.

1.2. Experiments Preview

Figure 1 gives a preview of empirical results. Details will be provided later. We use the standard benchmarks from the *international planning competition (IPC)*, which are all solvable; and we use an established collection of unsolvable benchmarks including those of the *unsolvability* IPC'16. Star topologies are found automatically using a simple *factoring strategy*, identifying an *X-shape* over the input task's variable dependencies; the factoring strategy *abstains* if no non-trivial star topology (≥ 2 leaves) is found this way.[2]

We build (the respective representations of) the entire reachable state spaces, as a canonical measure of reduction power: *How much can a reduction method achieve on its own?* We compare our method to partial-order reduction and Petri net unfolding, the most competitive related methods (a comparison to the most competitive localized factored planning method will be included later). The Petri net unfolding tools are provided with straightforward encodings of planning tasks into Petri nets, following Hickmott et al. [24].

Decoupled states are more complex structures than standard states, so we show not only the number of states, but also the amount of memory – the number of integer variables in our C++ implementation – used to represent them. For the standard state space, this is simply the number of states times the number of state variables. For decoupled states, we compute the state space of each leaf once up front, give IDs to its leaf states, and use these IDs for reference later on. As the unfolding tools often lag far behind in the number of state spaces successfully built, their state space representation size data is omitted from this preview (but will be discussed later).

The data clearly attest to the power of star-topology decoupling. Its completeness-only (COM) variant successfully builds more than twice as many state spaces as standard state space search, and more than three times as many as Petri net unfolding. On commonly solved instances, average state space size is typically reduced by at least one order of magnitude, and often by several orders of magnitude. In most cases, these improvements apply even when preserving optimality (OPT), and even relative to the state space pruned by strong stubborn sets.

That said, Figure 1 also exhibits two weaknesses. First, decoupled states are more costly to maintain, incurring a runtime overhead. In some cases, the overhead outweighs the reduction gain, and fewer state spaces are successfully built.

[2]The name "factoring" comes from factored planning; we will use the words "component" and "factor" interchangeably.

	# Instances		Reachable State Space. Middle & Right: Average over Instances Commonly Built by Std, POR, OPT, COM													
			Success						Number of States				Representation Size (in Thousands)			
Domain	All	X	Std	POR	Pun	Cun	OPT	COM	Std	POR	OPT	COM	Std	POR	OPT	COM
Solvable Benchmarks: From the International Planning Competition (IPC)																
Depots	22	22	4	4	2	2	3	**5**	1,193,760	1,193,760	1,173,096	**132,640**	30,954.8	30,954.8	35,113.1	**3,970.0**
Driverlog	20	20	5	5	3	3	8	**10**	3,005,640	3,005,640	27,404	**4,708**	35,632.4	35,632.4	706.1	**127.2**
Elevator	100	100	21	17	1	3	8	**41**	2,168,429	2,168,329	839,359	**7,902**	22,652.1	22,651.1	21,046.2	**186.7**
Floortile	80	80	**2**	**2**	0	0	0	**2**								
Logistic	63	63	12	12	7	11	23	**27**	422,508	422,507	2,605	**171**	3,793.8	3,793.8	85.5	**8.2**
Miconic	150	145	50	45	25	30	45	**145**	2,594,960	2,592,063	10,730	**68**	52,728.9	52,673.1	218.8	**2.4**
NoMyst	40	40	11	11	5	7	**40**	**40**	3,561,168	3,107,551	501	**370**	29,459.3	25,581.5	11.5	**10.0**
Pathway	30	30	**4**	**4**	3	3	**4**	**4**	875,781	**19,804**	177,383	177,378	54,635.5	**1,229.0**	11,211.9	11,211.9
PSR	50	3	3	3	3	3	3	3	2,385	2,055	**596**	**596**	39.4	33.9	**11.1**	**11.1**
Rovers	40	40	5	**6**	4	4	5	5	3,925,445	263,656	138,812	**131,127**	98,051.6	6,534.4	4,045.9	**4,032.9**
Satellite	36	36	**5**	**5**	**5**	**5**	4	4	170,808	36,149	109,268	**11,272**	2,864.2	582.5	2,219.1	**352.7**
TPP	30	29	5	5	4	4	**11**	**11**	10,029,509	9,593,134	20	**17**	340,961.5	326,124.8	.9	**.8**
Transport	140	140	28	23	11	11	18	**34**	621,070	621,047	499,889	**7,014**	4,958.6	4,958.5	12,486.4	**173.3**
Woodwor	100	87	11	20	16	**22**	16	16	9,042,574	**5,843**	370,345	197,138	438,638.5	**226.8**	16,624.1	9,688.9
Zenotrav	20	20	**7**	**7**	2	4	**7**	**7**	1,965,822	1,965,763	38,106	**3,385**	17,468.0	17,467.5	1,028.5	**99.4**
Unsolvability IPC'16																
BTransp	29	29	7	6	3	3	4	**11**	142,018	142,018	92,922	**1,225**	5,961	5,961	5,721	**73**
NoMyst	24	24	2	2	0	1	17	**18**	8,328,306	5,016,457	360	**296**	99,324	59,872	10	**9**
Rovers	20	20	7	7	0	3	7	**8**	8,949,038	6,169,356	830,272	**814,693**	247,683	172,389	25,687	**25,445**
Unsolvable Benchmarks: Extended from [42]																
NoMyst	40	40	9	8	2	4	**40**	**40**	5,014,955	3,875,186	113	**105**	85,254.2	65,878.2	3.9	**3.8**
Rovers	40	40	**4**	**4**	0	0	**4**	**4**	13,503,020	5,856,821	393,164	**361,084**	697,778.9	302,608.9	22,001.8	**20,924.4**
$\sum$	1074	1008	202	196	96	123	267	**435**								

Figure 1: State space size using an X-shape factoring strategy. Best results highlighted in **boldface**. "Success": reachable state space fully explored. "X": non-trivial X-shape ($\geq$ 2 leaves) identified, i. e., factoring strategy did not abstain. "Std": standard state space. "POR": standard state space with partial-order reduction using strong stubborn sets [16]. "Pun": Petri-net unfolding using Punf [43]. "Cun": contextual Petri-net unfolding using Cunf [44]. "OPT": decoupled state space, preserving completeness and optimality. "COM": decoupled state space, preserving completeness only. Representation size is the number of integer variables in the underlying C++ implementation (see text). All planning competition benchmark domains were run. Domains on which the factoring strategy abstained, and domains where no approach could build any state space, are not included in the table. Multiple test suites of the same domain are accumulated into the same table row. Runtime limit 30 minutes, memory limit 4 GB.

Second, Figure 1 shows only a selection of domains, namely those where the X-shape factoring strategy does (sometimes) not abstain. On the majority of the 49 IPC benchmark domains, the strategy abstains on every instance.

Summing up both observations, star-topology decoupling requires a particular form of structure to work well, namely a pronounced star topology, with many leaf components. Not all applications, nor all IPC benchmark domains, have that structure.

However, this weakness – requiring particular structure to work well – is

shared by all known search reduction techniques. The distinguishing feature of star-topology decoupling is that the required structure is explicit, and easily testable. This is valuable in practice, as it allows to avoid wasting runtime on unsuitable cases. The factoring strategy takes negligible runtime, and if it abstains, one can run an alternate technique instead. This is perfect for the design of solver portfolios, combining the strengths of different approaches (e. g. [45, 46]).

An issue not visible in Figure 1 is that, in practice, the objective is not to exhaust the entire state space, but to find a solution path, or prove that none exists. Search techniques orthogonal to ours are available to do so without having to exhaust the state space. Using such techniques reduces the advantage of star-topology decoupling over standard state space search. Nevertheless, as we shall see, star-topology decoupling often improves over the state of the art on IPC benchmarks.

1.3. Properties

Star-topology decoupling has *exponential separations* relative to all previous search reduction methods: example families whose decoupled state space has size polynomial in the size of the input, while the previous method's state space representation is exponential in that size.

Yet the technique is not without risks. The decoupled state space may be exponentially larger than the standard state space, and may even be infinite. As we show, however, (a) finiteness can be guaranteed with a simple *dominance pruning* technique, and (b) a more expensive *hypercube pruning* technique (involving a **co-NP**-complete sub-problem) guarantees that the number of reachable decoupled states is bounded by the number of reachable standard states. Empirically on the IPC benchmarks, (b) incurs a prohibitive computational overhead, and even without (b) the number of decoupled states never exceeds the number of standard states. So our implementation uses (a) only.

Star-topology decoupling combines gracefully with standard search methods, in particular with *heuristic search* algorithms, guiding state space exploration through *heuristic functions* mapping states to estimated goal distance [47]. Heuristic search has been extremely successful in AI planning (e. g. [48, 49, 50, 51, 52, 53, 54, 55]). As we show, heuristic search methods can be applied unmodified to decoupled search, via simple transformations, preserving their optimality and completeness guarantees.

The paper is structured as follows. Section 2 introduces the planning framework and basic notations. In Section 3, we define star topologies, as *factorings*

(state-variable partitions) inducing a star structure. Section 4 specifies the decoupled state space. Section 5 shows that blow-ups can occur, and identifies the dominance/hypercube pruning methods avoiding these. Section 6 discusses the relation to previous methods, including the exponential separations. Section 7 explains how to plug-in standard heuristic search techniques, and Section 8 presents our experiments. Section 9 concludes with a discussion of future research directions. Some proofs are moved out of the main text, into Appendix A.

2. Background

AI Planning is concerned with the design of mechanisms taking decision about action, finding *plans* that lead from an initial state to a goal. Here we consider *classical planning*, which assumes discrete state variables, complete knowledge about the initial state, and deterministic actions. The planning problem then consists of checking reachability in a large, discrete and deterministic, labeled transition system. We give our notation for such transition systems first, then give the syntax and semantics of our planning model.

A **labeled transition system** in our terminology is a tuple $\Theta = (S, L, c, T, I, S_G)$ consisting of a finite set of **states** S, a finite set of transition **labels** L, a function $c : L \mapsto \mathbb{R}^{0+}$ associating each label with its non-negative **cost**, a set of **transitions** $T \subseteq S \times L \times S$, an **initial state** $I \in S$, and a set of **goal states** $S_G \subseteq S$. We assume that Θ is **deterministic**, i. e., for every s and l there exists at most one s' such that $(s, l, s') \in T$. We will often write $s \xrightarrow{l} s'$ for $(s, l, s') \in T$, or $s \to s'$ if the label does not matter. A **solution** for $s \in S$ is a path π in Θ from s to a goal state. A solution for I is called a solution for Θ. We consider *additive cost*, i. e., the **cost** of a path π, denoted $\mathsf{cost}(\pi)$, is the summed-up cost of its labels. A solution for s (respectively Θ) is **optimal** if its cost is minimal among all solutions for s (respectively Θ).

Our planning syntax follows the finite-domain variables model (e. g., [56, 52]). A **planning task** is a tuple $\Pi = (V, A, c, I, G)$. V is a finite set of **state variables** v, **variables** for short, each associated with a finite domain $\mathcal{D}(v)$. 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**, where each action $a \in A$ is associated with its **precondition** $\mathsf{pre}(a)$, and its **effect** $\mathsf{eff}(a)$, each a partial assignment to V. The function $c : L \mapsto \mathbb{R}^{0+}$ associates each action with its **cost**. We will often write variable/value pairs (v, d) as $v = d$.

The semantics of planning tasks are defined via their *state spaces*, deterministic labeled transition systems as above. The state space of a planning task is

straightforwardly defined given the task's syntax; we do so via introducing a number of notations that will be useful. Our convention will be to denote states (as well as partial assignments) by p, q, reserving the more usual s, t for the *decoupled states* introduced later on. For a partial assignment p, $\mathcal{V}(p) \subseteq V$ denotes the subset of state variables on which p is defined. Given $V \subseteq \mathcal{V}(p)$, it will be convenient to denote by $p[V] := p|_V$ the restriction of p to V. An action a is **applicable** in a state p if $p[\mathcal{V}(\mathsf{pre}(a))] = \mathsf{pre}(a)$. The outcome of applying a in p is $p[\![a]\!] := p[\mathcal{V}(p) \setminus \mathcal{V}(\mathsf{eff}(a))] \cup \mathsf{eff}(a)$, i. e., we overwrite p with a's effect where defined. We will also use the notation $p[\![a]\!]$ for arbitrary partial assignments p.

Given a planning task $\Pi = (V, A, c, I, G)$, its **state space**, denoted Θ_Π, is the labeled transition system $\Theta_\Pi = (S, L, c, T, I, S_G)$ whose states S are all states of Π; whose labels L are the actions A; whose cost function c is that of Π; whose transitions $p \xrightarrow{a} q$ are those were a is applicable in p and $q = p[\![a]\!]$; whose initial state I is that of Π; and whose goal states S_G are those p where $p[\mathcal{V}(G)] = G$. A solution π for Θ_Π is a **plan** for Π. We will identify π with the sequence of actions labeling its transitions.

Given a planning task Π, deciding whether a plan exists is **PSPACE**-complete [57]. At an algorithmic level, AI planning distinguishes three different problems: *optimal planning*, where the objective is to find an optimal plan; *satisficing planning*, where it suffices to find any plan; and *proving unsolvability*, where the objective is to prove that the goal is unreachable.

Example 1. *We use three running examples, named the* Vanilla*,* NoEmpty*, and* Scaling *example respectively. The latter will be used for illustration as well as scalability arguments. All examples are based on simple transportation scenarios.*

In both the Vanilla and the NoEmpty example, there are two trucks t_A, t_B moving along three locations l_1, l_2, l_3 arranged in a line, and there is one transportable object o. The planning task $\Pi = (V, A, c, I, G)$ has variables $V = \{o, t_A, t_B\}$ where $\mathcal{D}(t_A) = \mathcal{D}(t_B) = \{l_1, l_2, l_3\}$ and $\mathcal{D}(o) = \{l_1, l_2, l_3, t_A, t_B\}$. The initial state is $I = \{t_A = l_1, t_B = l_3, o = l_1, \}$, i. e., t_A and o start at l_1, and t_B starts at l_3. The goal is $G = \{o = l_3\}$. The actions are truck moves and load/unload. All actions have cost 1*, and the only difference between the two examples is the precondition of truck moves. Namely, A consists of the actions:*

1. *move(t, x, y) for $t \in \{t_A, t_B\}$ and $\{x, y\} \in \{\{l_1, l_2\}, \{l_2, l_3\}\}$: precondition $\{t = x\}$ in the vanilla example; precondition $\{t = x, o = t\}$ in the no-empty example; effect $\{t = y\}$.*

2. *load*(t, x) *for* $t \in \{t_A, t_B\}$ *and* $x \in \{l_1, l_2, l_3\}$*: precondition* $\{t = x, o = x\}$*; effect* $\{o = t\}$.

3. *unload*(t, x) *for* $t \in \{t_A, t_B\}$ *and* $x \in \{l_1, l_2, l_3\}$*: precondition* $\{t = x, o = t\}$*; effect* $\{o = x\}$.

In the NoEmpty example, a truck can only move if the object is currently inside it. For both examples, an optimal plan is $\langle$*load*(t_A, l_1)*,* *move*(t_A, l_1, l_2)*,* *move*(t_A, l_2, l_3)*,* *unload*$(t_A, l_3)\rangle$.

The Scaling example is like the Vanilla example except that there is only one truck and we scale the number of objects as well as the length of the line. The planning task $\Pi = (V, A, c, I, G)$ *has variables* $V = \{t, o_1, \ldots, o_n\}$ *where* $\mathcal{D}(t) = \{l_1, \ldots, l_m\}$ *and* $\mathcal{D}(o_i) = \{l_1, \ldots, l_m, t\}$*. The initial state is* $I = \{t = l_1, o_1 = l_1, \ldots, o_n = l_1\}$*, i. e., the truck and all objects start at* l_1*. The goal is* $G = \{o_1 = l_m, \ldots, o_n = l_m\}$*, i. e., all objects must be transported to the other end of the line. The actions are as before, adapted to suit the modified example structure:*

1. *move*(x, y) *for* $x = l_i, y = l_j$ *such that* $|i - j| = 1$*: precondition* $\{t = x\}$*; effect* $\{t = y\}$.

2. *load*(o, x) *for* $o \in \{o_1, \ldots, o_n\}$ *and* $x \in \{l_1, \ldots, l_m\}$*: precondition* $\{t = x, o = x\}$*; effect* $\{o = t\}$.

3. *unload*(x, y) *for* $o \in \{o_1, \ldots, o_n\}$ *and* $x \in \{l_1, \ldots, l_m\}$*: precondition* $\{t = x, o = t\}$*; effect* $\{o = x\}$.

An optimal plan for this example loads all objects, drives to the other end of the line, and unloads all objects.

We require some additional notations and concepts. When we say that an action a **affects** a variable v, we mean that $v \in \mathcal{V}(\mathsf{eff}(a))$. Non-affected action preconditions, i. e., $\mathsf{pre}(a)[\mathcal{V}(\mathsf{pre}(a)) \setminus \mathcal{V}(\mathsf{eff}(a))]$, are referred to as **prevail conditions**. To characterize and identify star topologies, we will use the input task's **causal graph**, which captures direct state variable dependencies (e. g. [30, 58, 59, 52]). Given a planning task $\Pi = (V, A, c, I, G)$, the causal graph CG_Π is a directed graph whose vertices are the variables V, and that has an arc from u to v, denoted $u \to v$, if $u \neq v$ and there exists an action $a \in A$ such that either (i) $u \in \mathcal{V}(\mathsf{pre}(a))$ and $v \in \mathcal{V}(\mathsf{eff}(a))$, or (ii) $u \in \mathcal{V}(\mathsf{eff}(a))$ and $v \in \mathcal{V}(\mathsf{eff}(a))$. This captures (i) *precondition-effect* dependencies, as well as (ii) *effect-effect* dependencies. Intuitively, given a causal graph arc $u \to v$, changing the value of v

may involve changing that of u as well, because either (i) u may need to provide a precondition, or (ii) u may be affected as a side effect of changing the value of v.

We assume for simplicity that CG_Π is weakly connected. This is without loss of generality because, otherwise, the task can be equivalently split into several independent tasks.

Example 2. *The causal graphs of our running examples are shown in Figure 2.*

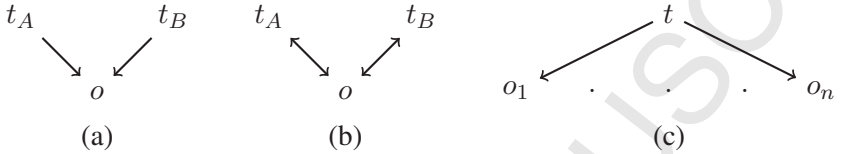


Figure 2: The causal graphs of our running examples: (a) Vanilla, (b) NoEmpty, (c) Scaling.

3. Star-Topology Factorings

We decompose planning tasks into components identified by disjoint sets of state variables. Following the factored planning literature (e. g. [33]), we refer to such decompositions, i. e., to partitions of the state variables, as *factorings*, and to the components as *factors*. A factoring identifies a star topology if its cross-component interactions take a star shape. Section 3.1 introduces the relevant variants of this concept. Section 3.2 characterizes the space of star-topology factorings, with a view on maximizing the number of leaf factors.

3.1. Concepts

We start with some special cases that are instructive due to their simplicity, and that are useful in practice as they are easy to identify (in particular, they underlie our current factoring strategies).

Definition 1 (Strict-Star Factoring). *Let Π be a planning task with variables V. A **factoring** $\mathcal{F}$ is a partition of V into disjoint non-empty subsets F, called **factors**.*

*Let $\mathcal{F}$ be a factoring. The **interaction graph** $\mathsf{IG}_\Pi(\mathcal{F})$ is the quotient graph of CG_Π given $\mathcal{F}$, i. e., the directed graph whose vertices are the factors, with an arc $F \to F'$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $v \to v'$ is an arc in CG_Π.*

*$\mathcal{F}$ is a **strict-star factoring** if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ s.t. all arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are incident to F^C. F^C is the **center** of $\mathcal{F}$, and each other factor $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ is a **leaf**.*

*$\mathcal{F}$ is a **fork factoring** if the arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are exactly $\{F^C \to F^L \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$; it is an **inverted-fork factoring** if the arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are exactly $\{F^L \to F^C \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.*

In a fork factoring, the only cross-factor interactions consist in the center factor establishing prevail conditions for actions affecting a leaf factor. In an inverted-fork factoring, the only cross-factor interactions consist in leaf factors establishing prevail conditions for actions affecting the center.[3] In a strict-star factoring, both directions of precondition-effect interactions are admitted, plus there may be actions simultaneously affecting the center and a leaf, i. e., whose effect variables intersect both F^C and one $F^L \in \mathcal{F}^L$.

Example 3. *Consider again the causal graphs of our running examples, Figure 2. In the Vanilla example (a), we obtain a fork factoring when grouping $F^C = \{t_A, t_B\}$ and $\mathcal{F}^L = \{\{o\}\}$, and we obtain an inverted-fork factoring when grouping $F^C = \{o\}$ and $\mathcal{F}^L = \{\{t_A\}, \{t_B\}\}$. In the NoEmpty example (b), each of these groupings yields a strict-star factoring (but neither is a fork nor inverted fork as the dependencies go both ways). In the Scaling example (c), the most sensible grouping is the fork factoring with $F^C = \{t\}$ and $\mathcal{F}^L = \{\{o_1\}, \ldots, \{o_n\}\}$: the truck is connected to every other variable so should be in the center.*

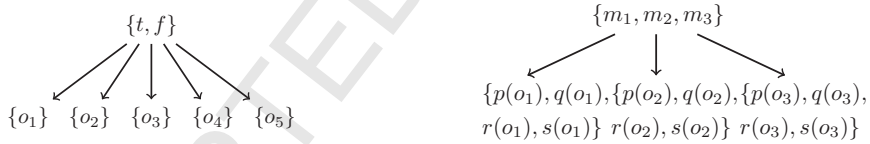


Figure 3: Possible fork factorings in transportation with fuel consumption (left), and production-planning problems with machines m_i processing objects o_j (right).

As a more practical illustration, Figure 3 shows fork factorings on examples similar to AI planning competition (IPC) benchmarks. On the left we consider a transportation domain with fuel consumption (as in the IPC NoMystery benchmark domain). A truck t with fuel supply f transports objects $o_1, \ldots, o_n$; t and f form the center factor, each o_i is a leaf factor on its own.

On the right, we consider a scheduling domain requiring to process a set of objects with a set of machines (similar to the IPC Woodworking benchmark domain). Individual objects o_i are mutually independent except for sharing the machines, so

[3]The fork/inverted-fork terminology here follows Katz and Domshlak [60], who considered similar causal graph structures in the context of tractability analysis.

that the machines are grouped into the center and each leaf factor groups together the properties pertaining to one o_i.

We use strict-star factorings, and the forks and inverted forks sub-cases, in practice, and we will use them throughout the paper in illustrations and practical discussions. This notwithstanding, our techniques are defined for, and work correctly on, more general structures.

Decoupled search branches on all actions affecting the center, i. e., these are included into the center paths being searched over. Therefore, these actions can be arbitrary. In particular, they can affect and/or rely on multiple leaves. The only restriction we require, thus, is that the other actions – those that do not affect the center – are limited to a single leaf factor each:

Definition 2 (General Star Factoring). *Let Π be a planning task, and let $\mathcal{F}$ be a factoring. $\mathcal{F}$ is a **star factoring** if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ such that, for every action a where $\mathcal{V}(\mathsf{eff}(a)) \cap F^C = \emptyset$, there exists $F \in \mathcal{F}$ with $\mathcal{V}(\mathsf{eff}(a)) \subseteq F$ and $\mathcal{V}(\mathsf{pre}(a)) \subseteq F \cup F^C$. F^C is the **center** of $\mathcal{F}$, and all other factors $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ are **leaves**.*

This restriction on non-center-affecting actions obviously holds in a strict-star factoring. However, a center-affecting action in a star factoring may affect multiple leaves. In other words, strict-star factorings are a special case of star factorings:

Proposition 1. *Let Π be a planning task. Then every strict-star factoring is a star factoring, but not vice versa.*

Observe that Definition 2 can always be enforced without loss of generality, simply by introducing redundant effects on F^C. However, the actions affecting F^C are those the decoupled search branches over, so this transformation is just another way of saying that "cross-leaf preconditions/effects can be tackled by centrally branching over the respective actions".

General star factorings as per Definition 2 cannot be characterized in terms of just the causal graph. If there is a causal-graph arc between two leaves, we cannot distinguish whether or not all responsible actions also affect the center. In contrast, strict-star factorings *can* be characterized in terms of just the causal graph, making them easier to identify. Also, as we shall see next, they already are quite powerful, motivating their use in practice.

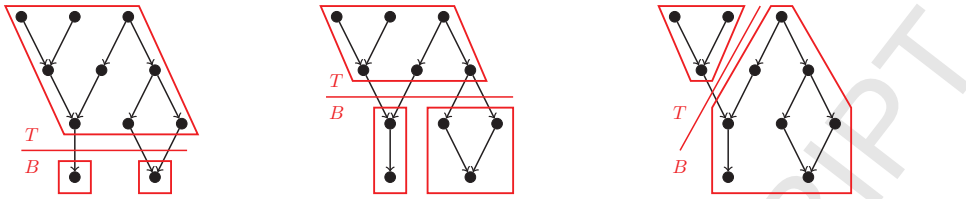


Figure 4: Illustration of Theorem 1: Characterizing fork factorings in terms of "horizontal lines" $\{T, B\}$ through the DAG of causal graph SCCs.

3.2. The Space of Strict-Star Factorings

To design an automatic factoring strategy, we need to know under which conditions a strict-star factoring exists, and if so, how to find one with a maximal number of leaves.

If a factoring $\mathcal{F}$ has K leaves, we say it is a K**-leaves factoring**. By the **maximum number of strict-star (fork/inverted-fork) leaves**, we refer to the number of leaves in a strict-star (fork/inverted-fork) factoring which maximizes that number.

Let us first consider the simple fork and inverted fork special cases. Denote by $\mathcal{F}_\Pi^{\text{SCC}}$ the factoring whose factors are the strongly connected components (SCC) of the causal graph CG_Π. Clearly, any fork or inverted-fork factoring $\mathcal{F}$ must be coarser than $\mathcal{F}_\Pi^{\text{SCC}}$, i. e., for every $F \in \mathcal{F}_\Pi^{\text{SCC}}$ we must have $F' \in \mathcal{F}$ with $F \subseteq F'$. As an immediate consequence, if the causal graph is strongly connected, $|\mathcal{F}_\Pi^{\text{SCC}}| = 1$, then we cannot obtain a factoring with more than one component, so no fork or inverted-fork factoring exists.

The opposite is also true: if $|\mathcal{F}_\Pi^{\text{SCC}}| > 1$, then fork and inverted-fork factorings exist. To see this, consider the interaction graph $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ over causal graph SCCs. This is a directed acyclic graph (DAG), and we can directly read off the maximum number of fork/inverted-fork leaves:

Theorem 1 (Fork & Inverted-Fork Factorings). *Let Π be a planning task. Then fork and inverted-fork factorings exist if and only if $|\mathcal{F}_\Pi^{\text{SCC}}| > 1$. In that case, the maximum number of fork leaves equals the number of leaf vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$, and the maximum number of inverted-fork leaves equals the number of root vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$.*

PROOF SKETCH: The "only if" in the first part of the claim has already been argued. The "if" direction follows from the second part of the claim. To see why that latter part holds true for fork factorings (the argument for inverted forks is symmetric), observe that, as illustrated in Figure 4, any fork factoring $\mathcal{F}$ can be

viewed as "drawing a horizontal line" through $\mathsf{IG}_\Pi(\mathcal{F}^{\text{SCC}}_\Pi)$ where the roots are at the top and the leaves are at the bottom. Denote by T the top part above the line, and by B the bottom part. A fork factoring is then obtained by taking T to be the center, and taking the set $\mathcal{W}$ of weakly connected components of $\mathsf{IG}_\Pi(\mathcal{F}^{\text{SCC}}_\Pi)$ within B to be the leaves. Clearly, $|\mathcal{W}|$ is at most the number of leaf vertices in $\mathsf{IG}_\Pi(\mathcal{F}^{\text{SCC}}_\Pi)$. Vice versa, drawing the line just above the leaf vertices, we obtain a fork factoring with exactly that number of leaf factors. □

Consider now the more general strict-star factorings. Observe first that these do not need to respect causal graph SCCs. The NoEmpty example, cf. Figure 2, has a single-SCC causal graph, but we can factorize it as pointed out in Example 3. Indeed, for any planning task, any partition of the variables into two non-empty subsets yields a strict-star factoring (where we are free to choose which factor is the center respectively the single leaf). So, for a task with variables V, there are at least $2^{|V|} - 2$ strict-star factorings. That wealth of factorings is, however, of unclear practical value, as they have only a single leaf.

It turns out that the number of strict-star leaves is characterized exactly by independent sets in the causal graph, i. e., subsets of variables with no CG_Π arcs between them:

Lemma 1. *Let Π be a planning task. Then from any size-K independent set in CG_Π one can construct a K-leaves strict-star factoring, and vice versa.*

Proof: Say Π has variables V. From left to right, let $I = \{v_1, \ldots, v_K\} \subseteq V$ be an independent set in CG_Π. Consider the factoring $\mathcal{F} = \{V \setminus I, \{v_1\}, \ldots, \{v_K\}\}$. Designating $V \setminus I$ as the center, $\mathcal{F}$ is a strict-star factoring because the $\mathsf{IG}_\Pi(\mathcal{F})$ arcs over pairs of leaf factors coincide with those of CG_Π over pairs of variables from I, of which by prerequisite there are none.

From right to left, let $\mathcal{F} = \{F^C, F^L_1, \ldots, F^L_K\}$ be a strict-star factoring. Design the variable subset I by picking, from every F^L_i, an arbitrary variable $v_i \in F^L_i$. Then I is an independent set in CG_Π because an arc $v_i \to v_j$ in CG_Π would imply an arc $F^L_i \to F^L_j$ in $\mathsf{IG}_\Pi(\mathcal{F})$. □

Intuitively, to obtain a strict-star topology, we can select the center as an arbitrary borderline separating the leaves. Given this, parts A vs. B of the causal graph can be turned into separate leaves iff they have no direct connection, i. e., there is no arc incident to both A and B. The only case where the maximum number of strict-star leaves is 1 is that where the causal graph is completely connected (every pair of variables has a direct dependency), an extremely rare case in practice.

On the downside, leaf-number maximization is now intractable:

Theorem 2 (Strict-Star Factorings). *Let Π be a planning task. Then the maximum number of strict-star leaves equals the size of a maximum independent set in CG_Π. Given $K \in \mathbb{N}$, it is **NP**-complete to decide whether the maximum number of strict-star leaves is $\geq K$.*

Proof: Immediate from Lemma 1 and **NP**-completeness of Maximum Independent Set [61]. □

One may use Theorem 2 for the design of factoring strategies based on approximations of Maximum Independent Set. For now, we employ the more straightforward approach, using Theorem 1 to efficiently find forks, inverted forks, and an "X-shape" combination thereof.

4. The Decoupled State Space

We next introduce our search reduction method. We assume an input planning task Π and a (general) star factoring $\mathcal{F}$ with center F^C and leaves $\mathcal{F}^L$. To make the exposition accessible, we begin in Section 4.1 by introducing the basic concepts and terminology illustrated with an example, and we provide two example walkthroughs in Section 4.2. Section 4.3 specifies the decoupled state space, as an alternative labeled transition system to search in. Section 4.4 proves correctness, i. e., soundness, completeness, and optimality, relative to the input planning task.

4.1. Concepts and Notations

Consider the Vanilla example (an object o and two trucks t_A, t_B moving along a line of three locations l_1, l_2, l_3), and consider the fork factoring given by $F^C = \{t_A, t_B\}$ and $\mathcal{F}^L = \{\{o\}\}$. We refer to actions affecting the center – in the example, the truck moves – as **center actions**, notation convention a^C; we denote the set of all center actions by A^C. The search is over **center paths**, sequences π^C of center actions applicable to I when ignoring preconditions on the leaves (in the example there are no such preconditions, so center paths are simply applicable sequences of truck moves). The search begins with the empty center path, $\pi^C = \langle\rangle$.

Each center path in the search ends in a **decoupled state** s. The decoupled state contains a **center state** $\mathsf{center}[s]$, a value assignment to F^C; for $\pi^C = \langle\rangle$, we have $\mathsf{center}[s] = \{t_A = l_1, t_B = l_3\}$. The decoupled state furthermore contains a **pricing function**, a mapping $\mathsf{prices}[s] : S^L \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ from *leaf states* to non-negative numbers, or ∞ to indicate unreachable leaf states. A **leaf state** s^L

is a value assignment to any one leaf $F^L \in \mathcal{F}^L$. $S^L[F^L]$ denotes the set of leaf states of a specific leaf F^L, and $S^L = \bigcup_{F^L \in \mathcal{F}^L} S^L[F^L]$ denotes the set of all leaf states across leaves.

In the example, $S^L = S^L|_{\{o\}} = \{\{o = l_1\}, \{o = l_2\}, \{o = l_3\}, \{o = t_A\}, \{o = t_B\}\}$ is the set of all value assignments to the object-position variable o. The prices in the initial decoupled state s are: $\mathsf{prices}[s](\{o = l_1\}) = 0$, $\mathsf{prices}[s](\{o = t_A\}) = 1$, and $\mathsf{prices}[s](s_L) = \infty$ for all other leaf states s_L. The price of $\{o = l_1\}$ is 0 as this is already true in the original initial state. The other prices correspond to the costs of cheapest π^C*-compliant leaf paths*, as will be defined shortly.

A **leaf action** a^L is an action affecting a leaf $F^L \in \mathcal{F}^L$. The set of all leaf actions is denoted A^L, and for the set of actions affecting a specific F^L we write $A^L[F^L]$. A **leaf path** π^L of F^L is a sequence of $A^L[F^L]$ actions applicable to I when ignoring preconditions on the center. In our case, $F^L = \{o\}$ and a leaf path is any sequence of loads/unloads applicable when ignoring the truck-position preconditions. Note that, in general (though not in our example), the center actions A^C are not disjoint from the leaf actions $A^L[F^L]$. In fact, for general star factorings, a center action a^C may affect multiple leaves. All center actions will be handled as part of the center search. Therefore, we define **center path cost** as the summed-up cost of the path's actions, while we define **leaf path cost** as the summed-up cost of only the path's *non-center*, $A^L[F^L] \setminus A^C$, actions.

A leaf path π^L of leaf F^L **complies** with a center path π^C, π^L is π^C**-compliant**, if: (1) the subsequences of $A^L[F^L] \cap A^C$ actions in π^L and π^C coincide; and (2) the $A^L[F^L] \setminus A^C$ actions in π^L can be scheduled alongside π^C so that (a) for the actions in π^L all preconditions on F^C are satisfied, and (b) for the actions in π^C all preconditions on F^L are satisfied. In our present example, (1) is moot because $A^L[F^L] \cap A^C$ is empty, and (2b) is moot because center actions do not have leaf preconditions. But (2a) matters because the object moves (load/unload) rely on center preconditions. As we show in Section 4.3, the leaf paths complying with a given center path π^C can be easily maintained in the form of a layered compliant-path graph (with layers corresponding to the steps along π^C).

Coming back to our pricing function $\mathsf{prices}[s]$, the price of $\{o = l_1\}$ is 0 because the empty leaf path $\pi^L = \langle\rangle$ complies with π^C. The price of $\{o = t_A\}$ is 1 because the leaf path $\pi^L = \langle \mathrm{load}(t_A, l_1)\rangle$ complies with π^C: the only center precondition of π^L, $t_A = l_1$, is satisfied in the center state $\mathsf{center}[s] = \{t_A = l_1, t_B = l_3\}$. For the leaf state $\{o = t_B\}$, however, there is no π^C-compliant leaf path because we would need the center precondition $t_B = l_1$, which is not true anywhere along π^C. In particular, the path $\langle \mathrm{load}(t_B, l_1)\rangle$ is not π^C-compliant for that reason. The path $\langle \mathrm{load}(t_B, l_3)\rangle$ is π^C-compliant, but is not actually a leaf path

because it is not applicable to I: its precondition $o = l_3$ on the leaf itself is not satisfied. Similarly for the leaf states $\{o = l_2\}$ and $\{o = l_3\}$. These leaf states are not reachable given π^C, so their price in s is ∞.

Keep in mind that a pricing function does not represent a commitment, but a set of options, one of which will be committed to later on. In the example, *if* we later on choose to load o into t_A, the cost for doing so will be 1. The commitments will only be made once we reach the goal. Namely, a **decoupled goal state** is one whose center state is **center goal state**, and where every leaf has a finite-price **leaf goal state**. Given a center path π^C leading to a decoupled goal state, we can extract a (global) plan π for the input task by augmenting π^C with π^C-compliant leaf goal paths. In fact, pricing functions allow to extract a plan π *optimal subject to using exactly the center action subsequence* π^C. This is so because every global plan decomposes into a center path augmented with compliant leaf paths, and the pricing functions keep track of the *cheapest* compliant leaf paths.

A variant of decoupled search is obtained by replacing the pricing functions with **reachability functions**, that distinguish only whether a leaf state is reachable ($\mathsf{prices}[s](s_L) < \infty$), or not ($\mathsf{prices}[s](s_L) = \infty$). This allows to preserve completeness, but does not allow to preserve optimality. It is of advantage in practice because reachability functions can be computed more efficiently, and as they make less distinctions so reduce the size of the decoupled state space. Reachability functions are equivalent to pricing functions in the modified task where all leaf action costs are set to 0, so we will specify the more general pricing functions only.

4.2. Example Walkthroughs

We illustrate the workings of decoupled search with two examples, the first continuing our example above, the second pointing out how state space size may be exponentially reduced. Both examples use fork factorings. Two additional examples, illustrating inverted-fork and strict-star factorings, are available in Appendix A.2.

Example 4. *Consider again the Vanilla example with* $F^C = \{t_A, t_B\}$ *and* $\mathcal{F}^L = \{\{o\}\}$. *Denote the empty center path by* π^C_0, *and the corresponding decoupled state (as above) by* s_0. *The outgoing transitions of a decoupled state are given by those center actions whose center precondition is satisfied, and whose precondition on each leaf has a finite price. In* s_0, *these actions are move*(t_A, l_1, l_2) *and move*(t_B, l_3, l_2).

Say we choose move(t_A, l_1, l_2)*. Denote the extended center path by* π_1^C*, and the outcome decoupled state by* s_1*. Then* $\mathsf{center}[s_1] = \{t_A = l_2, t_B = l_3\}$*. The prices* $\mathsf{prices}[s_1]$ *are* 0 *for* $\{o = l_1\}$ *and* 1 *for* $\{o = t_A\}$*, as these were the prices in* s_0*, and no cheaper compliant paths become available in* s_1*: in forks, prices can only decrease along a center path (this is not so for non-forks). The only change in* $\mathsf{prices}[s_1]$ *is that* $\{o = l_2\}$ *gets price* 2*, accounting for the leaf path* $\langle$*load*(t_A, l_1)*,* *unload*$(t_A, l_2)\rangle$*, which is* π_1^C*-compliant because it can be scheduled alongside* π_1^C *in the form* $\langle$*load*(t_A, l_1)*,* *move*(t_A, l_1, l_2)*,* *unload*$(t_A, l_2)\rangle$*.*

Now say we obtain π_2^C *and* s_2 *from* π_1^C *and* s_1 *by moving* t_B *to* l_2*. Then* $\mathsf{center}[s_2] = \{t_A = l_2, t_B = l_2\}$*, and we have the new finite price* $\mathsf{prices}[s_2](\{o = t_B\}) = 3$ *thanks to the* π_2^C*-compliant path* $\pi_2^L := \langle$*load*(t_A, l_1)*,* *unload*(t_A, l_2)*,* *load*$(t_B, l_2)\rangle$*. Say further that we obtain* π_3^C *and* s_3 *from* π_2^C *and* s_2 *by moving* t_B *back to* l_3*. Then* $\mathsf{prices}[s_3](\{o = l_3\}) = 4$ *thanks to the* π_3^C*-compliant path* π_3^L *extending* π_2^L *with* *unload*(t_B, l_3)*. We have now reached a decoupled goal state. We obtain a global plan by scheduling* π_3^L *alongside* π_3^C*, yielding the* 7*-step global plan that loads* o *onto* t_A*, moves* t_A *to* l_2*, unloads* o*, moves* t_B *to* l_2*, loads* o *onto* t_2*, moves* t_B *to* l_3*, and unloads* o*.*

Say finally that we continue on from this decoupled goal state, obtaining π_4^C *and* s_4 *from* π_3^C *and* s_3 *by moving* t_A *to* l_3*. Then* $\mathsf{center}[s_4] = \{t_A = l_3, t_B = l_3\}$*. The compliant leaf path* π_3^L *supporting* $\{o = l_3\}$ *in* s_3*, to yield the price tag* 4*, is superseded by the new* π_4^C*-compliant (but not* π_3^C*-compliant) path* $\pi_4^L :=$ $\langle$*load*(t_A, l_1)*,* *unload*$(t_A, l_3)\rangle$*. This decreases the price tag to* $\mathsf{prices}[s_4](\{o = l_3\}) = 2$*. Observe that we now get a* 6*-step global plan, i. e., a plan* better *than that of the decoupled goal state* s_3 *we passed through on the way to* s_4*. Intuitively, decoupled goal states have* leaf-goal price tags*, the cost of "buying" compliant leaf goal paths. The center paths account only for the center, not for the leaves, so the larger path costs of decoupled-goal-state descendants may be counteracted by a cheaper leaf-goal price tag, as in this example. We will show in Section 7 how optimal search algorithms can deal with this via a simple transformation.*

Example 5. *Consider the Scaling example, where one truck* t *and* n *objects* o_i *move along a line* $l_1, \ldots, l_m$ *of length* m*, the truck and all objects starting in* l_1*, the goal being to transport all objects to* l_m*. The standard state space has* $m(m+1)^n$ *reachable states. By contrast, using the fork factoring with* $F^C = \{t\}$ *and* $\mathcal{F}^L = \{\{o_1\}, \ldots, \{o_n\}\}$*, the decoupled state space has only* $\frac{m(m+1)}{2}$ *reachable decoupled states. This does not even depend on the number of objects.*

Intuitively, the reason is that pricing function changes happen synchronously across all leaves, as a function of truck moves, to the effect that we need to keep

track only of the subsequence of locations visited by the truck. In detail: The initial decoupled state allows each object to be loaded, so each has the price tags 0 *for* $\{o_i = l_1\}$ *and* 1 *for* $\{o_i = t_A\}$*. After moving to* l_2*, each object gets the additional price tag* 2 *for* $\{o_i = l_2\}$*. Now there are two choices, moving back to* l_1 *which yields the same center state as before but with the additional* $\{o_i = l_2\}$ *price tags, or moving ahead to* l_3 *which yields the new price tags* 2 *for* $\{o_i = l_3\}$*. In this manner, for each location* l_i*, the new decoupled states are the single one where the truck reaches* l_i *for the first time, plus the* $i - 1$ *ones reached by going back to* $l_{i-1}, \ldots, l_1$*. This yields the overall count* $\sum_{i=1}^{m} i = \frac{m(m+1)}{2}$*.*

Each decoupled state has size $1 + n(m + 1)$ *(truck position;* n *objects, each with* $m + 1$ *leaf states), so the overall state-space representation size is* $O(nm^3)$*.*

Remarkably, while the Scaling example is trivial, as we will detail in Section 6 it exponentially separates star-topology decoupling from most previous search reduction methods.

4.3. Specifying the Transition System

The decoupled state space is a labeled transition system over decoupled states. To compute the pricing functions, we maintain a compliant-path graph for each leaf. That graph represents exactly the compliant leaf paths. Namely, given a center path π^C, the π^C*-compliant-path graph* for a leaf F^L is a layered graph over time-stamped F^L leaf states, where the time steps t correspond to the center states along π^C, as follows:

Definition 3 (Compliant-Path Graph). *Let* Π *be a planning task,* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$*, and* $\pi^C = \langle a_1^C, \ldots, a_n^C \rangle$ *a center path traversing center states* $\langle s_0^C, \ldots, s_n^C \rangle$*. The* π^C***-compliant-path graph*** *for a leaf* $F^L \in \mathcal{F}^L$*, denoted* $\mathsf{CompG}_\Pi[\pi^C, F^L]$*, is the arc-labeled weighted directed graph whose vertices are* $\{s_t^L \mid s^L \in S^L[F^L], 0 \leq t \leq n\}$*, and whose arcs are:*

- *(i)* $s_t^L \xrightarrow{a^L} s'^L_t$ *with weight* $c(a^L)$ *whenever* $s^L, s'^L \in S^L[F^L]$ *and* $a^L \in A^L[F^L] \setminus A^C$ *are such that* $s_t^C[\mathcal{V}(\mathsf{pre}(a^L)) \cap F^C] = \mathsf{pre}(a^L)[F^C]$*,* $s^L[\mathcal{V}(\mathsf{pre}(a^L)) \cap F^L] = \mathsf{pre}(a^L)[F^L]$*, and* $s^L[\![a^L]\!] = s'^L$*.*
- *(ii)* $s_t^L \xrightarrow{0} s'^L_{t+1}$ *with weight* 0 *whenever* $s^L, s'^L \in S^L[F^L]$ *are such that* $s^L[\mathcal{V}(\mathsf{pre}(a_t^C)) \cap F^L] = \mathsf{pre}(a_t^C)[F^L]$ *and* $s^L[\![a_t^C]\!] = s'^L$*.*

Item (i) concerns transitions *within* each time step t, i. e., the graph captures how F^L – *only* F^L, not the center or anything else – can be moved given the center preconditions provided by s_t^C. Recall here that the center actions A^C are not disjoint from the leaf actions $A^L[F^L]$. Within time steps, we consider the leaf-only actions, $A^L[F^L] \setminus A^C$.

Item (ii) concerns transitions *across* time steps, from t to $t+1$, compliant with the center action a_t^C. Leaf states s_t^L at step t survive (have a transition to $t+1$) only if they comply with the leaf precondition required by a_t^C, and they get transformed to leaf states s'^L_{t+1} at step $t+1$ through the effect of a_t^C on F^L. Note that, if a_t^C has no precondition on F^L (e. g., in a fork), then all leaf states s_t^L survive. If a_t^C has no effect on F^L (e. g., in a fork or inverted fork), then the surviving leaf states remain the same at $t+1$, i. e., the item (ii) transitions have the form $s_t^L \xrightarrow{0} s_{t+1}^L$.

The arc weights capture the cost incurred for F^L, i. e., the cost of the π^C-compliant leaf paths of F^L. Within time steps, for the actions moving only F^L, these are just the action costs. Across time steps, where the center moves (which may or may not move F^L as a side effect), the arc weight is 0 because these costs are accounted for on the center path itself.

In short, $\mathsf{CompG}_\Pi[\pi^C, F^L]$ captures all ways in which leaf paths for F^L can be scheduled alongside Π^C. The π^C-compliant paths correspond exactly to the paths from $I[F^L]_0$, i. e., from the vertex representing F^L's initial state in $\mathsf{CompG}_\Pi[\pi^C, F^L]$, to the vertices of the last layer n, where n is the length of π^C. Consequently, we will define the pricing function for π^C and F^L through the *cheapest* such paths in $\mathsf{CompG}_\Pi[\pi^C, F^L]$.

Example 6. *Consider the Vanilla example with fork factoring* $F^C = \{t_A, t_B\}$ *and* $\mathcal{F}^L = \{\{o\}\}$*, and the center path* $\pi^C = \langle \mathit{move}(t_A, l_1, l_2)\rangle$*. The* π^C*-compliant-path graph for the leaf* $F^L = \{o\}$ *is shown in Figure 5.*

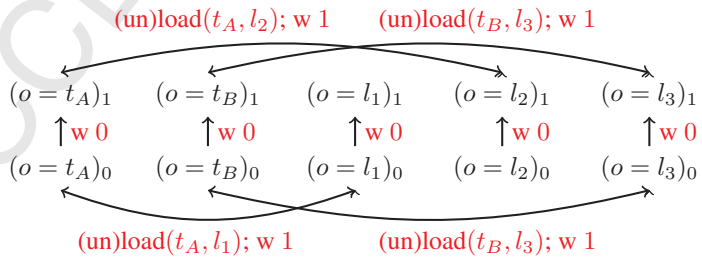


Figure 5: The compliant-path graph for $\pi^C = \langle \mathrm{move}(t_A, l_1, l_2)\rangle$ in the Vanilla example.

The π^C*-compliant leaf path* $\langle \mathit{load}(t_A, l_1), \mathit{unload}(t_A, l_2)\rangle$ *in this graph starts at* $I[F^L]_0$*, i. e., the vertex* $(o = l_1)_0$*. It follows the arc labeled* $\mathit{load}(t_A, l_1)$ *to* $(o =$

$t_A)_0$, *follows the* 0*-arc to* $(o = t_A)_1$, *and follows the arc labeled unload*(t_A, l_2) *to* $(o = l_2)_1$. *The non-compliant leaf path* $\langle$*load*(t_A, l_1), *unload*(t_A, l_2), *load*(t_A, l_2), *unload*$(t_A, l_1)\rangle$ *is not present in the graph as the arc (un)load*(t_A, l_2) *appears only at* $t = 1$, *while unload*(t_A, l_1) *is not available anymore at* $t = 1$.

The pricing function, corresponding to the distances of the $t = 1$ *vertices from the initial-state vertex* $(o = l_1)_0$, *is* 1 *for* $\{o = t_A\}$, 0 *for* $\{o = l_1\}$, 2 *for* $\{o = l_2\}$, *and* ∞ *elsewhere.*

Note that $\mathsf{CompG}_\Pi[\pi^C, F^L]$ *contains redundant parts, not reachable from the initial-state vertex* $(o = l_1)_0$. *This is just to keep Definition 3 simple. In practice, it suffices to maintain the reachable part of* $\mathsf{CompG}_\Pi[\pi^C, F^L]$.

Consider now the NoEmpty example with the same factoring $F^C = \{t_A, t_B\}$ *and* $\mathcal{F}^L = \{\{o\}\}$ *(now a strict-star factoring). Say again that* $\pi^C = \langle move(t_A, l_1, l_2)\rangle$. *The* π^C*-compliant-path graph for* $F^L = \{o\}$ *is shown in Figure 6.*

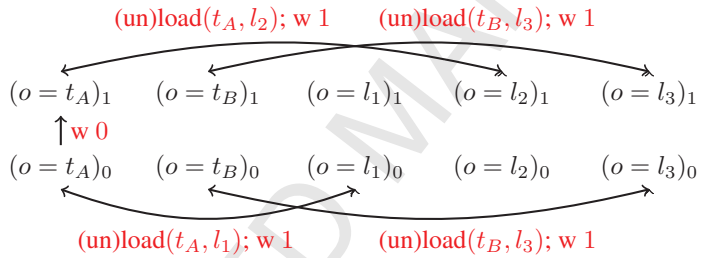


Figure 6: The compliant-path graph for $\pi^C = \langle \text{move}(t_A, l_1, l_2)\rangle$ in the NoEmpty example.

Note the (only) difference to Figure 5: From time 0 *to time* 1, *the only arc we have now is that from* $(o = t_A)_0$ *to* $(o = t_A)_1$. *This is because move*(t_A, l_1, l_2) *now has the precondition* $o = t_A$. *All other values of* o *do not comply with the center action being applied at this time step, and are excluded from the compliant paths. Consequently, the pricing function now is* ∞ *for* $\{o = l_1\}$.

We are now ready to define the decoupled state space:

Definition 4 (Decoupled State Space). *Let* $\Pi = (V, A, c, I, G)$ *be a planning task, and* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$. *A* ***decoupled state*** s *is a triple* $(\pi^C[s], \mathsf{center}[s], \mathsf{prices}[s])$ *where* $\pi^C[s]$ *is a center path,* $\mathsf{center}[s]$ *is a center state, and* $\mathsf{prices}[s]$ *is a* ***pricing function****,* $\mathsf{prices}[s] : S^L \mapsto \mathbb{R}^{0+} \cup \{\infty\}$, *mapping each leaf state to a non-negative* ***price****. The* ***decoupled state space*** *is a labeled transition system* $\Theta^{\mathcal{F}}_\Pi = (S^{\mathcal{F}}, A^C, c|_{A^C}, T^{\mathcal{F}}, I^{\mathcal{F}}, S^{\mathcal{F}}_G)$ *as follows:*

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

(ii) The transition labels are the center actions A^C.

(iii) The cost function is that of Π*, restricted to* A^C.

(iv) $T^{\mathcal{F}}$ *contains a transition* $(s \xrightarrow{a^C} t) \in T^{\mathcal{F}}$ *whenever* $a^C \in A^C$ *and* s, t *are such that:*

1. $\pi^C[s] \circ \langle a^C \rangle = \pi^C[t]$*;*
2. $\mathsf{center}[s][\mathcal{V}(\mathsf{pre}(a^C)) \cap F^C] = \mathsf{pre}(a^C)[F^C]$*;*
3. $\mathsf{center}[s][\![a^C]\!] = \mathsf{center}[t]$*;*
4. *for every* $F^L \in \mathcal{F}^L$ *where* $\mathcal{V}(\mathsf{pre}(a^C)) \cap F^L \neq \emptyset$*, there exists* $s^L \in S^L[F^L]$ *s.t.* $s^L[\mathcal{V}(\mathsf{pre}(a^C)) \cap F^L] = \mathsf{pre}(a^C)[F^L]$ *and* $\mathsf{prices}[s](s^L) < \infty$*; and*
5. *for every leaf* $F^L \in \mathcal{F}^L$ *and leaf state* $s^L \in S^L[F^L]$*,* $\mathsf{prices}[t](s^L)$ *is the cost of a cheapest path from* $I[F^L]_0$ *to* s^L_n *in* $\mathsf{CompG}_\Pi[\pi^C[t], F^L]$*, where* $n := |\pi^C[t]|$.

(v) $I^{\mathcal{F}}$ *is the* ***decoupled initial state****, where* $\mathsf{center}[I^{\mathcal{F}}] := I[F^C]$*,* $\pi^C[I^{\mathcal{F}}] := \langle\rangle$*, and, for every leaf* $F^L \in \mathcal{F}^L$ *and leaf state* $s^L \in S^L[F^L]$*,* $\mathsf{prices}[I^{\mathcal{F}}](s^L)$ *is the cost of a cheapest path from* $I[F^L]_0$ *to* s^L_0 *in* $\mathsf{CompG}_\Pi[\langle\rangle, F^L]$.

(vi) $S^{\mathcal{F}}_G$ *are the* ***decoupled goal states*** s_G*, where* $\mathsf{center}[s_G]$ *is a center goal state and, for every* $F^L \in \mathcal{F}^L$*, there exists a leaf goal state* $s^L \in S^L[F^L]$ *s.t.* $\mathsf{prices}[s_G](s^L) < \infty$.

Note that $\Theta^{\mathcal{F}}_\Pi$ is infinite, for two reasons: (1) decoupled states contain center paths, of which there are infinitely many unless the center cannot move in circles; (2) there are infinitely many pricing functions. We show in the next section that this finitess can be attained by a simple *dominance pruning* method.

We refer to paths $\pi^{\mathcal{F}}$ in $\Theta^{\mathcal{F}}_\Pi$ as **decoupled paths**. The notions of path cost and solutions, for decoupled states as well as for $\Theta^{\mathcal{F}}_\Pi$, are inherited from labeled transition systems. However, we also require a specialized notion of cost and optimality, *augmented cost/optimality*, different from the standard additive-cost notion. This is because, when applied to $\Theta^{\mathcal{F}}_\Pi$, the standard notion accounts only for the center-action costs. Augmented cost/optimality accounts also for the leaf-goal price tag, and as we shall see corresponds exactly to optimality in the input planning task.

Definition 5 (Augmented Cost & Optimality). *Let Π be a planning task, and $\mathcal{F}$ a star factoring. For a decoupled goal state s_G, its **leaf-goal price tag**, denoted $\mathsf{LGPrice}(s_G)$, is the sum over its minimal leaf goal state prices:*

$$\mathsf{LGPrice}(s_G) := \sum_{F^L \in \mathcal{F}^L} \min\{\mathsf{prices}[s_G](s^L) \mid s^L \in S^L[F^L], s^L[\mathcal{V}(G) \cap F^L] = G[F^L]\}$$

*For a decoupled path $\pi^{\mathcal{F}}$ ending in s_G, its **augmented cost** is:*

$$\mathsf{AugCost}(\pi^{\mathcal{F}}) := \mathsf{cost}(\pi^{\mathcal{F}}) + \mathsf{LGPrice}(s_G).$$

*A solution for a decoupled state s is **augmented-optimal** if its augmented cost is minimal among all solutions for s.*

In the definition of leaf-goal price tags, recall that each leaf factor may contain several state variables, and hence may have more than one leaf goal state. So we need to minimize over these.

When referring to optimality in decoupled search, from now on we will always mean augmented optimality. To avoid clumsy wording, we will sometimes drop the word "augmented" and simply talk about optimality.

4.4. Correctness

We now prove soundness, completeness, and optimality of $\Theta^{\mathcal{F}}_{\Pi}$ relative to the input planning task Π. We do so via a characterization of decoupled states in terms of their *hypercubes*:

Definition 6 (Hypercube). *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. For a decoupled state s in $\Theta^{\mathcal{F}}_{\Pi}$, a state p in Π is a **member state** of s if $p[F^C] = \mathsf{center}[s]$ and, for all $F^L \in \mathcal{F}^L$, $\mathsf{prices}[s](p[F^L]) < \infty$. The set of all member states of s is the **hypercube** of s, denoted $[s]$.*

In other words, $[s]$ is the product of the reached leaf states across leaf factors. This is naturally viewed as a hypercube whose dimensions are the leaf factors F^L.

Observe that s is a decoupled goal state if and only if its hypercube $[s]$ contains a goal state of Θ_{Π}. Furthermore, it is not difficult to prove that $[s]$ captures exactly those states of Θ_{Π} reachable using the center path $\pi^C[s]$:

Lemma 2. *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. Let s be a reachable decoupled state in $\Theta^{\mathcal{F}}_{\Pi}$. Then:*

(i) *$[s]$ is exactly the set of states p for which there exists a path π, from I to p in Θ_Π, whose center-action subsequence is $\pi^C[s]$.*

(ii) *For every $p \in [s]$, the cost of a cheapest such path π is*

$$\mathsf{cost}(\pi^C[s]) + \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L])$$

Our proof, given in Appendix A.3, is via notions of *embedded states*, and *embedded transitions*, linking member states, and member-state transitions, to decoupled states. Embedded states $\widehat{p}$ are in one-to-one correspondence with member states p, but replace the value assignment to each F^L with the respective vertex in the compliant-path graph for F^L. Embedded transitions $\widehat{p} \xrightarrow{a} \widehat{q}$ capture member state transitions $p \xrightarrow{a} q$ in the compliant-path graphs, and hence in the decoupled state space. One can think about this as capturing the decoupled state space in terms of atomic transition-addition steps.

Lemma 2 follows directly from the correspondence between member state transitions, embedded state transitions, and the decoupled state space. The same correspondence also shows that:

Lemma 3. *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. Let p be a reachable state in Π, and let π be a path reaching p. Then there exists a reachable decoupled state s in $\Theta_\Pi^\mathcal{F}$ so that $p \in [s]$, and $\pi^C[s]$ is the center action subsequence of π.*

The reader familar with state-space *abstractions*, i. e., state-space quotients relative to a state partition, might find hypercubes reminiscent of that method. But these two concepts actually are very different. State-space quotients over-approximate reachability, whereas the hypercubes in $\Theta_\Pi^\mathcal{F}$ capture reachability exactly, in the sense stated by Lemmas 2 and 3. Also, the hypercubes of distinct decoupled states may overlap. We will get back to this in Section 6.4.

Correctness relative to the input planning task Π now follows directly:

Theorem 3 (Soundness, Optimality Subject to Center Path). *Let Π be a planning task, $\mathcal{F}$ a star factoring, and $\pi^\mathcal{F}$ a solution for $\Theta_\Pi^\mathcal{F}$ ending in s. Then there is a plan π for Π where $\mathsf{cost}(\pi) = \mathsf{AugCost}(\pi^\mathcal{F})$, where the center-action subsequence of π is $\pi^C[s]$, and where π is cheapest among all plans for Π sharing that same center-action subsequence.*

Proof: As s is a decoupled goal state, there exists a member goal state in $[s]$. Let p be a cheapest such state, i. e., one that minimizes $\sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L])$. Using Lemma 2, we can obtain a plan π ending in p, where $\mathsf{cost}(\pi) = \mathsf{cost}(\pi^C[s]) + \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L])$. By construction, this cost is equal to $\mathsf{AugCost}(\pi^{\mathcal{F}})$.

Let now π' be any plan using $\pi^C[s]$, ending in goal state q. By Lemma 2 (i), $q \in [s]$. So, by construction, $\sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](q[F^L]) \geq \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L])$. By Lemma 2 (ii) applied to q, $\mathsf{cost}(\pi') \geq \mathsf{cost}(\pi^C[s]) + \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](q[F^L])$. Hence $\mathsf{cost}(\pi') \geq \mathsf{cost}(\pi)$, concluding the argument. □

Theorem 4 (Completeness, Global Optimality). *Let Π be a planning task, and $\mathcal{F}$ a star factoring. If Π is solvable, then so is $\Theta^{\mathcal{F}}_{\Pi}$. If $\pi^{\mathcal{F}}$ is an augmented-optimal solution to $\Theta^{\mathcal{F}}_{\Pi}$, then $\mathsf{AugCost}(\pi^{\mathcal{F}})$ equals the cost of an optimal plan for Π.*

Proof: Assume that Π is solvable. Let π be an optimal plan for Π, ending in goal state p. With Lemma 3, there exists a reachable decoupled state s s.t. $p \in [s]$. Let $\pi^{\mathcal{F}}$ be the decoupled path reaching s from $I^{\mathcal{F}}$. As p is a goal state, s is a decoupled goal state, so $\pi^{\mathcal{F}}$ is a solution for $\Theta^{\mathcal{F}}_{\Pi}$.

With Lemma 2 (ii), we have that $\mathsf{cost}(\pi) \geq \mathsf{cost}(\pi^C[s]) + \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L])$. By definition, $\mathsf{cost}(\pi^C[s]) + \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L]) \geq \mathsf{AugCost}(\pi^{\mathcal{F}})$. Thus $\mathsf{cost}(\pi) \geq \mathsf{AugCost}(\pi^{\mathcal{F}})$, i. e., $\Theta^{\mathcal{F}}_{\Pi}$ has a solution whose augmented cost is at most that of π. The second part of the claim now follows directly with Theorem 3. □

Together, Theorems 3 and 4 show that (augmented-optimal) search in $\Theta^{\mathcal{F}}_{\Pi}$ is a form of (optimal) planning for Π. Furthermore, given a solution $\pi^{\mathcal{F}}$ for $\Theta^{\mathcal{F}}_{\Pi}$, the corresponding plan π for Π as per Theorem 3 can be constructed by selecting, for every leaf factor, a cheapest leaf goal state s^L, and a cheapest path to s^L in the respective compliant-path graph. This construction is low-order polynomial-time in the size of Π and the size of the compliant path graphs along $\pi^{\mathcal{F}}$.

We have already seen, in our Scaling example (Example 5), that the decoupled state space can be exponentially smaller than the standard state space. We next examine possible blow-ups; then we show that our method is exponentially separated from all previous search reduction techniques; then we show how to design (augmented-optimal) search algorithms for $\Theta^{\mathcal{F}}_{\Pi}$.

5. Decoupled State Space Size and Pruning

As we pointed out, the decoupled state space is, in general, infinite due to (1) different center paths, and due to (2) different pricing functions. Of these, (1)

would be easy to tackle by considering decoupled states to be equivalent if they disagree only on the center path. However, as we now show, due to (2) there can be infinitely many reachable non-equivalent decoupled states. Furthermore, even in finite cases, the number of reachable non-equivalent decoupled states can blow up exponentially relative to the standard state space.

Fortunately, both can be tackled using pruning techniques. Finiteness can be achieved through a simple *dominance pruning* technique that we introduce in Section 5.1. Blow-ups can be avoided through a more powerful *hypercube pruning* technique that we introduce in Section 5.2. The latter guarantees that *there can never be more reachable decoupled states than reachable standard states*. The downside of hypercube pruning is that it incurs a **co-NP**-complete subproblem. Our implementation uses dominance pruning only as, empirically on the standard IPC benchmarks, hypercube pruning incurs a large computational overhead; reduces search space size only marginally relative to dominance pruning; and even without hypercube pruning, the number of decoupled states never exceeds that of standard states. We introduce hypercube pruning for its theoretical merit.

5.1. Finiteness and Dominance Pruning

Non-reachable states are not of interest in practice, so we focus on reachable states only. We denote by $\Theta_\Pi^{R\mathcal{F}}$ the subsystem of $\Theta_\Pi^{\mathcal{F}}$ containing only those decoupled states reachable from $I^{\mathcal{F}}$.

For reachability functions, as their number is finite, so is the number of non-equivalent decoupled states in $\Theta_\Pi^{R\mathcal{F}}$ (and even in $\Theta_\Pi^{\mathcal{F}}$). For fork factorings, the number of reachable pricing functions is finite because prices can only decrease as cheaper leaf paths become compliant. In non-fork factorings with pricing functions, however, the center may commit the leaves to provide preconditions, causing the leaf state prices to increase. Repeated commitments can cause the prices to diverge:

Example 7. *Consider the Vanilla example with inverted-fork factoring* $F^C = \{o\}$ *and* $\mathcal{F}^L = \{\{t_A\}, \{t_B\}\}$. *Say we continue the decoupled path* $s_0 \xrightarrow{load(t_B, l_1)} s_1$ *with the center action unload*(t_B, l_1). *Then the outcome decoupled state* s_2 *has the same center state as* s_0, $o = l_1$. *But the pricing function is different. The price tags for* t_B *in* s_0 *are* 0 *for* $\{t_B = l_3\}$, 1 *for* $\{t_B = l_2\}$, *and* 2 *for* $\{t_B = l_1\}$. *In* s_2, *they are* 2 *for* $\{t_B = l_1\}$, 3 *for* $\{t_B = l_2\}$, *and* 4 *for* $\{t_B = l_3\}$. *If we continue loading/unloading the object in alternating locations on the map, then the prices for truck positions will keep increasing ad infinitum.*

Intuitively, the decoupled states along load/unload sequences as outlined get *worse*, as the prices get higher. This leads to the following simple definition of dominance:

Definition 7 (Dominance). *Let Π be a planning task, $\mathcal{F}$ a star factoring, and s, t decoupled states. We say that t **dominates** s if $\mathsf{center}[t] = \mathsf{center}[s]$ and, for every leaf state s^L, $\mathsf{prices}[t](s^L) \leq \mathsf{prices}[s](s^L)$.*

Note that equivalence is a special case of dominance, in that equivalent s and t dominate each other.

Informally, if t dominates s, then anything one can do in s, one can do at least as well in t. Formally, dominance is a simulation relation (e. g. [62]):

Proposition 2 (Correctness of Dominance Pruning). *Let Π be a planning task, $\mathcal{F}$ a star factoring, and s, t decoupled states, where t dominates s. Then, for every transition $s \xrightarrow{a^C} s'$ in $\Theta^{\mathcal{F}}_{\Pi}$, $t \xrightarrow{a^C} t'$ also is a transition in $\Theta^{\mathcal{F}}_{\Pi}$, and t' dominates s'.*

Proof: The center precondition of a^C is trivially true in t, and its leaf preconditions have finite prices in t because that is so already in s. Hence $t \xrightarrow{a^C} t'$ is a transition in $\Theta^{\mathcal{F}}_{\Pi}$. To see that t' dominates s', just note that the compliant-path graph is extended with the same last time-step on both sides, so the cheaper prices in t can only lead to cheaper prices in t'. □

Simulation relations have previously been used for dominance pruning in standard state space search (e. g. [63, 64]). In particular, they can be used for optimality-preserving pruning when used only against previously visited states with equal or smaller *path costs*, i. e., reached with action sequences of equal or smaller cost. For the purpose of our discussion in this paper, we distinguish the following three forms of pruning:

1. **Ancestor:** Compare any new state s to its ancestor states t.
2. **Cheaper-Visited:** Compare any new state s to all previously visited states t whose path cost is at most that of s.
3. **All-Visited:** Compare any new state s to all previously visited states t.

In all cases, if s is dominated by one of the states t it is compared to, then s is pruned. As dominance generalizes equivalence, such pruning generalizes duplicate pruning over equivalent states.

Ancestor states are a special case of cheaper visited states, and cheaper visited states are a special case of visited states. Hence the pruning methods become stronger in the order listed. Cheaper-visited pruning preserves optimality [64]; all-visited pruning does not, because the search might first visit a dominating but more costly state, pruning the optimal solutions.

It turns out that, to avoid infinite reachable decoupled state spaces $\Theta_{\Pi}^{R\mathcal{F}}$, the weakest form of pruning considered herein – ancestor dominance pruning – already suffices.

For inverted-fork factorings, this is trivial: As the prices increase monotonically, with ancestor dominance pruning every search path must stop as soon as the same center state is visited a second time. For example, the search path $s_0 \xrightarrow{\text{load}(t_B,l_1)} s_1 \xrightarrow{\text{unload}(t_B,l_1)} s_2$ in Example 7 is pruned, because s_2 is dominated by s_0.

For more general factorings, with bidirectional dependencies between leaves and center, where the prices neither increase nor decrease monotonically, the proof is more involved:

Theorem 5 (Finiteness under Dominance Pruning). *Let Π be a planning task, and $\mathcal{F}$ a star factoring. Under ancestor dominance pruning, $\Theta_{\Pi}^{R\mathcal{F}}$ is finite.*

PROOF SKETCH: Consider the non-pruned paths $\pi^{\mathcal{F}}$ in $\Theta_{\Pi}^{R\mathcal{F}}$. We prove that, under ancestor dominance pruning, every $\pi^{\mathcal{F}}$ is finite. As the branching factor is finite, this proves the claim.

Observe that the non-pruned paths necessarily are *descending*: Intuitively, some prices along $\pi^{\mathcal{F}}$ must descend in each step as otherwise the new state would be dominated by some previous state. Formally, consider the pricing functions along $\pi^{\mathcal{F}}$ as vectors v over $\mathbb{R}^{0+} \cup \{\infty\}$. Define a relation $\succ$ over such vectors by $v \succ v'$ iff there exists a vector position k so that $v[k] > v'[k]$. Say that a vector sequence $\vec{v} = v_0, v_1, v_2, \ldots$ is *descending* if, whenever v precedes v' in the sequence, $v \succ v'$.

Assume to the contrary that there is an infinite descending path $\pi^{\mathcal{F}}$. Then from an infinite sub-path with identical center states we can extract an infinite descending vector sequence $\vec{p}$ over $\mathbb{R}^{0+} \cup \{\infty\}$. More precisely, $\vec{p}$ is over $R \cup \{\infty\}$ where R contains all possible finite action-cost sums in Π.

It is easy to see (Proposition 5 in Appendix A.4) that R has no infinite descending sequence of 1-vectors (within each finite cost value, only a finite number of non-0 cost actions can be used). But what about N-vectors? Observe that, for $N > 1$, the relation $\succ$ is not a partial order. It is neither transitive,

e. g. $(5,5) \succ (4,10)$ and $(4,10) \succ (5,9)$, but $(5,5) \not\succ (5,9)$; nor asymmetric, e. g. $(4,5) \succ (5,4)$ and $(5,4) \succ (4,5)$. Furthermore, $v \succ v'$ as soon as v' is strictly smaller anywhere; all other positions can increase by arbitrary amounts, e. g. $v \succ v'$ for $v = (5,5)$ and $v' = (4,1000)$. Nonetheless (Lemma 8 in Appendix A.4), if there is no infinite descending sequence of 1-vectors over some set $R \subseteq \mathbb{R}$, then there is no infinite descending sequence of N-vectors over $R \cup \{\infty\}$. Hence $\vec{p}$ must be finite, in contradiction, showing the claim. □

On fork factorings, in practice we employ an optimization of dominance pruning, that we refer to as *frontier pruning*. Given a decoupled state s and leaf factor F^L, a leaf state s^L is a *frontier state* if either (a) s^L is a leaf goal state; or (b) s^L can still contribute to improving the price of one of its neighbors, i. e., there exist a and t^L so that $s^L \xrightarrow{a} t^L$ and $\mathsf{prices}[s](s^L) + c(a) < \mathsf{prices}[s](t^L)$. Intuitively, as prices in a fork factoring can only decrease, frontier states are the only ones that may still have a role in obtaining the cheapest goal price. Hence one can test dominance on frontier states only: t frontier-dominates s if $\mathsf{center}[t] = \mathsf{center}[s]$ and, for every frontier leaf state s^L, $\mathsf{prices}[t](s^L) \leq \mathsf{prices}[s](s^L)$. This still preserves optimality. (The easy proof can be looked up in the work by Torralba et al. [3], who introduce frontier pruning as their most basic method.)

5.2. Size Blow-Up and Hypercube Pruning

Even in finite cases, the number of non-equivalent decoupled states can blow-up relative to the size of the standard state space. This is because the maintenance of pricing functions can result in a form of "memory of the center path taken": different paths may have provided center preconditions enabling different leaf moves, thus resulting in different pricing functions. This happens, indeed, even in very simple examples akin to the transportation examples we have been using all along:

Example 8. *Consider the following* BlowUp *example, with one truck t, one object o, and n locations $l_1, \ldots, l_n$, where the truck can move between any pair of locations; both t and o are initially at l_1, the goal is for o to be at l_n. In short, this is like the Scaling example except with only a single object, and with locations arranged as a fully-connected road map instead of a line. We consider the fork factoring $F^C = \{t\}$, $\mathcal{F}^L = \{\{o\}\}$.*

Obviously, the standard state space in this example is small ($n(n+1)$ reachable states). But the decoupled state space has size exponential in n. Through the pricing functions, the decoupled states "remember" the locations visited by t in the past. For example, the decoupled state reached through $\langle drive(l_1, l_2),$

drive$(l_2, l_3)\rangle$ *has finite prices for* $\{o = l_1\}$, $\{o = t\}$, $\{o = l_2\}$, *and* $\{o = l_3\}$; *while the decoupled state reached through* $\langle$*drive*(l_1, l_4), *drive*$(l_4, l_5)\rangle$ *has finite prices for* $\{o = l_1\}$, $\{o = t\}$, $\{o = l_4\}$, *and* $\{o = l_5\}$. *Hence the decoupled state space enumerates pricing functions corresponding to every subset of visited locations from* $\{l_2, \ldots, l_n\}$.

Note the simplicity of this example. Blow-ups may occur even for fork factorings, and even when maintaining reachability functions only (as the pricing functions here disagree even on reachability). The latter also implies that dominance pruning does not prevent the blow-up.[4]

It turns out that blow-ups can be avoided based on the previously introduced hypercube characterization. The core observation in this context is that solvability of a decoupled state is equivalent to solvability of at least one member state:

Lemma 4. *Let* Π *be a planning task,* $\mathcal{F}$ *a star factoring, and* s *a decoupled state. Then* s *is solvable if and only if at least one* $p \in [s]$ *is solvable.*

This follows from the same correspondence between member state transitions, embedded state transitions, and the decoupled state space, as used to prove correctness in Section 4.4.

The idea now simply is to prune a decoupled state s when its hypercube does not contribute any new member states, i. e., if s is *covered* by previously visited decoupled states:

Definition 8 (Hypercube Covering). *Let* Π *be a planning task,* $\mathcal{F}$ *a star factoring, and* $s, t_1, \ldots, t_n$ *decoupled states. We say that* $t_1, \ldots, t_n$ ***cover*** s *if* $\mathsf{center}[t_i] = \mathsf{center}[s]$ *for all* i, *and* $\bigcup_{i=1}^{n} [t_i] \supseteq [s]$.

From Lemma 4, we immediately get:

Theorem 6 (Correctness of Hypercube Pruning). *Let* Π *be a planning task, and* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$. *Let* $s, t_1, \ldots, t_n$ *be decoupled states, where* $t_1, \ldots, t_n$ *cover* s. *If* s *is solvable, then so is at least one* t_i.

[4] We remark that frontier pruning does prevent this blow-up. Further, on transportation-style IPC benchmarks, we never observed any state space size increase even when using only dominance pruning, not frontier pruning. Presumably, the loss from structure like in our BlowUp example (many tightly connected locations) is outweighed by the gain from structure like in our Scaling example (many transportable objects).

Therefore, hypercube pruning is correct: if s is covered by previously visited decoupled states $t_1, \ldots, t_n$, then pruning s does not forego completeness. This applies to the same three forms of pruning as above, i. e., ancestor pruning, cheaper-visited pruning, and all-visited pruning. In each case, the respective collection of t becomes the set $t_1, \ldots, t_n$ in the hypercube-covering check.

On the other hand, hypercube pruning does, as stated, not preserve optimality. The hypercube disregards leaf state *prices*, and so does the notion of hypercube covering.

Given this, cheaper-visited pruning does not make much sense. Ancestor pruning is not enough to prevent exponential blow-ups. However, all-visited pruning provides the desired guarantee:

Theorem 7 (Size Guarantee under Hypercube Pruning). *Let Π be a planning task, and $\mathcal{F}$ a star factoring. Under all-visited hypercube pruning, the number of decoupled states in $\Theta_\Pi^{R\mathcal{F}}$ is bounded by the number of reachable states in Θ_Π.*

Proof: Consider the non-pruned visited decoupled states s under all-visited hypercube pruning. With $t_1, \ldots, t_n$ being the decoupled states visited prior to s and sharing the same center state, denote by $\overline{[s]} := [s] \setminus \bigcup_{i=1}^n [t_i]$ the hypercube remaining of s when eliminating $t_1, \ldots, t_n$. Clearly, $\overline{[s]} \neq \emptyset$ as otherwise s would be pruned. Furthermore, for any other non-pruned visited decoupled state s', $\overline{[s]} \cap \overline{[s']} = \emptyset$: if s' was visited before s, this is because $[s']$ was eliminated from $\overline{[s]}$; if s' was visited after s, this is because $[s]$ was eliminated from $\overline{[s']}$. So the non-pruned decoupled states are associated with non-empty and disjoint sets $\overline{[s]}$ of states in Θ_Π. The claim now follows together with Lemma 2, which implies that, if s is reachable in $\Theta_\Pi^{\mathcal{F}}$, then every $p \in [s]$ is reachable in Θ_Π. □

The bad news is that testing Definition 8 is, in general, hard, even for the hypercubes that may actually occur during decoupled search:

Proposition 3. *Given a planning task Π and a star factoring $\mathcal{F}$, it is **co-NP**-complete to decide whether reachable decoupled states $t_1, \ldots, t_n$ cover a reachable decoupled state s.*

PROOF SKETCH: Membership follows directly from the results by Hoffmann and Kupferschmid [65] for general hypercube covering problems. Hardness follows by reduction from the complement of SAT, extending Hoffmann and Kupferschmid's argument by a simple construction of Π and $\mathcal{F}$. Given any CNF formula ϕ with clauses $C_1, \ldots, C_m$, the construction includes, for each clause C_j, a center

action a_j^C which is applicable to the initial state, and which allows to generate a hypercube t_j corresponding to the truth-value assignments disallowed by C_j. Another center action a_0^C allows to generate the hypercube s corresponding to the space of all truth-value assignments. Consider the time point in search where search has explored each of the alternatives $a_1^C, \ldots, a_m^C$ (applied each of these actions to the initial state separately), and now explores the alternative a_0^C. Then all-visited hypercube pruning checks whether $t_1, \ldots, t_m$ cover s. The latter is the case iff ϕ is unsatisfiable. □

We remark that $\mathcal{F}$ in the proof construction is a fork factoring, so **co-NP**-completeness holds for this restricted case already.

Hoffmann and Kupferschmid [65] devise an algorithm, "cube elimination", to solve hypercube covering problems. We implemented this algorithm, but found it to be ineffective in decoupled search, where it consumes prohibitive amounts of memory. We also experimented with an encoding into SAT, and a SAT solver API (Z3 [66]), but that was ineffective too, consuming prohibitive amounts of runtime. It remains an open question whether hypercube pruning can be made effective in practice. In particular, this pertains to optimality-preserving variants. (A simple such variant results from reducing each t_i to only those leaf states whose price is at most that in s.)

6. Related Techniques and Exponential Separations

Several previous methods relate to star-topology decoupling in that they may sometimes exploit similar structure. All these methods are complementary to star-topology decoupling, i. e., they do not dominate star-topology decoupling, nor vice versa. In particular, there are *exponential separations*: examples where star-topology decoupling results in exponentially less effort.

We discuss *partial-order reduction (POR)* methods in Section 6.1, *unfolding* methods in Section 6.2, and *factored planning* methods in Section 6.3. We summarize the relation to other methods in Section 6.4.

6.1. Partial-Order Reduction

POR methods prune applicable transitions, preserving completeness while avoiding unnecessary permutations. POR originates in Verification (e. g. [8, 9, 10, 11, 12, 13]), and has more recently also been successfully applied in classical planning, specifically in the form of *strong stubborn sets* [15, 16].

Star topologies relate to POR in that they can also be directly exploited for POR. In a fork topology, where the only cross-factor interactions consist of leaf

actions requiring preconditions on the center, one can fix an ordering of leaves, and schedule transitions in round-robin blocks "center – first leaf – … – last leaf", where each factor may choose to move or stand still, but if leaf F^L chooses to stand still then it has to remain so until the center moves in a way that enables a different transition for F^L.

Nevertheless, star-topology decoupling is exponentially separated from POR methods. Specifically, here and in the discussions below, by an **exponential separation** from X, where X is an alternate search reduction method, we mean a family $\{\Pi_n \mid n \in \mathbb{N}^+\}$ of planning tasks, of size (number of state variables and actions) polynomially related to n, so that (i) the number of reachable decoupled states is bounded by a polynomial in n, while (ii) the state space size under X is exponential in n. In most cases, the exponential separation will be our Scaling example, fixing, for simplicity, the number of locations to be the same as the number n of objects o_i to transport. Recall from Example 5 that condition (i) is satisfied (there are $\frac{n(n+1)}{2}$ reachable decoupled states), while the standard state space has $n(n+1)^n$ reachable states.

The Scaling example is an exponential separation from strong stubborn sets. These collect a subset of actions branching over which suffices to preserve optimality in a given state; applicable actions outside the strong stubborn set are pruned. A strong stubborn set must (a) contain actions that make "progress to the goal", as well as (b) contain all actions that *interfere* (that are not *concurrent*) with applicable actions already included into the stubborn set. Now, consider the initial state I in our Scaling example, where the truck and all objects are at l_1. To satisfy (a), it suffices to include a single load(o_i, l_1) action. However, we also need to include move(l_1, l_2), as this (b) interferes with load(o_i, l_1). But then, we must include *all* applicable load actions, $\{\text{load}(o_i, l_1) \mid 1 \leq i \leq n\}$, because these (b) interfere with move(l_1, l_2). The same argument applies, in the outcome state of applying any load(o_j, l_1) in I, to all other load actions $\{\text{load}(o_i, l_1) \mid 1 \leq i \leq n, i \neq j\}$. Hence the state space under strong stubborn set pruning enumerates all subsets of objects that may be loaded in the initial state, resulting in a number of states $\geq 2^n$.

6.2. Petri Net Unfolding

Instead of pruning permutations in the standard state space, *unfoldings* represent the reachable state space as an acyclic Petri net, capturing concurrency explicitly (e. g. [22, 23, 24, 25, 26]). This relates to star-topology decoupling in that both approaches build structures over partial (component) states, without necessarily enumerating their combinations. One can view star-topology decoupling

as a form of unfolding exploiting star topologies to get rid of two major sources of complexity, namely (1) testing whether a conjunctive condition is reached and (2) dealing with prevail conditions. Exponential separations are easy to construct given these complexity reductions; e. g., our Scaling example is one.

Petri net unfolding constructs a DAG, an unfolding *prefix* $\mathcal{P}$, whose vertices are *conditions*, annotated with state-variable values in our context; and *events*, annotated with actions in our context. Starting from $\mathcal{P}$ containing only the initial state conditions, the unfolding process adds *possible events*, and adds their effects as new conditions.

A possible event is one whose precondition is reachable within $\mathcal{P}$. Complexity source (1) arises because deciding whether that is so, i. e., whether a (partial) value assignment (a *marking*) has non-conflicting support within $\mathcal{P}$, is **NP**-complete given $\mathcal{P}$. Testing this property requires to find a jointly reachable combination of conditions annotated with the required values. This needs to be done every time a new event should be added, and, once the prefix is completed, when testing whether the goal is reachable. In contrast, such reachability testing is linear-time in the decoupled state space, whose organization via center paths avoids cross-leaf conflicts.

Regarding complexity source (2), prevail conditions are an important feature in our setting, specifically leaf actions with prevail conditions on the center. Consider for example the load actions in our examples, which require, but do not consume, a truck position $t = l$. The decoupled state space does not have to branch over such actions.

In Petri nets, prevail conditions correspond to *read arcs*, that do not consume their input. Standard Petri nets do not support read arcs. They can be encoded via the *plain* method, where $t = l$ is both an input and an output of every load/unload; or via the *place-replication* method, where the same is done but each load/unload gets its own copy of $t = l$, and a truck move away from l requires all these copies as inputs. Both encodings are quite different from the native treatment of prevail conditions in planning. In particular, in our Scaling example, both encodings result in unfoldings enumerating all subsets of objects that may be loaded at l_1. For the plain encoding, this is because the concurrency between the load(o_i, l_1) events is lost. In the place-replication encoding, adding move(l_1, l_2) to the prefix enumerates the object subsets because, for every copy of $t = l_1$, we can choose whether to use the condition from before, or after, the respective object-load event.

There is a Petri net variant, *contextual* Petri nets (*c-nets*), that natively supports read arcs [27]. The unfolding process for c-nets is like that sketched above, except that, instead of a prefix $\mathcal{P}$, c-net unfolding builds an *enriched prefix* $\mathcal{E} = (\mathcal{P}, \mathcal{H})$,

where $\mathcal{H}$ maps the events in $\mathcal{P}$ to *sets* of histories: the event history needs to distinguish the subset of prevail conditions consumed in the past, so that the unfolding process can determine which new events are still possible in the future. Any one event may have exponentially many different histories. Indeed, the number of event histories in $\mathcal{E}$ is asymptotically identical with the size of the place-replication unfolding [27]. In our Scaling example, the histories for move(l_1, l_2) are exactly the subsets of loaded objects.

Of course, unfolding has advantages too. Any example family whose unfolding is small, but that has no useful star topology, is an exponential separation from star-topology decoupling. Such cases do occur also in the IPC benchmarks. In the Airport IPC domain (an abstract version of airport ground traffic control [67]), unfolding yields strong state space reductions [24]. Yet all airplanes interact directly with each other, so there is no useful star topology. An interesting direction for future work are hybrids between decoupled search and unfolding, trying to combine advantages from both sides.

6.3. Factored Planning

As stated, star-topology decoupling has been inspired by factored planning, which also partitions the state variables into factors [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41].

In *localized* factored planning, the planning process is formulated as local planning on individual components, followed by global cross-component constraint resolution. The latter is done in terms of constraint satisfaction or message-passing methods over exhaustive sets of local plans. Several works have analyzed in depth the worst-case complexity of such planning processes, with major sources of exponential complexity being the length of local plans, and the treewidth of the global constraint graph (the interaction graph, in our terminology). Star-topology decoupling eliminates both of these complexity sources by restricting the form of global interactions allowed.

The most recent, and empirically the most successful, localized factored planning method is *partition-based pruning* [40], which originates from multi-agent planning. The local planning here takes place as part of a state space search, where a local component state space – an agent's "private" part of the search – is pursued exclusively, until "public" state changes (relevant to other components) are encountered. This can be viewed as a form of partial-order reduction, exploiting permutability of local plans across components.

In *hierarchical* factored planning, the factors are used within a hierarchy of increasingly more detailed levels, accumulating the factors processed so far as

search proceeds down the hierarchy. As a high-level plan may not be realizable at lower levels, one needs to allow backtracking across levels. In star-topology decoupling, the maintenance of compliant paths (low-level plans in a 2-level hierarchy with bidirectional dependencies) eliminates that need.

Almost all factored planning approaches cannot guarantee global plan optimality; sometimes, it is a hypothetical but impractical possibility involving exhaustive search over all local plan lengths. The single exception is the work by Fabre et al. [36], which uses finite automata to represent the local-plan languages (the sets of local plans) at any point in the search.

Algorithmically, star-topology decoupling is quite different from both, localized and hierarchical factored planning. At the core of the differences is the assumption of a particular structural profile, a star topology, in our method; vs. the handling of arbitrary cross-factor interactions in previous methods. Consequently, whereas previous works concentrate on the resolution of complex interactions – via constraint satisfaction, tree decomposition, message passing, backtracking – our work concentrates on algorithms specialized to star topologies, which facilitate a much different direct handling of the simple interactions between center and leaves. In particular, thanks to the latter, our algorithms end up being like state space search, only on a more complex state structure; this is not the case of any previous factored planning method (save partition-based pruning which is a partial-order reduction technique). The only strong conceptual commonality between star-topology decoupling and factored planning thus remains the use of a state variable partition.

Turning our attention to exponential separations, consider first localized factored planning. It is easy to see that our Scaling example is an exponential separation from partition-based pruning, as the only private actions are those of the truck, which does not affect the exponential behavior in the number n of packages. For most other localized factored planning approaches, exponential separations result from their scaling exponentially in local plan length [32, 33, 35, 39]. For example, one can modify our Scaling example so that objects have to traverse a more complex internal state space in addition to being loaded/unloaded (leaf state spaces can be made arbitrarily complex, without increasing the number of decoupled states, so long as the additional transitions do not have preconditions on the center). Similarly, Fabre et al.'s [36] framework scales exponentially on leaf factors whose sets of local plans have no compact finite-automata representation.

Regarding hierarchical factored planning, in our Scaling example such approaches will start with high levels containing only the objects (planning "from the leaves to the root"). So the example becomes an exponential separation when

each object may choose between several goal paths. For example, this happens when introducing several trucks which serve different parts of the map.

6.4. Other Methods

Other methods are more remotely related. *Symmetry reduction* (e. g. [17, 18, 19, 68, 21]) is complementary to star-topology decoupling as leaf factors do not need to be symmetric at all. What is more, even if leaf factors *are* symmetric, decoupling may have a stronger effect than symmetry reduction. Our Scaling example is an exponential separation because even perfect symmetry breaking still keeps track of the number of objects at every location.

Star-topology decoupling relates to search with compact state-set representations, in particular with *binary decision diagrams (BDD)* (e. g., [69, 70, 71]), to the extent that a decoupled state s is a compact representation of a state set, namely its hypercube $[s]$. This connection is weak, of course, as hypercubes are very particular state sets, pertaining to particular points in the search, whereas BDDs are used to represent large parts of the search space. Hypercubes always have a small representation by definition; whereas BDDs can represent arbitrary state sets and, consequently, may become large. Exponential separations are easy to construct, e. g., from a BDD-based breadth-first search. An example is a task encoding the adjacency in a $k \times k$ grid [72], which is known to lead to an exponential blow-up in a BDD representation. This task can be formulated with leaves $F^{\{i,j\}}$ encoding whether cells i and j are adjacent, resulting in a decoupled state space with polynomial size.

Decoupled states, specifically their hypercubes, may be reminiscent of *abstractions* (state-space quotients relative to a state partition), extensively used in AI to design lower-bounding heuristic functions (e. g. [73, 74, 75, 76, 77, 78]) Like for BDDs, the connection is weak though. The hypercubes in star-topology decoupling are not a partition (the hypercubes of different decoupled states may overlap). More importantly, they capture the original transition system paths exactly, merely representing and exploring them in a different manner.

Fork/inverted-fork factorings may be somewhat reminiscent of *safe abstraction* for automatic planning-task specification. This method is rooted in early works on hierarchical factored planning with the *downward refinement* property [79, 30], where all abstract plans are realizable at lower levels. Helmert [80] introduced this as an optimization in (an early version of) the Fast Downward system [52], removing inverted-leaf variables with strongly connected state spaces. Haslum [81] generalized this method to consider only the relevant variable values, Tozicka et al. [82] introduced a related reduction technique merging mutually

reachable variable values. All these methods work at the level of single state variables, not factors; none of them can guarantee global optimality; downward refinement is only possible in case of single-directional dependencies, unable to handle more general star topologies. Furthermore, all these techniques rely on mutual reachability between variable values, so that exponential separations are trivial to construct by removing "backwards-actions" not needed in the plan (e. g., in our Scaling example, truck moves towards l_1 and unloading actions at locations other than l_m).

Analyzing task structure has a long tradition in planning. The causal graph, which we use for factorization, has been extensively explored for complexity analysis, in particular the identification of tractable fragments [58, 83, 59, 84, 85, 86, 87, 88, 89, 90]. Recent research has extended this to the consideration of fixed-parameter tractability, with "backdoors" to planning encapsulating the exponential part of the search [91, 92]. While many structural aspects considered in such research relate to our factoring types, star-topology decoupling is directed not at tractable parts of the task, but at worst-case exponential parts that interact in a limited manner. That said, work on the identification of backdoors might inspire new factoring methods in the future.

7. Heuristic Search

Heuristic search has been highly successful in AI Planning since the end of the 90s (e. g. [48, 49, 50, 51, 52, 53, 54, 55]). State space search here is guided by a *heuristic function*, *heuristic* for short, a function h that maps states s to estimates of *remaining cost*, i. e., of the cost of an optimal solution for s. We denote that latter cost as the *perfect heuristic* h^*. If h is a lower bound on h^* – if h is *admissible* – then it can be used for optimal searches like A* [93, 47].

We show in this section that star-topology decoupling combines gracefully with standard heuristic search methods. Section 7.1 discusses heuristic functions, Section 7.2 discusses search algorithms.

7.1. Heuristic Functions

Our definition of heuristic functions for decoupled search follows the standard concepts, instantiated for augmented optimality:

Definition 9 (Heuristic Function). *Let Π be a planning task, and $\mathcal{F}$ a star factoring. A* heuristic function*,* heuristic *for short, is a function $h^{\mathcal{F}} : S^{\mathcal{F}} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$.*

The augmented-perfect *heuristic, $h^{\mathcal{F}*}$, assigns each $s \in S^{\mathcal{F}}$ the augmented cost of an augmented-optimal solution for s. We say that a heuristic h is* augmented-admissible *if $h^{\mathcal{F}} \leq h^{\mathcal{F}*}$.*

The possibility to return ∞, instead of a numeric estimate, is intended to allow $h^{\mathcal{F}}$ to identify unsolvable states, a capability many classical-planning heuristic functions have (e. g. [94, 78, 42]).

Per the definition of augmented optimality (Definition 5, page 24), the augmented-perfect heuristic $h^{\mathcal{F}*}$ accounts, not only for standard path cost from a decoupled state s, but also for the cost of compliant leaf goal paths that a solution for s needs to be augmented with. A subtlety here is that, given the least-commitment strategy for leaf factors, selecting the *entire* leaf path only at the end, part of the associated price lies "in the past", before s. Specifically, let $\pi^{\mathcal{F}}$ be a solution for s, ending in s_G. $h^{\mathcal{F}*}$ accounts for (1) the cost of $\pi^{\mathcal{F}}$, i. e., of the center action sequence π^C underlying $\pi^{\mathcal{F}}$; plus (2) the cost of $\pi^C[s_G]$-compliant leaf goal paths π^{L_i}. In (2), $\pi^C[s_G] = \pi^C[s] \circ \pi^C$, so part of the paths π^{L_i} will be scheduled alongside the path $\pi^C[s]$ leading to s. In particular, while heuristic functions usually return 0 on goal states, that is not so for $h^{\mathcal{F}*}$: we still have to pay the leaf-goal price tag, moving the leaves into place.

Pricing functions, however, capture everything relevant about the past, so that it suffices to consider the possible futures *starting from the current pricing function*. We formulate this in terms of a compilation into classical planning, allowing to estimate $h^{\mathcal{F}*}$ through standard classical-planning heuristics. The idea is to force the plan to "buy" exactly one leaf state from each leaf factor:

Definition 10 ($h^{\mathcal{F}*}$ as Classical Planning). *Let $\Pi = (V, A, c, I, G)$ be a planning task, $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$, and s a decoupled state. The* ***buy-leaves compilation*** *is the planning task $\Pi_{L\$} = (V_{L\$}, A_{L\$}, c_{L\$}, s_{L\$}, G_{L\$})$, obtained from Π and s as follows:*

1. *The variables $V_{L\$}$ add a new Boolean variable $\mathsf{bought}[F^L]$ for every leaf F^L, $V_{L\$} := V \cup \{\mathsf{bought}[F^L] \mid F^L \in \mathcal{F}^L\}$. For all variables $v \notin F^C$, we add the new value none into $\mathcal{D}(v)$.*

2. *The initial state is $s_{L\$} := \mathsf{center}[s] \cup \{v = \mathsf{none} \mid v \notin F^C\} \cup \{\mathsf{bought}[F^L] = 0 \mid F^L \in \mathcal{F}^L\}$.*

3. *The goal is $G_{L\$} := G \cup \{\mathsf{bought}[F^L] = 1 \mid F^L \in \mathcal{F}^L\}$.*

4. *The actions $A_{L\$}$ are the previous ones A, adding precondition* $\mathsf{bought}[F^L] = 1$ *to a whenever $\mathcal{V}(\mathsf{pre}(a)) \cap F^L \neq \emptyset$. We furthermore add, for every leaf F^L, and for every leaf state $s^L \in S^L[F^L]$ where* $\mathsf{prices}[s](s^L) < \infty$*, a new action $a[s^L]$ with precondition* $\mathsf{pre}(a[s^L]) := \{\mathsf{bought}[F^L] = 0\}$ *and effect* $\mathsf{eff}(a[s^L]) := s^L \cup \{\mathsf{bought}[F^L] = 1\}$.

5. *The cost function $c_{L\$}$ extends the previous one c by setting* $c_{L\$}(a[s^L]) := \mathsf{prices}[s](s^L)$ *for each new action $a[s^L]$.*

We will use the buy-leaves compilation to design heuristic functions for decoupled search.[5] We explain this below; let us first explain the compilation. The leaf factors are assigned the value none initially to indicate that they "do not have a state yet". Before we can do anything relying on a leaf factor F^L, we have to buy (exactly) one of its states, at the price specified in the decoupled state s at hand. The price we pay in doing so accounts for F^L's compliant path before s; the classical plan obtained afterwards accounts for F^L's compliant path behind s.

Note that the goal in $\Pi_{L\$}$ forces the plan to buy a leaf state from *every* F^L, even if F^L has no goal and would otherwise not be touched by any actions in the plan for $\Pi_{L\$}$. This is necessary because F^L may have had to move *before* s: we need to account for any costs incurred in F^L in order to enable (to comply with) the decoupled path $\pi^C[s]$ leading to s in the first place.

Example 9. *Consider the Vanilla example, with object o and two trucks moving along the line l_1, l_2, l_3. Initially, $t_A = l_1$, $t_B = l_3$, and $o = l_1$. The goal is $o = l_3$. Consider the strict-star factoring $F^C = \{t_A, t_B\}$ and $\mathcal{F}^L = \{\{o\}\}$, and consider the decoupled state s reached by applying move(t_A, l_1, l_2). We have $h^{\mathcal{F}*}(s) = 3$ due to the optimal solution induced by the center path $\langle$move$(t_A, l_2, l_3)\rangle$, augmented with the leaf goal path $\langle$load(t_A, l_1), unload$(t_A, l_3)\rangle$. Observe that load(t_A, l_1) is scheduled before s, while unload(t_A, l_3) is scheduled behind s.*

The finite prices in s are 0 for $\{o = l_1\}$, 1 for $\{o = t_A\}$, and 2 for $\{o = l_2\}$. In the buy-leaves compilation $\Pi_{L\$}$, we can pay one of these prices to obtain a leaf state other than none*. The cheapest plan for $\Pi_{L\$}$ results from buying $\{o = t_A\}$, yielding the plan $\langle a[\{o = t_A\}]$, move(t_A, l_2, l_3), unload$(t_A, l_3)\rangle$ which corresponds to the part behind s in the optimal solution above.*

[5] One could, in principle, use the compilation to find solutions for decoupled states. But that would be nonsensical, replacing decoupled search with standard search, defeating the purpose of our approach.

Consider now the NoEmpty example, which is the same except that truck moves require o to be in the truck. Say, however, that the goal is $t_A = l_3$. Consider again the decoupled state s reached by applying move(t_A, l_1, l_2). The finite prices in s now are 1 for $\{o = t_A\}$ and 2 for $\{o = l_2\}$; $\{o = l_1\}$ is no longer reachable as the truck move committed o to $\{o = t_A\}$. Observe that $h^{\mathcal{F}*}(s) = 2$ because the optimal solution is $\langle$move$(t_A, l_2, l_3)\rangle$ augmented with the compliant leaf goal path $\langle$load$(t_A, l_1)\rangle$. Now, if $\Pi_{L\$}$ did not *have the goal* $\mathsf{bought}[\{o\}] = 1$*, then $\langle$move$(t_A, l_2, l_3)\rangle$ would be a plan for $\Pi_{L\$}$, of cost $1 < h^{\mathcal{F}*}(s)$. The goal* $\mathsf{bought}[\{o\}] = 1$ *forces the plan to pay for any services o may have needed to provide before s.*

Given a classical-planning heuristic function h, we obtain the **buy-leaves heuristic function** $h^{\mathcal{F}}_{L\$}$ by setting, given a decoupled state s, the value of $h^{\mathcal{F}}_{L\$}$ on s to the value of h on $s_{L\$}$ in $\Pi_{L\$}$. In other words, $h^{\mathcal{F}}_{L\$}$ results from h's estimate of initial-state remaining cost in the buy-leaves compilation for s. This construction guarantees two very desirable properties: *if h is admissible, then so is $h^{\mathcal{F}}_{L\$}$*; and *if h is perfect, then so is $h^{\mathcal{F}}_{L\$}$*. The former is of course crucial for optimal planning. The latter curbs information loss: if the heuristic information given by h is perfect, then no loss is incurred. Both properties follow directly from the fact that $\Pi_{L\$}$ indeed captures $h^{\mathcal{F}*}$:

Lemma 5. *Let Π be a planning task, $\mathcal{F}$ a star factoring, and s a decoupled state. Let $\Pi_{L\$} = (V_{L\$}, A_{L\$}, c_{L\$}, s_{L\$}, G_{L\$})$ be the buy-leaves compilation. Then $h^*(s_{L\$}) = h^{\mathcal{F}*}(s)$.*

Proof: "$\leq$": Say $\pi^{\mathcal{F}}$ is any solution for s, ending in decoupled state s_G. Let π^C be the sequence of center actions underlying $\pi^{\mathcal{F}}$. We can construct a plan π for $\Pi_{L\$}$ by augmenting π^C with a cheapest π^C-compliant sequence of leaf actions for each leaf factor F^L, starting from a finite-price leaf state s^L in $s \in S^L[F^L]$; and buying that leaf state via the action $a[s^L]$. By construction, the cost of π in $\Pi_{L\$}$ is $\mathsf{cost}(\pi^C) + \mathsf{LGPrice}(s_G) = \mathsf{AugCost}(\pi^{\mathcal{F}})$. Hence $h^*(s_{L\$}) \leq h^{\mathcal{F}*}(s)$.

"$\geq$": Say π is any plan for $\Pi_{L\$}$. Let π^C be the subsequence of center actions. As the construction of $\Pi_{L\$}$ forces the plan to buy, for each leaf factor, exactly one leaf state $s^L \in S^L[F^L]$, π^C must be augmentable with π^C-compliant leaf paths achieving the leaf goals starting from these s^L. So π^C induces a solution for s, ending in some s_G. If the leaf paths used by π are the cheapest ones, then $\mathsf{cost}(\pi) = \mathsf{cost}(\pi^C) + \mathsf{LGPrice}(s_G) = \mathsf{AugCost}(\pi^{\mathcal{F}})$. For arbitrary leaf paths, we have that $\mathsf{cost}(\pi) \geq \mathsf{AugCost}(\pi^{\mathcal{F}})$, and hence that $h^*(s_{L\$}) \geq h^{\mathcal{F}*}(s)$ as desired. □

Theorem 8. *Let Π a planning task, and $\mathcal{F}$ a star factoring. Let h be a classical-planning heuristic function. If h is admissible, then $h^{\mathcal{F}}_{L\$}$ is augmented-admissible. If $h = h^*$, then $h^{\mathcal{F}}_{L\$} = h^{\mathcal{F}*}$.*

Proof: Direct from Lemma 5. □

It should be noted that, while our construction can in principle be used with any heuristic function h, doing so efficiently may be challenging. In particular, the subset of artificial actions $a[s^L]$ present, as well as their cost, depend on the decoupled state s. This is problematic for heuristic functions relying crucially on precomputations prior to search, like the aforementioned abstraction heuristics. We have so far realized the buy-leaves compilation for a number of canonical heuristic functions not relying on precomputation. (Namely for $h^{\max}$, $h^{\text{LM-cut}}$, and h^{FF}; see more details in Section 8.)

7.2. Heuristic Search Algorithms

Disregarding solution quality, one can run any search algorithm on the decoupled state space $\Theta^{\mathcal{F}}_{\Pi}$, treating it like an arbitrary transition system. When taking solution quality into account, in particular for optimality, matters are a little more subtle as we need to tackle augmented-optimality in $\Theta^{\mathcal{F}}_{\Pi}$, in difference to the standard additive-cost notion for which heuristic search algorithms are defined. In particular, as we illustrated in Example 4, a solution for $\Theta^{\mathcal{F}}_{\Pi}$ may have a worse solution as a prefix. The issue is easy to resolve though, by making the leaf-goal price tags explicit, encoding them as additional transitions to a fresh goal state:

Definition 11 (Explicit Leaf-Goal Price). *Let $\Pi = (V, A, c, I, G)$ be a planning task, $\mathcal{F}$ a star factoring, and $\Theta^{\mathcal{F}}_{\Pi} = (S^{\mathcal{F}}, A^C, c|_{A^C}, T^{\mathcal{F}}, I^{\mathcal{F}}, S^{\mathcal{F}}_G)$ the decoupled state space. The **explicit leaf-goal price** state space $\Theta^{LG\mathcal{F}}_{\Pi}$ is like $\Theta^{\mathcal{F}}_{\Pi}$, but with a new state s^* set to be the only goal state; and adding, for every $s_G \in S^{\mathcal{F}}_G$, a new transition $s_G \to s^*$ with a new label whose cost is $\mathsf{LGPrice}(s_G)$.*

In other words, $\Theta^{LG\mathcal{F}}_{\Pi}$ requires to pay the leaf-goal price tag at the end of any solution, in the form of a transition having that cost. Obviously, the solutions for $\Theta^{LG\mathcal{F}}_{\Pi}$ are in one-to-one correspondence with those for $\Theta^{\mathcal{F}}_{\Pi}$, where additive cost in the former corresponds to augmented cost in the latter.

Consider now any optimal search algorithm X for additive cost in labeled transition systems, and consider a heuristic function $h^{\mathcal{F}}$ defined on $S^{\mathcal{F}}$. Define a heuristic function $h^{\mathsf{LG}\mathcal{F}}$ for $\Theta^{LG\mathcal{F}}_{\Pi}$ simply by extending $h^{\mathcal{F}}$ with $h^{\mathsf{LG}\mathcal{F}}(s^*) := 0$. Define the algorithm **Decoupled-X (D-X)** as running X with $h^{\mathsf{LG}\mathcal{F}}$ on $\Theta^{LG\mathcal{F}}_{\Pi}$, returning $\pi^{\mathcal{F}}$ when X returns $\pi^{\mathcal{F}} \circ \langle s_G \to s^* \rangle$. With the above, we have:

Proposition 4. *If X is complete, then D-X is complete. If X is optimal for admissible heuristic functions, then D-X is augmented-optimal for augmented-admissible heuristic functions.*

With Theorem 8 and Proposition 4 together, we can (in principle) take any complete/optimal heuristic search algorithm X, and any classical-planning heuristic function h, and turn them into a complete/augmented-optimal search algorithm for $\Theta^{\mathcal{F}}_{\Pi}$, searching with $h^{\mathsf{LG}\mathcal{F}}_{L\$}$ on $\Theta^{LG\mathcal{F}}_{\Pi}$. By Theorems 3 and 4, this yields a complete/optimal planning algorithm.

We remark that, while this construction is simple and canonical, there also are heuristic search algorithms specialized to star-topology decoupling. One can define heuristic functions estimating only the remaining cost for the center. This enables search algorithms exploring decoupled states by estimated center cost, pruning against the best known global solution so far. We described and tested such an algorithm in our prior work [1]. We omit this here for brevity, as the empirical benefits over the simpler approach are limited on standard benchmarks.

8. Experiments

We consider all three algorithmic planning problems, i. e., optimal planning, satisficing planning, and proving unsolvability. For each of these, we use star-topology decoupling together with canonical baseline heuristic search planning algorithms, comparing its performance to the state of the art.

We begin in Section 8.1 by briefly describing important aspects of our implementation, and explaining the experiments setup. Section 8.2 discusses state space size reductions, filling in some data not covered by our preview in Figure 1. The three algorithmic planning problems are covered, in turn, by Sections 8.3, 8.4, and 8.5.

8.1. Implementation & Experiments Setup

Our implementation of star-topology decoupling is in C++, within the Fast Downward (FD) framework [52]. FD is the most commonly used framework in classical planning, and it in particular contains the state-of-the-art heuristic search planning algorithms. Our implementation is modular in that it affects only FD's representation of states, and the computation of state transitions. All of FD's search algorithms can be run via Proposition 4. All of FD's heuristic functions can be run (in principle) via Theorem 8. In our experiments, we instantiate these connections with canonical baseline algorithms. These algorithms are orthogonal

to ours, and their details are not needed to understand our results. Hence we only give a brief overview. The search algorithms we use are:

1. A*, which is used in all state-of-the-art heuristic search optimal-planning systems.

2. **Greedy best-first search (GBFS)**, specifically FD's lazy greedy best-first search with a dual open queue for *preferred operators*. This algorithm is canonical for satisficing heuristic search planning, in that its performance is close to the state of the art, and it is commonly used as a baseline. Preferred operators are applicable actions used in the abstract plan internally generated by a heuristic function. One of the two open queues uses only these actions; the other open queue retains the states generated by other actions, preserving completeness.

The heuristic functions we use are:

1. $\mathbf{h}^{\text{max}}$ [49], a basic admissible heuristic function mainly useful for detecting unsolvable states during search. h^{max} estimates remaining cost by assuming that, from each subgoal conjunction, it suffices to achieve the single most costly variable value.

2. $\mathbf{h}^{\text{LM-cut}}$ [53], a canonical admissible heuristic function, among the state of the art in optimal planning and widely used as a baseline in that context. $h^{\text{LM-cut}}$ estimates remaining cost through identifying *landmarks* [95], action sets that must be hit by every plan.

3. $\mathbf{h}^{\text{FF}}$ [50], a canonical inadmissible heuristic function, widely used in satisficing-planning systems. h^{FF} estimates remaining cost through the cost of an abstract plan pretending that state variables accumulate, rather than switch between, their values (this is commonly known as the *delete relaxation* [48, 49]).

4. A **blind** heuristic, which returns 0 on goal states and the cost of the cheapest action otherwise. Using this in A* is the most common design of a cost-optimal blind search in FD.

The key aspect of decoupled search implementation is the representation, and computation, of decoupled states. As earlier hinted, to avoid the repetition of leaf states across decoupled states, we precompute the state space of each leaf factor just once before search begins, giving IDs to its leaf states and storing these in an array. In the decoupled states, we represent reachability functions as sets of IDs

(the reached leaf states), and we represent pricing functions as sets of (ID,price) pairs (the prices of the reached leaf states). Instead of compliant path graphs, we merely store for each reached leaf state s^L the last action in a cheapest compliant path to s^L. This information is maintained incrementally across decoupled-state transitions $s \to t$.

To automatically identify a star factoring for an input task Π, we use one of three simple strategies:

1. **Fork (F):** Compute the interaction graph $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ of the factoring into causal graph SCCs. Let $\mathcal{I}$ be the set of leaf vertices F in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ that, as a simple sanity criterion, satisfy the size bound $\prod_{v \in F} |\mathcal{D}(v)| < 2^{32}$. Abstain if $|\mathcal{I}| < 2$. Otherwise, take $\mathcal{I}$ as the leaf factors, $\mathcal{F}^L := \mathcal{I}$, and collect all remaining variables into the center $F^C := V \setminus \bigcup_{F \in \mathcal{I}} F$.

2. **Inverted-Fork (IF):** As above, but defining $\mathcal{I}$ to contain the root vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$.

3. **X-Shape (X):** As above, but defining $\mathcal{I}$ to contain the leaf vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$, plus those root vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ that do not have an arc to a leaf vertex.

As the benefit of decoupled search lies in avoiding the multiplication of states across leaves, we abstain from single-leaf factorings. As decoupled states explicitly enumerate leaf-factor state spaces, these should not be excessively large, so we apply the simple sanity criterion stated. By Theorem 1, unless the sanity criterion applies, the F (IF) strategy yields a fork (inverted-fork) factoring with maximal possible number of leaves. The X strategy is just a simple way of combining the two. As $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ typically is very small, all three strategies take negligible runtime (rounded to 0.00 seconds in most cases).

Figure 7 shows statistics regarding the number of instances factorized using each strategy, the number of leaf factors, and the state space size of those leaf factors (the set of domains, slightly larger than that in Figure 1, will be explained below). As the data shows, leaf factor state spaces are typically small, with notable exceptions in Mystery, Pathways, and TPP. The X strategy subsumes the F and IF ones in terms of the number of factorizable instances (this is not necessarily so as there could be cases with a single $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ leaf SCC and several root SCCs all of which have an arc to the leaf). Moreover, in almost all cases, X yields exactly the outcome of either F or IF, behaving like a switch choosing whichever of the two structures is present (the only exceptions to this are Pathways and Woodworking). Given this, search performance with X almost always subsumes that of both F and

		Fork (F)			Inverted-Fork (IF)			X-Shape (X)				
	# Inst	# Inst	Leaf Factors		# Inst	Leaf Factors		# Instances			Leaf Factors	
Domain	All	F	#	MSize	IF	#	MSize	X	F	IF	#	MSize
Solvable Benchmarks: From the International Planning Competition (IPC)												
Childsnack	40				40	3	4	40		40	3	4
Depots	22				22	2.8	5.2	22		22	2.8	5.2
Driverlog	20	20	8.3	12.2				20	20		8.3	12.2
Elevators	100				100	4.5	7.4	100		100	4.5	7.4
Floortile	80				80	2.3	2	80		80	2.3	2
Logistics	63	63	11.2	71.2	63	23.3	9.8	63	63		11.2	71.2
Miconic	150	145	16	4				145	145		16	4
Mystery	19				1	3	189660	1		1	3	189660
NoMystery	40	40	9	10.5				40	40		9	10.5
Pathways	30	29	21.1	2				30	25		20.6	7670.6
PSR	50	3	2	2				3	3		2	2
Rovers	40	40	20.9	2	38	7.7	35.2	40	40		20.9	2
Satellite	36	36	54.8	2	34	7.4	69.3	36	36		54.8	2
TPP	30	27	8.3	954.4	26	4.8	6.9	29	27	2	8.3	889.4
Transport	140				140	3.1	48.6	140		140	3.1	48.6
Woodworking	100	61	3.9	43.6	87	3.9	44.7	87	8		4	60.2
Zenotravel	20	20	9.9	11.7	18	3.5	63.4	20	20		9.9	11.7
Unsolvable Benchmarks: From the Unsolvability IPC'16												
BagTransport	29				29	2.9	55.9	29		29	2.9	55.9
NoMystery	24	24	14.1	15.1				24	24		14.1	15.1
Rovers	20	20	13.0	2.0				20	20		13.0	2.0
Unsolvable Benchmarks: IPC Mystery, Others Extended from [42]												
Mystery	11				3	2.7	43673.7	3		3	2.7	43673.7
NoMystery	40	40	15	13				40	40		15	13
Rovers	40	40	26.8	2				40	40		26.8	2
$\sum$	1144	608			681			1052	551	417		

Figure 7: Factoring strategy statistics. "F"/"IF"/"X": fork/inverted-fork/X-shape factoring identified. In the X-Shape part of the table, "F" and "IF" show the number of instances whose X factoring actually is a fork/inverted-fork factoring. Per-domain averages shown for the leaf factor statistics. "MSize" is maximum leaf factor size, i. e., the number of reachable leaf states when ignoring preconditions on the center.

IF, and in the below we consider X only. We will include data for the rare cases where either F or IF results in substantially better performance.

We also experimented with more complex factoring strategies, in particular, attempting to greedily maximize not only the number of leaves, but also *leaf mobility*, the number of actions affecting only a leaf (possibly with a prevail condition on the center), which the search need not branch over. The results were not convincing. Another reason to prefer simple factoring strategies is to avoid overfitting to the benchmarks, by fine-tuning factoring outcomes, and in particular by fine-tuning the set of benchmarks abstained upon: If we can already see (after less than 0.01 seconds) that decoupled search does not make sense on an input task, then the natural thing to do is to abstain. But this limits the view of the benchmark

set, and fine-tuning factoring strategies runs the risk of fine-tuning that view. We hence stick to the simplest possible form of that view, to evaluate the basic properties of star-topology decoupling, rather than the possibilities for system tuning. Nevertheless, throughout the following results, the reader should keep in mind that they are taken *over the subset of benchmarks with an obvious star topology*. The results must be qualified against that selection.[6]

The collection of benchmarks we start from is that of the IPC, more specifically, of the classical-planning tracks of the IPC, and of these tracks only those benchmarks that fit our planning task syntax. The major part of the IPC – 100 *test suites* in total – does fit that profile, so this is not limiting. Of the 100 test suites, the X strategy selects (does not abstain on at least one instance from) 34. The IPC benchmarks are organized in terms of *domains*, sets of related planning tasks. Each domain may be associated with several test suites. Our IPC benchmark collection contains 49 domains in total, and 17 of those are selected by the X strategy. As there is not much interest in distinguishing the different test suites per domain (they differ mainly in instance size), we accumulate these, presenting results per-domain rather than per-test-suite, as already done in Figures 1 and 7 above. The IPC benchmarks are publicly available from the IPC web pages.[7].

All IPC benchmarks, save 11 instances of the Mystery domain, are solvable. For proving unsolvability, we use the benchmarks used in the unsolvability IPC'16, as well as the well-established benchmark collection by Hoffmann et al. [42]. The former benchmark collection contains 15 domains, of which our strategy can detect an X factoring in 3. The latter contains 8 domains (one test suite each), of which the X strategy selects 3. In 2 of these 3 domains, NoMystery and Rovers, both of which are concerned with resource-constrained planning, we introduce additional easier instances in order to get some performance data for all algorithms tested. Specifically, we extend the range of the *constrainedness levels* C – the ratio between available and minimum-needed amount of fuel/energy [96] – from $\{0.5, 0.6, \ldots, 0.9\}$ in the original benchmarks, to $\{0.2, 0.3, \ldots, 0.9\}$. Instances with smaller constrainedness level are easier to prove unsolvable as they are more tightly constrained.

All experiments were run on a cluster of machines with Intel Xeon E5-2660

[6]We could, in principle, invoke an arbitrary algorithm outside that selection. But that would only water down our results. It would also be ill-defined. There is no reason, in practice, to choose any algorithm over another, unless one investigates the design of prediction-based dynamic portfolios.

[7]http://www.icaps-conference.org/index.php/Main/Competitions

processors running at 2.2 GHz. As is customary in the IPC, the runtime/memory limit was set to 30 minutes/4 GB.

The state of the art algorithms differ considerably depending on the purpose, i. e., optimal planning vs. satisficing planning vs. proving unsolvability. We will therefore introduce these algorithms, which we compare to empirically, in the subsections below. A few words are in order up front though, regarding factored planning, and regarding Petri net unfolding.

Localized factored planning [32, 33, 35, 36, 37, 39, 40] has inspired star-topology decoupling, so is a natural approach to compare to empirically. Yet, all but one of these approaches (that by Fabre et al. [36]) cannot guarantee global optimality; many works are purely theoretical; and good results on standard IPC benchmarks are scarce. Some works run proof-of-concept experiments on a few custom-designed (non-IPC) toy benchmarks only. Some approaches (e. g., that by Fabre et al.) are explicitly reported by their authors to not work well on IPC benchmarks. The empirically most successful approach is the aforementioned partition-based pruning [40], which exploits permutability of intra-factor plans. Hence we compare to that approach here.

Petri net unfolding methods are not defined on planning tasks, so given a task Π we need to translate it into a Petri net first. Such translations are easy and known [24]. To maximize comparability with our FD-based tools, we implemented translations starting from FD's internal planning task representation (and hence relying on the same relevance and invariance analysis preprocesses as used by FD [97]). Each variable value $v = d$ becomes a place $p_{v=d}$ in the Petri net, and each action a becomes a transition t_a in the Petri net. If $v = d$ is a precondition of a, then $p_{v=d}$ becomes an input of t_a, and if $v = d$ is an effect of a, it becomes an output of t_a. In case $v = d$ is not consumed, i. e., is a prevail condition, we need to encode a read arc from $p_{v=d}$ to t_a. Contextual Petri nets support this natively. For translation into a standard Petri net, we use the aforementioned plain method, making $p_{v=d}$ both, an input and an output, of t_a. We compare to standard respectively contextual unfolding tools, Punf [43] respectively Cunf [44], regarding state space/unfolding size. Regarding planning performance, we compare to Bonet et al.'s [28] heuristic-search-planning Petri net unfolding tool, which realizes $h^{\max}$ and h^{FF} (but not $h^{\text{LM-cut}}$, nor preferred operators) in this context.

8.2. State Space Size

Figure 8 shows data supplementary to the preview given in Figure 1. We show also those 3 domains where no state space was successfully built by any method;

and we include the Petri net unfolding methods into the representation size comparison. The representation size for Punf is the number of events generated, that for Cunf is the number of event histories generated.

	# Instances		Reachable State Space. Right: Average over Instances Commonly Built											
			Success						Representation Size (in Thousands)					
Domain	All	X	Std	POR	Punf	Cunf	OPT	COM	Std	POR	Punf	Cunf	OPT	COM
Solvable Benchmarks: From the International Planning Competition (IPC)														
Childsnack	40	40	0	0	0	0	0	0						
Depots	22	22	4	4	2	2	3	**5**	407.2	407.2	59.5	58.6	471.1	**54.3**
Driverlog	20	20	5	5	3	3	8	**10**	446.7	446.7	112.3	95.5	74.7	**17.7**
Elevators	100	100	21	17	1	3	8	**41**	1,941.8	1,941.5	544.3	543.3	2,098.6	**36.3**
Floortile	80	80	**2**	**2**	0	0	0	**2**						
Logistics	63	63	12	12	7	11	23	**27**	1,121.2	1,121.2	111.2	118.4	133.2	**12.1**
Miconic	150	145	50	45	25	30	45	**145**	154.6	152.3	139.1	143.1	5.9	**.7**
Mystery	19	1	0	0	0	0	0	0						
NoMystery	40	40	11	11	5	7	**40**	**40**	266.2	248.8	124.2	101.3	4.3	**3.9**
Pathways	30	30	**4**	**4**	3	3	**4**	**4**	4,748.8	115.9	**53.8**	103.5	1,787.3	1,787.2
PSR	50	3	3	3	3	3	3	3	39.4	33.9	10.2	**7.9**	11.1	11.1
Rovers	40	40	5	**6**	4	4	5	5	113.6	76.5	8.9	**6.8**	40.3	39.3
Satellite	36	36	**5**	**5**	**5**	**5**	4	4	2,864.2	582.5	53.7	**45.1**	2,219.1	352.7
TPP	30	29	5	5	4	4	**11**	**11**	192.5	192.5	11.6	12.4	**.2**	**.2**
Transport	140	140	28	23	11	11	18	**34**	151.7	151.7	90.0	90.0	325.3	**9.2**
Woodworking	100	87	11	20	16	**22**	15	16	109,174.4	199.9	**.6**	1.2	11,895.8	4,274.2
Zenotravel	20	20	**7**	**7**	2	4	**7**	**7**	1.3	1.3	2.2	2.3	**.3**	**.3**
Unsolvable Benchmarks: From the Unsolvability IPC'16														
BagTransport	29	29	7	6	3	3	4	**11**	1,889.8	1,889.8	219.6	219.6	2,025.8	**34.9**
NoMystery	24	24	2	2	0	1	17	**18**						
Rovers	20	20	7	7	0	3	7	**8**						
Unsolvable Benchmarks: IPC Mystery, Others Extended from [42]														
Mystery	11	3	0	0	0	0	0	0						
NoMystery	40	40	9	8	2	4	**40**	**40**	91.7	38.8	23.9	5.5	**.5**	**.5**
Rovers	40	40	**4**	**4**	0	0	**4**	**4**						
$\sum$	1144	1052	202	196	96	123	267	**435**						

Figure 8: State space size using the X-shape factoring strategy. Best results highlighted in **boldface**. "POR" is partial-order reduction using strong stubborn sets [16]. Punf [43] is Petri net unfolding, Cunf [44] is contextual Petri net unfolding. "OPT" and "COM" are the optimality-preserving (pricing functions) respectively completeness-preserving (reachability functions) variants of the decoupled state space.

The partial-order reduction technique we use is strong stubborn sets pruning as per Wehrle and Helmert [16]. We remark that this approach (in difference to the other approaches compared here) depends on the goal condition. Naturally, we use the original goal conditions supplied with the instances. On goal states (in solvable instances), we allow POR to prune all actions.

Petri net unfolding sometimes is superior at building small state-space representations. This pertains, foremost, to the Woodworking benchmark where Cunf excels. But it also pertains to some other domains, where the representation size advantage does not translate into an advantage in the number of state spaces built.

This is due to the runtime overhead of building the unfolding: in the respective smallest unsuccessful benchmark instances, the unfolding methods run out of time.

Observe that, to a large extent, the data for factorings in Figure 7 explains that for decoupled search in Figure 8. The state-space reduction tends to be large if (a) there is a sufficient number of leaf factors, and each leaf factor is neither (b) too small (not allowing a substantial reduction), nor (c) too large (incurring substantial overhead). Prime examples satisfying all of (a)–(c) are Driverlog, Elevators, Logistics, Miconic, NoMystery, TPP, Transport, Woodworking, and Zenotravel – a very good match with those domains where star-topology decoupling works well. Examples not satisfying (a) are Floortile and PSR; examples not satisfying (b) are these same two domains, plus Rovers and Satellite; examples not satisfying (c) are Mystery and Pathways. Note that, given this strong connection between simple factoring-structure features and state-space reduction performance, one could easily design factoring strategies filtering out the bad cases in Figure 8. We do not do that here for the sake of clarity, cf. the discussion of fine-tuning above.

Even with the bad cases, Figure 8 certainly shows that star-topology decoupling can yield huge advantages compared to related methods. We examine next to what extent these advantages are preserved when supplying the search with the orthogonal search techniques – heuristic functions, preferred operators pruning, etc. – intensively developed in the AI planning community, geared at optimal planning, at satisficing planning, respectively at proving unsolvability.

8.3. Optimal Planning

Optimal planning is a separate track in the international planning competitions. In this subsection, we use the test suites of the IPC optimal tracks only. We consider additive-cost optimality, used in the IPC since 2008.[8] For this form of planning, the most competitive algorithms are based on heuristic state space search, on BDDs, and on optimality-preserving state-space reduction techniques. We hence compare against the leading representatives of these techniques in AI planning: Symba [101], the winner of the IPC'14 optimal track, which runs a BDD-based heuristic search; A* with strong stubborn sets pruning as used previously already [16]; A* with symmetry reduction [21]; and A* with a combination of symmetry reduction and strong stubborn sets [102]. We furthermore compare

[8] Previous IPCs allowed also other forms of optimality, e. g. the makespan of parallel plans (e. g., [98, 99, 100]).

to partition-based pruning and heuristic-search Petri net unfolding, as discussed above. Figure 9 shows the data.

			(A) Coverage (Number of Tasks Solved)																		(B) Nodes		(C) Runtime					
	# Inst			Blind							$h^{\max}$	F Blind			$h^{\text{LM-cut}}$						$h^{\text{LM-cut}}$		$h^{\text{LM-cut}}$					
Dom	All	X	BD	A*	P	Sy	Psy	PP	HP	D-	HP	F	A*	D-	A*	P	Sy	Psy	PP	D-	Psy	D-	BD	P	Sy	Psy	PP	D-
Csna	20	20	3	0	±0	**+6**	**+6**	±0	±0	±0	±0				0	±0	**+6**	**+6**	±0	±0								
Depo	22	22	5	4	±0	+2	+1	±0	-2	±0	-2				7	±0	**+1**	**+1**	±0	±0	**2.3**	1.2	1.1	0.9	2.0	**1.9**	1	0.9
Drive	20	20	**14**	7	±0	±0	±0	±0	-3	+4	-3	20	7	+4	13	±0	±0	±0	±0	±0	1.7	**2.2**	0.6	0.9	1.2	**1.4**	1.2	0.6
Eleva	50	50	**44**	26	±0	+2	+4	+5	-10	-10	-10				40	±0	±0	±0	+1	+1	**2.2**	1.8	**6.9**	1.1	1.2	2.0	3	1.6
Floor	40	40	**34**	2	±0	±0	±0	±0	-2	±0	-2				13	±0	+3	+3	±0	±0	**2.0**	1.0	**148**	0.9	2.0	1.9	1.1	1.1
Logi	63	63	25	12	±0	+2	+3	+2	-2	+14	±0	63	12	+14	26	±0	±0	+1	±0	**+8**	7.8	**211**	0.4	3.1	1.5	8.2	2.8	**293**
Mico	150	145	102	50	-5	+1	-4	-5	-35	-5	-32	145	50	-5	**136**	**±0**	**±0**	**±0**	**±0**	-1	2.1	**2.2**	0.0	0.9	**1.9**	1.8	0.8	0.4
Myst	19	1	0	0	±0	±0	±0	±0	±0	±0	±0				0	±0	±0	+1	±0	±0								
NoM	20	20	15	8	±0	+1	-1	±0	-6	+12	-1	20	8	+12	14	±0	±0	±0	±0	**+6**	1.0	**29.4**	0.6	0.8	1.1	0.4	0.9	**30.2**
Path	30	30	**5**	4	±0	±0	±0	±0	-1	±0	±0	29	3	±0	**5**	**±0**	**±0**	**±0**	**±0**	-1	1.7	**4.6**	0.1	**1.2**	1.0	**1.2**	0.6	0.8
PSR	50	3	3	3	±0	±0	±0	±0	±0	±0	±0	3	3	±0	3	±0	±0	±0	±0	±0	1.6	**1.7**	0.1	**1.0**	**1.0**	**1.0**	0.6	**1.0**
Rove	40	40	**13**	6	+1	±0	+1	±0	-5	±0	±0	40	6	±0	7	+2	±0	+3	+3	+1	3.3	**4.8**	**11.8**	3.3	0.9	2.8	2.5	2.5
Sate	36	36	8	6	±0	±0	+1	+1	-2	±0	-1	36	6	±0	7	+4	+6	**+7**	+5	±0	**96**	1.0	32.3	23.9	4.1	**99**	17.1	0.7
TPP	30	29	7	5	±0	+1	+1	±0	-2	**+18**	-3	27	5	**+18**	5	±0	+1	+1	±0	+13	6.9	**2557**	11.7	0.9	6.5	6.2	1.1	**346**
Trans	70	70	**29**	24	-1	±0	-1	±0	-8	-5	-6				23	±0	±0	±0	±0	±0	**1.4**	0.8	**6.6**	0.9	1.1	1.1	1.4	0.7
Wood	50	39	27	8	+7	±0	+9	±0	+7	+2	+4	18	4	+2	20	+17	±0	+17	+2	+8	**1468**	11.2	5.1	517	1.5	**765**	1	13.0
Zeno	20	20	10	8	±0	±0	±0	±0	-2	+3	-1	20	8	+3	**13**	**±0**	**±0**	**±0**	**±0**	**±0**	2.0	**4.6**	1.9	0.9	1.5	1.6	2.1	**5.4**
$\sum$	730	648	344	173	+2	+15	+20	+3	-73	+33	-57	421	112	+48	332	+23	+17	**+40**	+11	+35								

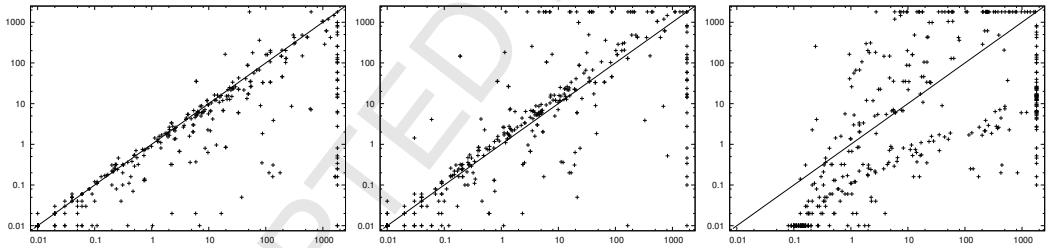
Figure 9: Optimal planning performance. Best results highlighted in **boldface**. Part (A) shows coverage, relative to the baseline A* except for "BD". Parts (B) and (C) show *improvement factors* relative to the baseline, i. e., the ratio baseline/planner of (B) the per-domain sum of evaluated search nodes (calls to $h^{\text{LM-cut}}$), and (C) the per-domain sum of runtime. The underlying instances in (B) and (C) are those commonly solved by the baseline and the planners shown, skipping in (C) trivial instances commonly solved in ≤ 0.1 seconds. "F Blind" shows data for fork factoring. "BD" is Symba [101], the BDD-based winner of the IPC'14 optimal track. "HP" is optimal planning with Bonet et al.'s [28] heuristic-search Petri net unfolding tool (coverage for $h^{\max}$ shown relative to Blind A*). "P" is partial-order reduction using strong stubborn sets [16], "Sy" is A* with symmetry reduction [21], "Psy" is A* with both [102]. "PP" is A* with partition-based pruning [40]. "D-" is D-A*.

Consider first the coverage data, i. e., the number of benchmark instances solved, in part (A) of the table. For blind state space search, star-topology decoupling is the most effective technique, somewhat more so when using fork factorings only, due to a performance loss in Elevators when using X-shape factorings. For both, blind search and search with $h^{\text{LM-cut}}$, the combination of strong stubborn sets with symmetry reduction, *Psy* in the figure, outperforms each of its component techniques, and outperforms partition-based pruning. It is complementary to star-topology decoupling in that it works well on different domains. The strongest domains of Psy are Childsnack, Floortile, Rovers, Satellite, and Woodworking, whereas those of star-topology decoupling are Driverlog, Logistics, No-Mystery, TPP, Woodworking, and Zenotravel. In cross-domain overall coverage,

both methods lead to substantial gains (differing somewhat basically due to benchmark scaling). Petri net unfolding generally is not competitive; BDD-based search is, yielding the strongest performance in several domains largely complementary to those listed above.

Parts (B) and (C) show search space size and runtime data. In (B), we include only Psy as that provably dominates each of stubborn sets and symmetry reduction; and as it empirically dominates partition-based pruning, with minor exceptions. BD and HP are not included in (B) as the number of search nodes cannot be directly compared, HP is not included in (C) as it is not competitive.

In both (B) and (C), the advantages over the baseline are reflected in substantial improvement factors, and the complementarity of star-topology decoupling vs. other methods is again reflected in per-domain strengths and weaknesses. The most significant reductions are obtained in Floortile by BDDs; in Satellite and Woodworking by Psy; and in Logistics, NoMystery, and TPP by star-topology decoupling. Figure 10 provides a per-instance view. (a) shows that D-A* (decoupled A*) incurs hardly any risk vs. A* (cases with larger runtime are rare). (b) and (c) exhibit again the complementarity of the methods (which is especially strong vs. BDDs, expectedly, as the two methods are unrelated).



(a) D-A* vs. A* (b) D-A* vs. Psy (c) D-A* vs. BDD

Figure 10: Runtime in optimal planning: D-A* using $h^{\text{LM-cut}}$ (y-axis) vs. (a) A* using $h^{\text{LM-cut}}$; (b) A* using $h^{\text{LM-cut}}$ with strong stubborn sets and symmetry reduction [102]; (c) BDD-based search with Symba [101].

Coming back to the question whether the state space reductions in Figure 8, for OPT (pricing functions) in this case, are preserved under advanced search techniques: The superiority of star-topology decoupling does disappear or reduce sometimes, where heuristic search already achieves similar reductions. This is especially pronounced in Driverlog, where OPT was able to build more state spaces than standard search, yet loses this advantage completely when $h^{\text{LM-cut}}$ is in use. However, star-topology decoupling still yields additional reductions, to varying extents, and up to several orders of magnitude. In 3 domains, it outperforms

all competing techniques. (This is without partial-order and symmetry reduction techniques. Our recent work shows that strong stubborn sets can be extended to decoupled search, inheriting, and sometimes surpassing, the best performance from both sides [103]. Presumably, this is possible for symmetry reduction as well.)

8.4. Satisficing Planning

In this subsection, we use the test suites of the IPC satisficing tracks only. For this form of planning, the most competitive algorithms are based on heuristic state space search. We use FD's lazy greedy best-first search with h^{FF} as the competitive baseline, cf. above; and we compare to the recent winners of the IPC satisficing tracks, LAMA [54] (winner IPC'08 and IPC'11) and Mercury [55] (highest scorer among non-portfolios, IPC'14). LAMA is designed for anytime plan-quality improvement, running an iteration of searches with different parameters; we focus on runtime performance here, so we run only LAMA's first search iteration, which is geared at that performance parameter. Specifically, LAMA's first search iteration runs FD's lazy greedy best-first search, with h^{FF} and an additional inadmissible heuristic function based on landmarks. Mercury runs FD's lazy greedy best-first search with an improved version of h^{FF}, based on more informed – partially delete-relaxed instead of fully delete-relaxed – abstract plans.

Consider first the coverage data *without* preferred operators in Figure 11 part (A). Star-topology decoupling yields huge advantages, especially with inverted-fork factorings which are much more effective than X-shape factorings in Rovers, Satellite, and TPP. Preferred operators, however, are extremely effective in these benchmarks. Switching them on, much of the coverage advantage disappears. Improvements remain in Childsnack, Depots, NoMystery, and Transport, where the state space reduction counter-acts weaknesses of the delete relaxation. In Transport, GBFS trivializes under star-topology decoupling: the search merely expands the states along a path to the goal.

LAMA and Mercury share the prowess in Transport, especially Mercury which does not need to search at all as the abstract plan identified by its heuristic function on the initial state is an actual plan for the input task. All three techniques also behave similarly in Depots. The complementarity across techniques is exhibited in Childsnack, NoMystery, and Pathways.

Coverage on the IPC satisficing benchmarks carries limited information as many benchmarks are solved already by the baseline (559 out of 657, compared to 332 out of 648 for optimal planning). So it is especially important here to

			(A) Coverage (Number of Tasks Solved)												(B) # Nodes				(C) Runtime			
	# Inst		No PO				IF No PO			PO					No PO	PO			No PO	PO		
Domain	All	X	GS	PP	HP	D-	IF	GS	D-	GS	PP	LA	Mer	D-	D-	LA	Mer	D-	D-	LA	Mer	D-
Csnack	20	20	0	±0	±0	±0	20	±0	±0	3	+1	-3	-3	**+3**								
Depots	22	22	14	+2	-3	**+5**	22	14	**+5**	18	±0	+2	**+3**	+2	10.7	5.7	**5.8**	5.5	11.2	2.9	4.8	**4.9**
Driverlog	20	20	18	±0	-3	**+1**				**20**	-1	**±0**	**±0**	**±0**	3.8	1.3	**1.7**	1.4	5.7	1.2	**1.6**	0.9
Elevator	50	50	48	**+2**	-35	**+2**	50	48	**+2**	**50**	-2	**±0**	**±0**	**±0**	8.7	0.9	**2253.9**	18.8	10.3	1.1	11.6	**16.0**
Floortile	40	40	**8**	±0	-4	-2	40	**8**	-2	8	**+1**	±0	±0	±0	2.6	0.3	1.0	**3.5**	2	0.2	0.5	**2.6**
Logistic	63	63	54	+4	-11	**+9**	63	54	**+9**	**63**	-1	**±0**	**±0**	**±0**	34.2	4.3	**5382.1**	40.4	18.4	4.9	**39.0**	5.5
Miconic	150	145	**145**	**±0**	-29	**±0**				**145**	**±0**	**±0**	**±0**	**±0**	2.5	0.5	**176.0**	4.3	1.1	0.4	0.9	**1.4**
Myst	19	1	0	±0	**+1**	**+1**	1	0	**+1**	**1**	**±0**	**±0**	**±0**	**±0**		0.7	0.2	**59.3**		0.9	0.6	0.0
NoMyst	20	20	9	+1	+4	**+11**				10	+3	+3	+4	**+10**	153.1	848.6	2013.8	**63060.9**	3.7	547.9	634.1	**1574.4**
Pathway	30	30	11	±0	-7	**+2**				20	+2	+4	**+10**	±0	12.7	0.3	0.3	**1.4**	5.4	0.3	0.3	0.3
PSR	50	3	3	±0	±0	±0				3	±0	±0	±0	±0	1.6	0.5	0.5	1.4				
Rovers	40	40	23	+2	-8	-1	38	21	**+17**	40	±0	±0	±0	±0	1.2	0.4	0.5	**1.3**	0.7	0.3	0.2	1.0
Satellite	36	36	30	-3	-19	+3	34	28	**+6**	**36**	-9	**±0**	**±0**	**±0**	2.7	0.2	**11.3**	1.9	1.2	0.1	1.7	**2.4**
TPP	30	29	21	+1	-18	+3	26	18	**+8**	**29**	**±0**	**±0**	**±0**	-3	3147.5	0.4	0.3	0.0	505.9	0.4	0.3	0.0
Transport	70	70	16	+10	-6	**+54**	70	16	**+54**	45	-6	+18	**+25**	**+25**	310.3	0.5	**4979.6**	331.2	170.5	0.6	14.8	**14.9**
Woodwor	50	48	47	-24	-39	**+1**	48	47	**+1**	48	±0	±0	±0	±0	2.7	**4.9**	0.2	1.2	1.2	**1.9**	0.2	0.6
Zenotrav	20	20	**20**	**±0**	-5	**±0**	18	18	±0	20	±0	±0	±0	±0	23.9	0.4	**87.1**	4.4	4.5	0.4	0.7	1.0
$\sum$	730	657	467	-5	-182	+89	430	272	**+101**	559	-12	+24	**+39**	+37								

Figure 11: Satisficing planning performance. Best results within each category ("PO" vs. "No PO") highlighted in **boldface**. Part (A) shows coverage relative to the baseline, greedy best-first search (GBFS) with h^{FF}, abbreviated as "GS" here. Parts (B) and (C) show *improvement factors* relative to the baseline, i. e., the ratio baseline/planner of (B) the per-domain sum of evaluated search nodes (calls to h^{FF}), and (C) the per-domain sum of runtime. The underlying instances in (B) and (C) are those commonly solved by the baseline and the planners shown, skipping in (C) trivial instances commonly solved in ≤ 0.1 seconds. We distinguish between using vs. not using preferred operators, "PO", as that has a large impact on performance. "IF No PO" shows data for inverted-fork factoring. "PP" is GBFS with partition-based pruning [40]. "HP" is satisficing planning with Bonet et al.'s [28] heuristic-search Petri net unfolding tool. "LA" is LAMA [54], the winner of the IPC'08 and IPC'11 satisficing tracks. "Mer" is Mercury [55], the winner (among non-portfolios) of the IPC'14 satisficing track."D-" is D-GBFS.

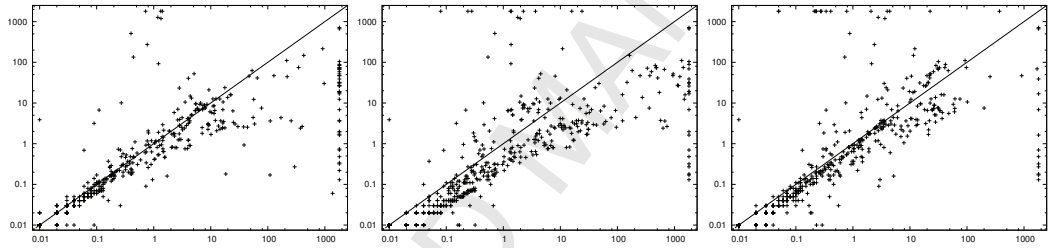
consider the more fine-grained performance measures in (B) and (C) (we do not include PP and HP here as they are not competitive).

For search space size (B), without preferred operators star-topology decoupling yields reductions in every domain, of a factor > 10 in 7 cases, of 3 orders of magnitude in TPP. With preferred operators, the reductions tend to be weaker but still present, the most striking exceptions being TPP which is easily solved by the baseline now, and NoMystery where the improvement factor grows as the commonly solved instances are larger. The large improvement factors for Mercury in Elevators, Logistics, Transport, and Zenotravel are all due to the phenomenon discussed above: the abstract plan for the initial state solves the input task. Mercury subsumes LAMA except in Woodworking. Star-topology decoupling is superior in NoMystery, where h^{FF} is not informative due to the fuel constraints, and state

space reduction is essential for success (we get back to this below).

For runtime (C), the improvement factors of all three methods are smaller, as all three incur a runtime overhead. Without preferred operators, star-topology decoupling still yields large improvements in many cases; with preferred operators, the improvements pertain mainly to Depots, Elevators, Logistics, NoMystery, and Transport. Interestingly, Mercury's search-space-size superiority, relative to star-topology decoupling, does not pay off in Elevators, Transport, and Zenotravel, as the runtime for even a single call to Mercury's heuristic function exceeds that of decoupled search.

Figure 12 provides a per-instance view. In all three comparisons, D-GBFS is more often in the advantage than not. The risk vs. the baseline (a) is relatively small, though not as small as in optimal planning. Similarly for the strength of the complementarity (b) and (c) vs. the state of the art.

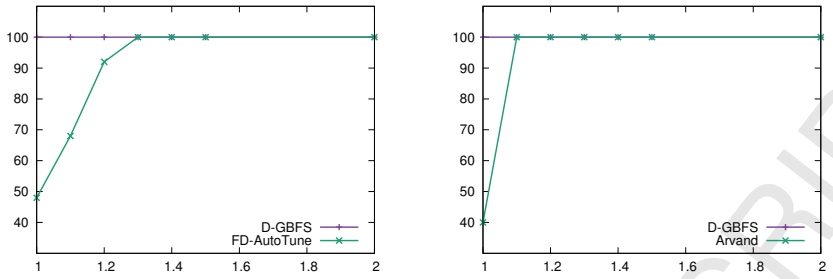


(a) D-GBFS vs. GBFS (b) D-GBFS vs. LAMA (c) D-GBFS vs. Mercury

Figure 12: Runtime in satisficing planning: D-GBFS using h^{FF} and preferred operators (y-axis) vs. (a) GBFS using h^{FF} and preferred operators; (b) LAMA [54]; (c) Mercury [55].

Coming back to the question whether the state space reductions in Figure 8, for COM (reachability functions), are preserved under advanced search techniques: The reductions get lost in many cases as greedy search with delete-relaxation heuristics is extremely effective on these benchmarks. This is especially pronounced in TPP, where the gigantic advantage of COM disappears completely when employing preferred operators (though not without them). That said, star-topology decoupling does yield reductions in half of the domains. In NoMystery, the reduction is by 5 orders of magnitude, and star-topology decoupling outperforms all competing techniques. (This is without relying on the additional heuristic functions used in LAMA and Mercury. Using these in decoupled search is straightforwardly possible via Theorem 8.)

Let us briefly extend on NoMystery, which captures an interesting kind of structure pertaining to satisficing planning in *critically constrained* situations. Consider Figure 13.



(a) NoMystery "small" (b) NoMystery "large"

Figure 13: Coverage in satisficing planning on the two original NoMystery test suites, by Nakhost et al. [96], controlling the constrainedness level C (the ratio between available and minimum-needed amount of fuel), shown on the x-axis. We compare D-GBFS using h^{FF} and preferred operators to the best-performing planners in Nakhost et al.'s comprehensive experiments on these test suites: "FD-Autotune" is an IPC'11 self-tuned variant of FD [104, 105]; Arvand is Nakhost et al.'s planner extending GBFS using h^{FF} with random walks for increased exploration.

The test suites underlying Figure 13 differ from those used in the IPC. They were created by Nakhost et al. [96] specifically to test planner performance as a function of the constrainedness level C. They created a set of base instances, and made copies of these differing only in the amount of fuel available, i. e., in the value of C. The closer C is to 1.0, the harder it becomes for greedy techniques to find a plan, as many action sequences – making sub-optimal use of fuel – will lead into unsolvable states (*dead-ends*) where insufficient fuel is left to solve the task. For $C = 1.0$, only the optimal use of fuel will succeed. Delete-relaxation heuristic functions are unable to recognize this, as the underlying abstract plans pretend that fuel is not being consumed. Nakhost et al. proposed to combat this with random walk techniques. Yet even the most competitive planners from their comprehensive experiments fail to solve many instances for $C = 1.0$. With star-topology decoupling, all instances are solved. In short: on critically constrained problems with a pronounced star topology, decoupled search is superior to the state of the art. This kind of structure is infrequent in the IPC, but is certainly practically relevant (e. g., cooperative agents with shared resources).

8.5. Proving Unsolvability

As before, we use heuristic search as the baseline. This is useful – it avoids exhausting the entire state space, which we already considered above – provided the heuristic function can identify dead-ends, returning ∞ (cf. Section 7.1). Most AI planning heuristic functions do have that capability. In particular, this is so for the three heuristic functions h^{max}, $h^{\text{LM-cut}}$, and h^{FF} considered here. We use h^{max}

as the baseline, because all three heuristic functions return ∞ in exactly the same cases, and $h^{\max}$ is the fastest among the three.

Among the state of the art in proving unsolvability, in AI planning, are *merge-and-shrink (M&S)* abstractions [78], specifically those designed by Hoffmann et al. [42] to either preserve goal reachability exactly (and thus prove unsolvability without search), or to approximate and serve as heuristic functions for identifying dead-ends during search. On our selection of (X-shape) benchmarks, most successful runs of M&S were ones proving unsolvability without search, so we include that variant ("Own+A" in Hoffmann et al.'s paper) here. The only exception is the BagTransport domain, where the "N100k M100k" variant of Hoffmann et al. – combining the M&S heuristic with $h^{\max}$ in the search – proves to be better. Therefore, we include data for the former in all domains except BagTransport, where we show "N100k M100k".

Additionally, we include the winner and runner-up of the Unsolve IPC'16 (UIPC), Aidos, respectively SymPA. Aidos is a portfolio consisting of many techniques designed to prove planning tasks unsolvable [106]. SymPA uses symbolic perimeter abstraction heuristics [107]. We furthermore include all planners tested for optimal planning above, as these techniques are often suitable also for proving unsolvability. In particular, the most competitive non-M&S planner in Hoffmann et al.'s [42] experiments was the BDD-based planner Symba.

			(A) Coverage (Number of Tasks Proved Unsolvable)																		(B) # Nodes		(C) Runtime								
	# Inst						Blind							$h^{\max}$																	
Domain	All	X	BD	MS	SP	Ai	A*	P	Sy	Psy	PP	HP	D-	A*	P	Sy	Psy	PP	HP	D-	Psy	D-	BD	MS	SP	Ai	P	Sy	Psy	PP	D-
Unsolvable Benchmarks: From UIPC'16																															
BTransp	29	29	7	6	8	**22**	7	-1	±0	-1	±0	-3	+4	6	±0	±0	±0	±0	-2	+4	1.2	**141**	4.3	0.5	22.0	**1513**	0.5	0.9	0.5	0.8	73.0
NoMyst	24	24	6	14	6	14	2	±0	±0	±0	±0	-2	**+16**	2	±0	+1	±0	±0	-2	**+16**	1.5	**873**	1.1	51.2	0.6	16.2	0.5	1.2	0.3	0.8	**61.4**
Rovers	20	20	11	**15**	13	14	7	±0	±0	-3	±0	-6	+1	7	±0	±0	±0	±0	-6	+1	2.1	**9.3**	130	**369**	97.9	14.9	0.5	0.8	0.2	0.7	1.2
Unsolvable Benchmarks: IPC Mystery, Others Extended from [42]																															
Myst	9	3	0	2	0	**3**	0	±0	±0	+2	±0	±0	±0	0	+1	±0	+2	±0	±0	±0											
NoMyst	40	40	19	**40**	20	**40**	9	-1	+1	-3	±0	-6	**+31**	12	±0	+2	-1	±0	-4	**+28**	2.1	**7190**	9.8	291	6.4	129	0.5	1.7	0.2	0.8	**395**
Rovers	40	40	18	32	18	**38**	4	±0	-1	-1	±0	-4	±0	9	-1	-1	-1	-1	-5	+1	1.9	**22.5**	11.5	**682**	15.5	34.9	0.6	0.8	0.4	0.7	2.7
$\sum$	162	156	61	109	65	**131**	29	-2	±0	-6	±0	-21	+52	36	±0	+2	±0	-1	-19	+50											

Figure 14: Proving unsolvability performance. Best results within each category highlighted in **boldface**. Part (A) shows coverage relative to the baseline A*. Parts (B) and (C) show improvement factors relative to the baseline, i. e., the ratio baseline/planner of (B) the per-domain sum of evaluated search nodes (calls to $h^{\max}$), and (C) the per-domain sum of runtime. The underlying instances in (B) and (C) are those commonly proved unsolvable by the baseline and the planners shown, skipping in (C) trivial instances commonly proved unsolvable in ≤ 0.1 seconds. "MS" is the best of Hoffmann et al.'s [42] "Own+A" and "N100k M100k" merge-and-shrink abstractions. "SP"/"Ai" are the SymPA/Aidos planners from the unsolvability IPC [107, 106]. All other abbreviations are as in Figure 9.

Consider Figure 14. Star-topology decoupling excels in NoMystery as before (recall that in both the UIPC, and Hoffmann et al.'s version, these are the instances originally by Nakhost et al. [96], but with constrainedness levels < 0). In the UIPC'16 version of the domain, it is even able to outperform advanced systems like Aidos and SymPA, particularly designed to prove planning tasks unsolvable. Except in Rovers and Mystery, decoupled search performs better than all competitors excluding Aidos. In Rovers, BD, M&S, and SymPA are better, in Mystery, it's only M&S and the combination of partial-order and symmetry reduction (Psy).

Coming back to the question whether the state space reductions in Figure 8, for COM (reachability functions), are preserved under advanced search techniques: From the limited collection of unsolvable planning benchmarks, only NoMystery has strong state space reductions. These are preserved when using $h^{\max}$. M&S abstractions are equally effective in NoMystery, by very different means. (We remark that, as one would expect, there are natural cases where M&S abstractions are much inferior to star-topology decoupling; for example, a simple unsolvable variant of TPP has this property.)

9. Conclusion

Star-topology decoupling is a new state space reduction method for reachability analysis in compactly described discrete transition systems. It exploits a form of conditional independence between leaf components in a star topology, given a fixed transition sequence of the center component. The decoupled state space branches over center transitions only, and maintains leaf state spaces separately. Thanks to that separation, state space size can be exponentially reduced. Blow-ups can also occur in principle, but that can be avoided with suitable pruning techniques. The method has exponential separations from all previous state space reduction methods in theory. In practice, on standard AI Planning benchmarks with a pronounced star topology, it outperforms previous methods in terms of the most basic measure of reduction power, exhausting the state space. The empirical advantage reduces when using orthogonal search enhancements, but is still manifested in situations where previous methods are weak.

From these results, it is clear that star-topology decoupling is a promising new method for reachability analysis. Its suitability for pronounced star topologies – with many leaf components that exhibit individual mobility, not affecting the center state in every transition – makes it an exciting method to try on problems that come with such structure by design. Multi-agent systems of cooperative agents

with shared variables, e. g. shared resources, are one example. The application to model checking also is very promising, as star topology is a pervasive design paradigm in distributed and concurrent computing settings. Client-server architectures are a classical example. A highly relevant recent direction are concurrent programs under weak memory models (e. g. [108, 109, 110, 111]). Processes run locally on separate processors, and read/write operations are performed on local memory. Explicit "fence" operations wait for local memory to be fully committed to global memory, and a consistent view of memory needs to be guaranteed. Observe that star-topology decoupling is especially suited here, as there may be many processors (leaf components), and many local operations (leaf mobility), and the objective is to verify correctness (exhaust the state space). Key challenges include the adaptation to the system-description languages used in model checking, and the extension to checking temporal properties, beyond reachability (i. e., beyond safety).

It may be possible to better leverage star-topology decoupling in planning through better factoring methods. Initial progress in this direction was recently made [112]. An interesting question is whether recent methods for identifying planning backdoors can be adapted for that purpose.

Star-topology decoupling is complementary to previous search enhancements, and there are manifold opportunities for combining it with these. The authors have already devised combinations with strong stubborn sets [103], symmetry reduction [113], and BDDs for representing leaf state spaces [114]. Beyond these, the combination with unfolding methods seems particularly promising.

Last but not least, star-topology decoupling can be viewed as a form of *target-profile factoring*, that looks for a particular structural profile suited to specialized combinatorial search algorithms. Beyond star profiles, this suggests an entirely new way of exploiting structure in AI Planning:

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.

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

We thank Carmel Domshlak for his contribution in extending decoupled search beyond fork topologies. We thank Álvaro Torralba for discussions, for his contribution to dominance pruning methods, and for contributing the proof idea for Lemma 8.

Appendix A. Proofs and Additional Examples

Appendix A.1. Factorings

Proposition 1. *Let Π be a planning task. Then every strict-star factoring is a star factoring, but not vice versa.*

Proof: Assume that $\mathcal{F}$ is a strict-star factoring, and let a be an action where $\mathcal{V}(\mathsf{eff}(a)) \cap F^C = \emptyset$. If there were two different leaf factors $F_1^L \neq F_2^L$ both affected by a, then $\mathsf{IG}_\Pi(\mathcal{F})$ would contain the arc $F_1^L \to F_2^L$, in contradiction. So there exists a leaf factor F_1^L with $\mathcal{V}(\mathsf{eff}(a)) \subseteq F_1^L$. But then, $\mathcal{V}(\mathsf{pre}(a)) \subseteq F_1^L \cup F^C$, because again, otherwise, if a was also preconditioned on $F_2^L \neq F_1^L$, $\mathsf{IG}_\Pi(\mathcal{F})$ would contain the arc $F_1^L \to F_2^L$ in contradiction.

Not every star factoring is a strict-star factoring because star factorings, but not strict-star factorings, allow action effects to touch multiple leaf factors. For example, if we add to the Scaling example an action that loads all objects, as well as moves the truck, simultaneously, then the grouping $F^C = \{t\}$ and $\mathcal{F}^L = \{\{o_1\}, \ldots, \{o_n\}\}$ is a star factoring but not a strict-star factoring. □

Theorem 1 (Fork & Inverted-Fork Factorings). *Let Π be a planning task. Then fork and inverted-fork factorings exist if and only if $|\mathcal{F}_\Pi^{\text{SCC}}| > 1$. In that case, the maximum number of fork leaves equals the number of leaf vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$, and the maximum number of inverted-fork leaves equals the number of root vertices in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$.*

Proof: Say Π has variables V. The "only if" direction in the first part of the claim holds because any fork or inverted-fork factoring must be coarser than $\mathcal{F}_\Pi^{\text{SCC}}$. The "if" direction follows from the second part of the claim.

We prove the second part of the claim for fork factorings; the argument for inverted forks is symmetric because those are equivalent to forks in the modification of $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ where the direction of each arc is inverted.

Denote by K the maximum number of fork leaves. Denote by K' the number of leaves in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$.

$K \geq K'$ holds simply because we obtain a fork factoring by setting the leaves in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ to be the leaf factors, and taking the remaining state variables to form the center.

$K \leq K'$ holds because any fork leaf must be closed under following arcs in the causal graph, and hence every fork leaf must contain at least one leaf in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$. Precisely, let $\mathcal{F}$ be any fork factoring, and let F^L be any leaf factor of $\mathcal{F}$. Let F, F' be factors in $\mathcal{F}_\Pi^{\text{SCC}}$ such that $F \to F'$ is an arc in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$. If $F \subseteq F^L$, then we must have $F' \subseteq F^L$: otherwise, either $F' \subseteq F^C$ in contradiction, or $F' \subseteq F_2^L$ for some other leaf factor F_2^L in contradiction. As $\mathcal{F}$ is coarser than $\mathcal{F}_\Pi^{\text{SCC}}$, there exists at least one F where $F \subseteq F^L$. Applying the argument transitively starting from F, we obtain a leaf F'' in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$ so that $F'' \subseteq F^L$. But then, as the leaf factors in $\mathcal{F}$ are disjoint, there cannot be more leaf factors than leaves in $\mathsf{IG}_\Pi(\mathcal{F}_\Pi^{\text{SCC}})$, concluding the argument. □

Appendix A.2. Additional Example Walkthroughs

Example 10. *Consider the Vanilla example, with the inverted-fork factoring $F^C = \{o\}$, $\mathcal{F}^L = \{\{t_A\}, \{t_B\}\}$, where the object is in the center and each truck is a leaf. In contrast to the fork factoring (Example 4), center actions now have preconditions on leaves but not vice versa, and center states are object positions while pricing functions pertain to truck positions.*

For $\pi_0^C = \langle\rangle$ and the corresponding decoupled state s_0, $\mathsf{center}[s_0] = \{o = l_1\}$. As the leaves now don't have preconditions on the center, the pricing function corresponds to cheapest paths within each leaf component, starting from the current leaf states $\{t_A = l_1\}$ and $\{t_B = l_3\}$. Hence the prices are 0 for $\{t_A = l_1\}$, 1 for $\{t_A = l_2\}$, 2 for $\{t_A = l_3\}$, 0 for $\{t_B = l_3\}$, 1 for $\{t_B = l_2\}$, and 2 for $\{t_B = l_1\}$.

The applicable center actions in s_0 are load(t_A, l_1) and load(t_B, l_1): their center preconditions are satisfied, and their leaf preconditions have finite prices, $\mathsf{prices}[s_0](\{t_A = l_1\}) = 0 < \infty$ and $\mathsf{prices}[s_0](\{t_B = l_1\}) = 2 < \infty$. Say we apply load$(t_B, l_1)$. Denote the resulting center path and decoupled state by π_1^C and s_1.

We have $\mathsf{center}[s_1] = \{o = t_B\}$. The pricing function remains unchanged regarding t_A. However, the prices regarding t_B change completely: the center action we have chosen uses a leaf precondition on t_B, namely $t_B = l_1$. This commits that leaf to move to (any) one of its finite-price leaf states complying with that leaf precondition. Here, there is just one such leaf state, $\{t_B = l_1\}$. In terms of compliant paths, of the compliant paths with respect to π_0^C the ones supporting $\{t_B = l_3\}$ and $\{t_B = l_2\}$ are not compliant anymore, because they don't provide the leaf precondition $t_B = l_1$. All cheapest π_1^C-compliant paths for t_B start with

the prefix $\langle$*move*(t_B, l_3, l_2), *move*$(t_B, l_2, l_1)\rangle$, *to be scheduled before load*(t_B, l_1). *After that, they may append any cheapest sequence of* t_B *moves to some other leaf state, to be scheduled behind load*(t_B, l_1). *So the prices in* s_1 *are* 2 *for* $\{t_B = l_1\}$, 3 *for* $\{t_B = l_2\}$, *and* 4 *for* $\{t_B = l_3\}$. *Note that the prices have increased relative to the previous decoupled state: while, in forks, prices* decrease *monotonically, in inverted-forks they* increase *monotonically.*

In s_1, *the applicable center actions are all unloads of the object from* t_B, *at any location. In particular, we can already reach a decoupled goal state, by applying unload*(t_B, l_3). *In the outcome decoupled state* s_2, $\mathsf{center}[s_2] = \{o = l_3\}$ *which is a center goal state as desired; and* t_B *is now committed to* l_2 *which it must reach by a compliant path, so its new price tags are* 4 *for* $\{t_B = l_3\}$, 5 *for* $\{t_B = l_2\}$, *and* 6 *for* $\{t_B = l_1\}$. *The global plan extracted is the one using* t_B, *at cost* $4 + 2 = 6$.

Observe that the pricing functions here are extremely volatile. Every center action commits some leaf to a unique leaf state, completely changing its pricing function in the next state. This is typical of inverted-fork factorings. The reachability functions, on the other hand, remain constant throughout our example here, and for any inverted fork with strongly connected leaf state spaces: from whatever leaf state a leaf is committed to, all other leaf states are reachable so the prices are always $< \infty$.

Example 11. *Consider the NoEmpty example, where truck moves have the additional precondition* $o = t$ *(no empty-truck moves). Say we use the strict-star factoring* $F^C = \{t_A, t_B\}$, $\mathcal{F}^L = \{\{o\}\}$, *which has bidirectional dependencies (compare Figure 2 (b), page 11).*

The initial decoupled state s_0 *is* $\mathsf{center}[s_0] = \{t_A = l_1, t_B = l_3\}$, *with prices* 0 *for* $\{o = l_0\}$, 1 *for* $\{o = t_A\}$, ∞ *elsewhere. Whereas in the Vanilla example we could apply either of move*(t_A, l_1, l_2) *or move*(t_B, l_3, l_2), *now only the first choice is possible because move*(t_B, l_3, l_2) *has the leaf precondition* $o = t_B$ *whose price is infinite. Applying move*(t_A, l_1, l_2) *to obtain* s_1, *we get* $\mathsf{center}[s_1] = \{t_A = l_2, t_B = l_3\}$. *The price is* 1 *for* $\{o = t_A\}$, *keeping its previous price because this is the leaf state the center action choice committed* o *to. The price of* $\{o = l_1\}$, *however, is now* ∞. *Through the commitment to* $\{o = t_A\}$, *we lost the previous price* 0, *and given the new center state there is no compliant path re-achieving* $\{o = l_1\}$. *For* $\{o = l_2\}$ *on the other hand, we now get price* 2 *due to the leaf path* $\langle$*load*(t_A, l_1), *unload*$(t_A, l_2)\rangle$, *which is now compliant thanks to the new center precondition* $t_A = l_2$ *provided in* s_1. *All other prices remain infinite.*

Note how this combines fork behavior with inverted-fork behavior: the price for $\{o = l_2\}$ *has decreased while that for* $\{o = l_1\}$ *has increased. Due to the*

now bidirectional dependency, the prices are neither monotonically decreasing nor monotonically increasing.

Appendix A.3. Decoupled State Space Correctness

To capture decoupled-state reachability, we introduce an intermediate concept, *embedded states*, exhibiting the link between member states and decoupled states. Instead of an explicit leaf-state assignment, embedded states contain a link to the respective compliant-path graph vertex:

Definition 12 (Embedded State). *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. For a decoupled state s in $\Theta^{\mathcal{F}}_{\Pi}$, an **embedded state** in s is a function $\widehat{p}$ on $\mathcal{F}$, mapping F^C to $\mathsf{center}[s]$, and mapping each $F^L \in \mathcal{F}^L$ to a vertex s^L_n in $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$ with $\mathsf{prices}[s](s^L) < \infty$, where $n := |\pi^C[s]|$. The set of all embedded states of s is the **embedded hypercube** of s, denoted $[\widehat{s}]$.*

*The **initial embedded state**, $\widehat{I}$, maps F^C to $\mathsf{center}[I^{\mathcal{F}}] = I[F^C]$ and, for each $F^L \in \mathcal{F}^L$, maps F^L to the vertex $I[F^L]_0$ in $\mathsf{CompG}_{\Pi}[\langle\rangle, F^L]$.*

Given a decoupled state s, as $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$ contains exactly one vertex s^L_n for every $s^L \in S^L[F^L]$, the member states and embedded states of s are in one-to-one correspondence. Given a decoupled state s and member state $p \in [s]$, we denote the unique corresponding embedded state by $\widehat{p}$, and vice versa.

Intuitively, embedded states $\widehat{p}$ serve to "track the progress of the corresponding states p through the decoupled state space". We formalize this through a notion of *embedded transitions*:

Definition 13 (Embedded Transitions). *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. Let s be a decoupled state in $\Theta^{\mathcal{F}}_{\Pi}$, and $\widehat{p} \in [\widehat{s}]$ an embedded state. Then $\widehat{p} \xrightarrow{a} \widehat{q}$ is an **embedded transition***

(i) on F^L in $\Theta^{\mathcal{F}}_{\Pi}$, if $a \in A^L[F^L] \setminus A^C$, $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$ contains an arc $\widehat{p}(F^L) \xrightarrow{a} \widehat{q}(F^L)$, and for all $F^L \neq F \in \mathcal{F}$ we have $\widehat{p}(F) = \widehat{q}(F)$;

(ii) on F^C in $\Theta^{\mathcal{F}}_{\Pi}$, if $a \in A^C$, $s \xrightarrow{a} t$ is a transition in $\Theta^{\mathcal{F}}_{\Pi}$, and for all $F^L \in \mathcal{F}^L$ we have that $\mathsf{CompG}_{\Pi}[\pi^C[t], F^L]$ contains an arc $\widehat{p}(F^L) \xrightarrow{0} \widehat{q}(F^L)$.

*We refer to paths of embedded transitions as **embedded paths**. The **cost** of an embedded path $\widehat{\pi}$, denoted $\mathsf{cost}(\widehat{\pi})$, is the summed-up cost of its transition labels.*

Note here that, necessarily by the construction of $\widehat{q}$ and compliant path graphs, $\widehat{q} \in [\widehat{s}]$ in (i), and $\widehat{q} \in [\widehat{t}]$ in (ii). Decoupled-state reachability is captured in the following form:

Lemma 6. *Let Π be a planning task, and $\mathcal{F}$ a star factoring. Let $S^{\mathcal{F}}$ be the decoupled states in $\Theta^{\mathcal{F}}_{\Pi}$. For any $s \in S^{\mathcal{F}}$, $\widehat{p} \in [\widehat{s}]$, and $\widehat{q}$ reachable from $\widehat{p}$ in $\Theta^{\mathcal{F}}_{\Pi}$, there exists $t \in S^{\mathcal{F}}$ such that $\widehat{q} \in [\widehat{t}]$ and t is reachable from s in $\Theta^{\mathcal{F}}_{\Pi}$. Vice versa, for any $s, t \in S^{\mathcal{F}}$ where t is reachable from s in $\Theta^{\mathcal{F}}_{\Pi}$, and for any $\widehat{q} \in [\widehat{t}]$, there exists $\widehat{p} \in [s]$ such that $\widehat{q}$ is reachable from $\widehat{p}$ in $\Theta^{\mathcal{F}}_{\Pi}$.*

Proof: For the first part of the claim, t and $\widehat{q}$ as specified must exist simply because the individual transitions on an embedded path from $\widehat{p}$ to $\widehat{q}$ all follow decoupled transitions (F^C) respectively compliant-path graph arcs (F^L) present in $\Theta^{\mathcal{F}}_{\Pi}$. For the second part of the claim, if $\pi^{\mathcal{F}}$ is a decoupled path from s to t, then we can backchain from $\widehat{q}$ through the compliant-path graphs along $\pi^{\mathcal{F}}$ to obtain the desired embedded state $\widehat{p}$ in s. □

Having clarified the basics of embedded states and how they capture reachability, let us get back to the link with member states. The core observation is that, like the member states and embedded states themselves, their transitions also are in one-to-one correspondence:

Lemma 7. *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. Let s be a decoupled state in $\Theta^{\mathcal{F}}_{\Pi}$. Then, for any member state $p \in [s]$ and action a, $p \xrightarrow{a} q$ is a transition in Π if and only if $\widehat{p} \xrightarrow{a} \widehat{q}$ is an embedded transition in $\Theta^{\mathcal{F}}_{\Pi}$.*

Proof: From left to right, say a is applicable to p and $q = p[\![a]\!]$. We distinguish two cases. First, a is a non-center action, $a \notin A^C$. Then, as $\mathcal{F}$ is a star factoring, a affects a single leaf factor F^L, $a^L \in A^L[F^L] \setminus A^C$. As a is applicable to p, we have $\mathsf{pre}(a)[F^C] = \mathsf{center}[s][\mathcal{V}(\mathsf{pre}(a)) \cap F^C]$ and $\mathsf{pre}(a)[F^L] = p[\mathcal{V}(\mathsf{pre}(a)) \cap F^L]$. Therefore, by Definition 3 (i), the compliant path graph $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$ layer at time n corresponding to s contains the arc $p[F^L]_n \xrightarrow{a} q[F^L]_n$, which establishes the desired embedded transition.

Second, say a is a center action, $a \in A^C$. As a is applicable to p, for every $F^L \in \mathcal{F}^L$ where $\mathsf{pre}(a)[F^L] \neq \emptyset$, there exists a finite-price leaf state in s, namely $p[F^L]$. Hence there exists a transition $s \xrightarrow{a} t$ in $\Theta^{\mathcal{F}}_{\Pi}$. Furthermore, for each F^L, by Definition 3 (ii) the compliant path graph $\mathsf{CompG}_{\Pi}[\pi^C[t], F^L]$ contains the arc $p[F^L]_n \xrightarrow{0} q[F^L]_{n+1}$. Together, these establish the desired embedded transition.

From right to left, say $\widehat{p} \xrightarrow{a} \widehat{q}$ is an embedded transition in $\Theta^{\mathcal{F}}_{\Pi}$. Distinguishing the same two cases, if $a^L \in A^L[F^L] \setminus A^C$ then $\widehat{p} \xrightarrow{a} \widehat{q}$ is an F^L transition. By Definition 13 (i), $\widehat{p}$ and $\widehat{q}$ differ only in terms of taking a single arc on F^L, so p and q differ only on F^L. By Definition 3 (i), $\mathsf{pre}(a)[F^C] = \mathsf{center}[s][\mathcal{V}(\mathsf{pre}(a)) \cap F^C]$, $\mathsf{pre}(a)[F^L] = p[\mathcal{V}(\mathsf{pre}(a)) \cap F^L]$, and $p[F^L][\![a]\!] = q[F^L]$. But this immediately implies that a is applicable to p and $q = p[\![a]\!]$, as desired.

Say finally that $a \in A^C$ so $\widehat{p} \xrightarrow{a} \widehat{q}$ is an F^C transition. By Definition 13 (ii), $\widehat{q}$ results from $\widehat{p}$ by updating the center state according to a, and taking the arc $\widehat{p}(F^L) \xrightarrow{0} \widehat{q}(F^L)$ for every F^L. By Definition 3 (ii) for every F^L we have $\mathsf{pre}(a)[F^L] \subseteq p[\mathcal{V}(\mathsf{pre}(a)) \cap F^L]$ and $p[F^L][\![a]\!] = q[F^L]$. But then, again a is applicable to p and $q = p[\![a]\!]$, concluding the proof. □

We are now ready to prove the two core lemmas:

Lemma 2. *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. Let s be a reachable decoupled state in $\Theta^{\mathcal{F}}_{\Pi}$. Then:*

(i) $[s]$ is exactly the set of states p for which there exists a path π, from I to p in Θ_{Π}, whose center-action subsequence is $\pi^C[s]$.

(ii) For every $p \in [s]$, the cost of a cheapest such path π is

$$\mathsf{cost}(\pi^C[s]) + \sum_{F^L \in \mathcal{F}^L} \mathsf{prices}[s](p[F^L])$$

Proof: To prove (i), say first that $p \in [s]$. Consider the corresponding embedded state $\widehat{p}$. As all compliant path graph vertices in $\widehat{p}$ are reached at s, for every F^L there must exist a path $\pi[F^L]$ from $I[F^L]_0$ to $\widehat{p}(F^L)$ in $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$. From the collection of these paths, along with $\pi^C[s]$, we obtain an embedded path $\widehat{\pi}$ from $\widehat{I}$ to $\widehat{p}$: simply interleave the embedded F^C transitions induced by $\pi^C[s]$ with the embedded F^L transitions induced by the $\pi[F^L]$. With Lemma 7, from $\widehat{\pi}$ we obtain a path π as desired.

Say now that π is a path in Π from I to some p, where the center action subsequence of π is $\pi^C[s]$. With Lemma 7, we obtain an embedded path $\widehat{\pi}$ from $\widehat{I}$ to $\widehat{p}$. Collecting the F^L transitions from $\widehat{\pi}$ for each F^L, clearly we get paths $\pi[F^L]$ from $I[F^L]_0$ to $\widehat{p}(F^L)$ in $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$. Therefore, $p \in [s]$ as desired.

Claim (ii) now follows directly, because the above shows that, for every $p \in [s]$, the paths π as specified are in one-to-one correspondence with $\pi^C[s]$ augmented with the possible selections of $\mathsf{CompG}_{\Pi}[\pi^C[s], F^L]$ paths from $I[F^L]_0$ to

$p[F^L]_n$, where $n := |\pi^C[s]|$. Thus, by the definition of pricing functions, the cheapest such π has exactly the specified cost. □

Lemma 3. *Let Π be a planning task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. Let p be a reachable state in Π, and let π be a path reaching p. Then there exists a reachable decoupled state s in $\Theta^{\mathcal{F}}_{\Pi}$ so that $p \in [s]$, and $\pi^C[s]$ is the center action subsequence of π.*

Proof: With Lemma 7, π corresponds to an embedded path $\widehat{\pi}$ from $\widehat{I}$ to $\widehat{p}$. By Lemma 6, there exists a decoupled state s such that $\widehat{p} \in [\widehat{s}]$, and s is reachable from $I^{\mathcal{F}}$ in $\Theta^{\mathcal{F}}_{\Pi}$. Clearly, the decoupled transitions taken in reaching s, according to the proof of Lemma 6, correspond exactly to the center action subsequence of π. □

Appendix A.4. Decoupled State Space Size

In what follows, we consider N-vectors $v \in R^N$ over some subset $R \subseteq \mathbb{R}^{0+}$ of non-negative reals. For the special case of $N = 1$, we identify v with $v[1]$. We consider (possibly infinite) sequences $\vec{v} = v_0, v_1, v_2, \ldots$ of vectors. We define the relation $\succ$ over vectors by saying that $v \succ v'$ iff there exists a vector position $1 \leq k \leq N$ so that $v[k] > v'[k]$. We say that a vector sequence $\vec{v}$ is *descending* if, whenever v precedes v' in the sequence, $v \succ v'$. We say that R *has an infinite descending N-sequence* if there exists an infinite descending sequence of N-vectors.

Theorem 5 (Finiteness under Dominance Pruning). *Let Π be a planning task, and $\mathcal{F}$ a star factoring. Under ancestor dominance pruning, $\Theta^{R\mathcal{F}}_{\Pi}$ is finite.*

Proof: Consider the non-pruned paths $\pi^{\mathcal{F}}$ in $\Theta^{R\mathcal{F}}_{\Pi}$. Observe that such paths necessarily are descending: some prices along $\pi^{\mathcal{F}}$ must descend each time we encounter the same center state, as otherwise the new state would be dominated by some previous state. We prove that there is no infinite descending path, i. e., every $\pi^{\mathcal{F}}$ has finite length. As every $s \in S^{R\mathcal{F}}$ must be the endpoint of such a path, and as the branching factor is finite, this proves the claim.

Assume to the contrary that there is an infinite descending path $\pi^{\mathcal{F}}$. As the number of different center states is finite, there must exist a center state s^C visited infinitely often on $\pi^{\mathcal{F}}$. Denote by $\vec{s} = s_0, s_1, \ldots$ the sub-sequence of decoupled states along $\pi^{\mathcal{F}}$ where $\mathsf{center}[s_i] = s^C$.

Collect, from each s_i, the vector p_i of leaf state prices (using some arbitrary order of leaf states to fix the ordering of vector positions). Denoting by N the

number of leaf states, $\vec{p} := p_0, p_1, \ldots$ is a sequence of N-vectors over $\mathbb{R}^{0+} \cup \{\infty\}$. More precisely, $\vec{p}$ is a sequence of N-vectors over possible plan cost values, i. e., over $R \cup \{\infty\}$ where R contains $\sum_{i=1}^{n} c(a_i)$ for any finite sequence $\langle a_1, \ldots, a_n \rangle$ of actions in Π.

Proposition 5 below shows that R has no infinite descending 1-sequence. Lemma 8 below shows that, given this, $R \cup \{\infty\}$ has no infinite descending N-sequence for any N.

However, by construction, whenever p precedes p' on $\vec{p}$ then there exists a vector position k so that $p[k] > p'[k]$. That is, $\vec{p}$ is descending, in contradiction, showing the claim. □

Proposition 5. *Let Π be a planning task. Let $R \subseteq \mathbb{R}^{0+}$ be the set of numbers that contains $\sum_{i=1}^{n} c(a_i)$ for any finite sequence $\langle a_1, \ldots, a_n \rangle$ of actions in Π. Then R does not have an infinite descending 1-sequence.*

Proof: Consider any sequence $\vec{v} = v_0, v_1, v_2, \ldots$ of 1-vectors over R. Then, in particular, $v_i < v_0$ for all $i > 0$. But, for any $C \in \mathbb{R}^{0+}$, there is only a finite number of values $\sum_{i=1}^{n} c(a_i) < C$. This is because any non-0 cost action a can occur at most $\lfloor C/c(a) \rfloor$ times on $\langle a_1, \ldots, a_n \rangle$. □

Lemma 8. *Let $R \subseteq \mathbb{R}$ be a set of numbers that has no infinite descending 1-sequence. Let $\vec{v}$ be a descending sequence of N-vectors over $R \cup \{\infty\}$. Then $\vec{v}$ is finite.*

Proof: Let $K \subseteq \{1, \ldots, N\}$ be an arbitrary subset of vector positions. Denote by $\vec{v}[K, \infty]$ the subsequence of vectors v on $\vec{v}$ where $v[k] \neq \infty$ iff $k \in K$. In words, consider the subsequence of vectors that fits the "∞-profile" given by K. We show that $\vec{v}[K, \infty]$ is finite, which proves the claim as there is only a finite number of profiles.

Construct the vector sequence $\vec{v}[K, \infty]|_K$ by projecting each element of $\vec{v}[K, \infty]$ onto the position subset K. Then $\vec{v}[K, \infty]|_K$ is a sequence of vectors over $R \cup \{\infty\}$. By construction, as $\vec{v}[K, \infty]$ is a descending sequence, and because the elements of $\vec{v}[K, \infty]$ all agree on the positions outside K, we have that $\vec{v}[K, \infty]|_K$ is a descending sequence. It thus remains to show that every descending sequence of finite vectors over R is finite. We prove this by induction over N.

The induction base case, $N = 1$, holds by prerequisite as R has no infinite descending 1-sequence.

For the inductive case, assume that there is no infinite descending sequence of N-position vectors over R. We show that there is no infinite descending sequence of $N+1$-position vectors over R.

Let $\vec{w} = w_0, w_1, w_2, \ldots$ be any descending sequence of $N+1$-position vectors over R. Denote $w_0 = (c^1, \ldots, c^{N+1})$, where each c^j is a constant, i. e., $c^j \in R$.

Let $j \in \{1, \ldots, N+1\}$ be arbitrary. Denote $C^j := \{c' \in R \mid c' < c^j\}$. Then C^j is finite because otherwise we could sequence its elements into an infinite descending 1-sequence over R. Let $c' \in C^j$ be arbitrary. Denote by $\vec{w}[j, c']$ the subsequence of vectors w on $\vec{w}$ where the j-th position has value c'. Obviously, every w_i for $i > 0$ must be contained in at least one $\vec{w}[j, c']$. We show that each $\vec{w}[j, c']$ is finite. As there is a finite number of choices of j' and c', this proves the claim.

Denote $K := \{1, \ldots, N+1\} \setminus \{j\}$. Construct the vector sequence $\vec{w}[j, c']|_K$ by projecting each element of $\vec{w}[j, c']$ onto the position subset K. By construction, as $\vec{w}[j, c']$ is a descending sequence, and because the elements of $\vec{w}[j, c']$ all agree on the single position j that is outside K, we have that $\vec{w}[j, c']|_K$ is a descending sequence. As $\vec{w}[j, c']|_K$ is a sequence of vectors with N positions, by induction assumption, $\vec{w}[j, c']|_K$ is finite. Hence $\vec{w}[j, c']$ is finite as desired, concluding the argument. □

Lemma 4. *Let Π be a planning task, $\mathcal{F}$ a star factoring, and s a decoupled state. Then s is solvable if and only if at least one $p \in [s]$ is solvable.*

Proof: From left to right, say the decoupled goal state t is reachable from s in $\Theta^{\mathcal{F}}_{\Pi}$. Clearly, there exists a goal state $q \in [t]$. By Lemma 6, there exists an embedded state $\widehat{p} \in [\widehat{s}]$ such that $\widehat{q}$ is reachable from $\widehat{p}$ in $\Theta^{\mathcal{F}}_{\Pi}$. By Lemma 7, q is reachable from p in Π. Hence the state p is solvable as desired.

From right to left, say the goal state q is reachable from p in Π. By Lemma 7, $\widehat{q}$ is reachable from $\widehat{p}$ in $\Theta^{\mathcal{F}}_{\Pi}$. By Lemma 6, there exists a decoupled state t such that $\widehat{q} \in [\widehat{t}]$ and t is reachable from s in $\Theta^{\mathcal{F}}_{\Pi}$. As q is a goal state, t is a decoupled goal state, as desired. □

Proposition 3. *Given a planning task Π and a star factoring $\mathcal{F}$, it is **co-NP**-complete to decide whether reachable decoupled states $t_1, \ldots, t_n$ cover a reachable decoupled state s.*

Proof: Membership follows directly from the results by Hoffmann and Kupferschmid [65] for general hypercube covering problems.

Hardness follows by reduction from the complement of SAT, extending Hoffmann and Kupferschmid's argument by a simple construction of Π and $\mathcal{F}$. Assume a CNF formula ϕ with propositional variables $P_1, \ldots, P_n$ and clauses $C_1, \ldots, C_m$. The construction of Π includes state variables $p_1, \ldots, p_n$, each with domain $\{u, 0, 1\}$ where u is the initial value; there furthermore is a variable c with domain $\{u, 0, 1, \ldots, m\}$. The goal does not matter for our purposes. The factoring $\mathcal{F}$ has center $\{c\}$ and leaves $\{\{p_1\}, \ldots, \{p_n\}\}$.

The actions are as follows. For each clause C_j there is a center action a^C_j which is applicable to the initial state, and which allows to generate a hypercube corresponding to the truth-value assignments disallowed by c_j. Specifically, we set $\mathsf{pre}(a^C_j) = \{c = u\}$, and $\mathsf{eff}(a^C_j) = \{c = j\}$. Furthermore, we include leaf actions a^L_l, one for each literal $l \in C_j$, with $\mathsf{pre}(a^C_l) = \{c = j\}$, and $\mathsf{eff}(a^C_l) = \{\bar{l}\}$ where $\bar{l}$ is the opposite of l, i. e., $p_i = 1$ if $l = \neg P_i$, and $p_i = 0$ if $l = P_i$. Finally, we include leaf actions a^L_{ij0} and a^L_{ij1} for each variable P_i that does not occur in C_j, with $\mathsf{pre}(a^L_{ij0}) = \mathsf{pre}(a^L_{ij1}) = \{c = j\}$, $\mathsf{eff}(a^L_{ij0}) = \{p_i = 0\}$, and $\mathsf{eff}(a^L_{ij1}) = \{p_i = 1\}$. Observe that, once a^C_j has been applied, the hypercube t_j of reached leaf states over the variables p_i corresponds exactly to those assignments over P_i which do not satisfy C_j.

We finally include a center action a^C_0 which is applicable to the initial state, and allows to generate a hypercube corresponding to *all* truth-value assignments. Specifically, we set $\mathsf{pre}(a^C_0) = \{c = u\}$, and $\mathsf{eff}(a^C_0) = \{c = 0\}$, and we include leaf actions a^L_{i00} and a^L_{i01} for each $1 \leq i \leq n$, with $\mathsf{pre}(a^L_{i00}) = \mathsf{pre}(a^L_{i01}) = \{c = 0\}$, $\mathsf{eff}(a^L_{i00}) = \{p_i = 0\}$, and $\mathsf{eff}(a^L_{i01}) = \{p_i = 1\}$. Observe that, once a^C_0 has been applied, the hypercube s of reached leaf states over the variables p_i corresponds exactly to the space of all assignments over P_i.

Consider the time point in search where search has explored each of the alternatives $a^C_1, \ldots, a^C_m$ (applied each of these actions to the initial state separately), and now explores the alternative a^C_0. Then all-visited hypercube pruning checks whether $t_1, \ldots, t_m$ cover s. The latter is the case iff ϕ is unsatisfiable. □

References

[1] D. Gnad, J. Hoffmann, Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search, in: R. Brafman, C. Domshlak, P. Haslum, S. Zilberstein (Eds.), Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15), AAAI Press, 2015, pp. 88–96.

[2] D. Gnad, J. Hoffmann, C. Domshlak, From fork decoupling to star-topology decoupling, in: L. Lelis, R. Stern (Eds.), Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15), AAAI Press, 2015, pp. 53–61.

[3] Á. Torralba, D. Gnad, P. Dubbert, J. Hoffmann, On state-dominance criteria in fork-decoupled search, in: [116], 2016.

[4] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, 2004.

[5] G. Lamperti, M. Zanella, Diagnosis of Active Systems, Kluwer Academic Publishers, 2003.

[6] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 2001.

[7] R. E. Korf, W. Zhang, Divide-and-conquer frontier search applied to optimal sequence alignment, in: H. A. Kautz, B. Porter (Eds.), Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI'00), AAAI Press, Austin, TX, USA, 2000, pp. 910–916.

[8] A. Valmari, Stubborn sets for reduced state space generation, in: Proceedings of the 10th International Conference on Applications and Theory of Petri Nets, 1989, pp. 491–515.

[9] P. Godefroid, P. Wolper, Using partial orders for the efficient verification of deadlock freedom and safety properties, in: Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91), 1991, pp. 332–342.

[10] A. Valmari, A stubborn attack on state explosion, Formal Methods in System Design 1 (1992) 297–322.

[11] D. Peled, All from one, one for all: on model checking using representatives, in: Proceedings of the 5th International Conference on Computer Aided Verification (CAV'93), 1993, pp. 409–423.

[12] P. Godefroid, Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem, volume 1032 of *Lecture Notes in Computer Science*, Springer, 1996.

[13] S. Edelkamp, S. Leue, A. Lluch-Lafuente, Partial-order reduction and trail improvement in directed model checking, International Journal on Software Tools for Technology Transfer 6 (2004) 277–301.

[14] R. Nissim, U. Apsel, R. I. Brafman, Tunneling and decomposition-based state reduction for optimal planning, in: L. D. Raedt (Ed.), Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12), IOS Press, Montpellier, France, 2012, pp. 624–629.

[15] M. Wehrle, M. Helmert, Y. Alkhazraji, R. Mattmüller, The relative pruning power of strong stubborn sets and expansion core, in: [115], 2013.

[16] M. Wehrle, M. Helmert, Efficient stubborn sets: Generalized algorithms and selection strategies, in: S. Chien, M. Do, A. Fern, W. Ruml (Eds.), Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14), AAAI Press, 2014.

[17] P. Starke, Reachability analysis of petri nets using symmetries, Journal of Mathematical Modelling and Simulation in Systems Analysis 8 (1991) 293–304.

[18] E. A. Emerson, A. P. Sistla, Symmetry and model-checking, Formal Methods in System Design 9 (1996) 105–131.

[19] M. Fox, D. Long, The detection and exploitation of symmetry in planning problems, in: [124], 1999, pp. 956–961.

[20] N. Pochter, A. Zohar, J. S. Rosenschein, Exploiting problem symmetries in state-based planners, in: [117], 2011.

[21] C. Domshlak, M. Katz, A. Shleyfman, Enhanced symmetry breaking in cost-optimal planning as forward search, in: [121], 2012.

[22] K. L. McMillan, Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits, in: G. von Bochmann, D. K. Probst (Eds.), Proceedings of the 4th International Workshop on Computer Aided Verification (CAV'92), Lecture Notes in Computer Science, Springer, 1992, pp. 164–177.

[23] J. Esparza, S. Römer, W. Vogler, An improvement of mcmillan's unfolding algorithm, Formal Methods in System Design 20 (2002) 285–310.

[24] S. L. Hickmott, J. Rintanen, S. Thiébaux, L. B. White, Planning via petri net unfolding, in: [119], 2007, pp. 1904–1911.

[25] B. Bonet, P. Haslum, S. L. Hickmott, S. Thiébaux, Directed unfolding of petri nets, Transactions on Petri Nets and Other Models of Concurrency 1 (2008) 172–198.

[26] J. Esparza, K. Heljanko, Unfoldings – A Partial-Order Approach to Model Checking, Monographs in Theoretical Computer Science, Springer, 2008.

[27] P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodríguez, S. Schwoon, Efficient unfolding of contextual Petri nets, Theoretical Computer Science 449 (2012) 2–22.

[28] B. Bonet, P. Haslum, V. Khomenko, S. Thiébaux, W. Vogler, Recent advances in unfolding technique, Theoretical Computer Science 551 (2014) 84–101.

[29] E. D. Sacerdoti, Planning in a hierarchy of abstraction spaces, Artificial Intelligence 5 (1974) 115–135.

[30] C. Knoblock, Automatically generating abstractions for planning, Artificial Intelligence 68 (1994) 243–302.

[31] A. L. Lansky, L. Getoor, Scope and abstraction: Two criteria for localized planning, in: S. Mellish (Ed.), Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), Morgan Kaufmann, Montreal, Canada, 1995, pp. 1612–1619.

[32] E. Amir, B. Engelhardt, Factored planning, in: G. Gottlob (Ed.), Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Morgan Kaufmann, Acapulco, Mexico, 2003, pp. 929–935.

[33] R. Brafman, C. Domshlak, Factored planning: How, when, and when not, in: Y. Gil, R. J. Mooney (Eds.), Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI'06), AAAI Press, Boston, Massachusetts, USA, 2006, pp. 809–814.

[34] E. Kelareva, O. Buffet, J. Huang, S. Thiébaux, Factored planning using decomposition trees, in: [119], 2007, pp. 1942–1947.

[35] R. I. Brafman, C. Domshlak, From one to many: Planning for loosely coupled multi-agent systems, in: [122], 2008, pp. 28–35.

[36] E. Fabre, L. Jezequel, P. Haslum, S. Thiébaux, Cost-optimal factored planning: Promises and pitfalls, in: R. I. Brafman, H. Geffner, J. Hoffmann, H. A. Kautz (Eds.), Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10), AAAI Press, 2010, pp. 65–72.

[37] R. Nissim, R. I. Brafman, C. Domshlak, A general, fully distributed multi-agent planning algorithm, in: W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, S. Sen (Eds.), Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), IFAAMAS, 2010, pp. 1323–1330.

[38] M. Crosby, M. Rovatsos, R. P. A. Petrick, Automated agent decomposition for classical planning, in: [115], 2013.

[39] R. Brafman, C. Domshlak, On the complexity of planning for agent teams and its implications for single agent planning, Artificial Intelligence 198 (2013) 52–71.

[40] R. Nissim, R. Brafman, Distributed heuristic forward search for multi-agent planning, Journal of Artificial Intelligence Research 51 (2014) 293–332.

[41] D. Wang, B. C. Williams, tburton: A divide and conquer temporal planner, in: [120], 2015, pp. 3409–3417.

[42] J. Hoffmann, P. Kissmann, Á. Torralba, "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability, in: T. Schaub (Ed.), Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14), IOS Press, Prague, Czech Republic, 2014.

[43] V. Khomenko, M. Koutny, Towards an efficient algorithm for unfolding petri nets, in: Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01), 2001, pp. 366–380.

[44] C. Rodríguez, S. Schwoon, Cunf: A tool for unfolding and verifying petri nets with read arcs, in: Proceedings of the 11th International Symposium

on Automated Technology for Verification and Analysis (ATVA'13), 2013, pp. 492–495.

[45] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: Portfolio-based algorithm selection for sat, Journal of Artificial Intelligence Research 32 (2008) 565–606.

[46] I. Cenamor, T. de la Rosa, F. Fernández, IBACOP and IBACOP2 planner, in: IPC 2014 planner abstracts, 2014, pp. 35–38.

[47] J. Pearl, Heuristics, Morgan Kaufmann, 1984.

[48] D. V. McDermott, Using regression-match graphs to control search in planning, Artificial Intelligence 109 (1999) 111–159.

[49] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (2001) 5–33.

[50] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.

[51] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs, Journal of Artificial Intelligence Research 20 (2003) 239–290.

[52] M. Helmert, The Fast Downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246.

[53] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: What's the difference anyway?, in: A. Gerevini, A. Howe, A. Cesta, I. Refanidis (Eds.), Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09), AAAI Press, 2009, pp. 162–169.

[54] S. Richter, M. Westphal, The LAMA planner: Guiding cost-based anytime planning with landmarks, Journal of Artificial Intelligence Research 39 (2010) 127–177.

[55] C. Domshlak, J. Hoffmann, M. Katz, Red-black planning: A new systematic approach to partial delete relaxation, Artificial Intelligence 221 (2015) 73–114.

[56] C. Bäckström, B. Nebel, Complexity results for SAS^+ planning, Computational Intelligence 11 (1995) 625–655.

[57] T. Bylander, The computational complexity of propositional STRIPS planning, Artificial Intelligence 69 (1994) 165–204.

[58] P. Jonsson, C. Bäckström, Incremental planning, in: European Workshop on Planning, 1995.

[59] R. Brafman, C. Domshlak, Structure and complexity in planning with unary operators, Journal of Artificial Intelligence Research 18 (2003) 315–349.

[60] M. Katz, C. Domshlak, Structural patterns heuristics via fork decomposition, in: [122], 2008, pp. 182–189.

[61] M. R. Garey, D. S. Johnson, Computers and Intractability—A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.

[62] M. R. Henzinger, T. A. Henzinger, P. W. Kopke, Computing simulations on finite and infinite graphs, in: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95), IEEE Computer Society, 1995, pp. 453–462.

[63] D. Hall, A. Cohen, D. Burkett, D. Klein, Faster optimal planning with partial-order pruning, in: [115], 2013.

[64] Á. Torralba, J. Hoffmann, Simulation-based admissible dominance pruning, in: [118], 2015, pp. 1689–1695.

[65] J. Hoffmann, S. Kupferschmid, A covering problem for hypercubes, in: L. P. Kaelbling (Ed.), Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05), Morgan Kaufmann, Edinburgh, UK, 2005, pp. 1523–1524.

[66] L. D. Moura, N. Bjørner, Z3: An efficient SMT solver, in: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08), 2008, pp. 337–340.

[67] S. Trüg, J. Hoffmann, B. Nebel, Applying automatic planning systems to airport ground-traffic control — a feasibility study, in: S. Biundo, T. Frühwirth, G. Palm (Eds.), KI-04: Advances in Artificial Intelligence, Springer-Verlag, Ulm, Germany, 2004, pp. 183–197.

[68] J. Rintanen, Symmetry reduction for SAT representations of transition systems, in: E. Giunchiglia, N. Muscettola, D. Nau (Eds.), Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03), Morgan Kaufmann, Trento, Italy, 2003, pp. 32–41.

[69] R. E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers 35 (1986) 677–691.

[70] K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

[71] S. Edelkamp, Taming numbers and durations in the model checking integrated planning system, Journal of Artificial Intelligence Research 20 (2003) 195–238.

[72] S. Edelkamp, P. Kissmann, On the complexity of BDDs for state space search: A case study in connect four, in: [117], 2011.

[73] J. C. Culberson, J. Schaeffer, Pattern databases, Computational Intelligence 14 (1998) 318–334.

[74] S. Edelkamp, Planning with pattern databases, in: [123], 2001, pp. 13–24.

[75] A. Felner, R. Korf, S. Hanan, Additive pattern database heuristics, Journal of Artificial Intelligence Research 22 (2004) 279–318.

[76] P. Haslum, A. Botea, M. Helmert, B. Bonet, S. Koenig, Domain-independent construction of pattern database heuristics for cost-optimal planning, in: A. Howe, R. C. Holte (Eds.), Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07), AAAI Press, Vancouver, BC, Canada, 2007, pp. 1007–1012.

[77] J. Seipp, M. Helmert, Counterexample-guided Cartesian abstraction refinement, in: [115], 2013, pp. 347–351.

[78] M. Helmert, P. Haslum, J. Hoffmann, R. Nissim, Merge & shrink abstraction: A method for generating lower bounds in factored state spaces, Journal of the Association for Computing Machinery 61 (2014).

[79] F. Bacchus, Q. Yang, Downward refinement and the efficiency of hierarchical problem solving, Artificial Intelligence 71 (1994) 43–100.

[80] M. Helmert, Fast (diagonally) downward, in: IPC 2006 planner abstracts, 2006.

[81] P. Haslum, Reducing accidental complexity in planning problems, in: [119], 2007, pp. 1898–1903.

[82] J. Tozicka, J. Jakubuv, M. Svatos, A. Komenda, Recursive polynomial reductions for classical planning, in: A. Coles, A. Coles, S. Edelkamp, D. Magazzeni, S. Sanner (Eds.), Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16), AAAI Press, 2016, pp. 317–325.

[83] C. Domshlak, Y. Dinitz, Multi-agent offline coordination: Structure and complexity, in: [123], 2001, pp. 34–43.

[84] M. Helmert, A planning heuristic based on causal graph analysis, in: S. Koenig, S. Zilberstein, J. Koehler (Eds.), Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04), Morgan Kaufmann, Whistler, Canada, 2004, pp. 161–170.

[85] M. Katz, C. Domshlak, New islands of tractability of cost-optimal planning, Journal of Artificial Intelligence Research 32 (2008) 203–288.

[86] O. Giménez, A. Jonsson, The complexity of planning problems with simple causal graphs, Journal of Artificial Intelligence Research 31 (2008) 319–351.

[87] O. Giménez, A. Jonsson, Planning over chain causal graphs for variables with domains of size 5 is NP-hard, Journal of Artificial Intelligence Research 34 (2009) 675–706.

[88] O. Giménez, A. Jonsson, The influence of k-dependence on the complexity of planning, Artificial Intelligence 177-179 (2012) 25–45.

[89] M. Katz, E. Keyder, Structural patterns beyond forks: Extending the complexity boundaries of classical planning, in: J. Hoffmann, B. Selman (Eds.), Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12), AAAI Press, Toronto, ON, Canada, 2012, pp. 1779–1785.

[90] M. Aghighi, P. Jonsson, S. Ståhlberg, Tractable cost-optimal planning over restricted polytree causal graphs, in: [120], 2015, pp. 3225–3231.

[91] M. Kronegger, S. Ordyniak, A. Pfandler, Backdoors to planning, in: C. E. Brodley, P. Stone (Eds.), Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14), AAAI Press, Austin, Texas, USA, 2014, pp. 2300–2307.

[92] M. Kronegger, S. Ordyniak, A. Pfandler, Variable-deletion backdoors to planning, in: [120], 2015, pp. 3305–3312.

[93] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (1968) 100–107.

[94] P. Haslum, H. Geffner, Admissible heuristics for optimal planning, in: S. Chien, R. Kambhampati, C. Knoblock (Eds.), Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00), AAAI Press, Menlo Park, Breckenridge, CO, 2000, pp. 140–149.

[95] J. Hoffmann, J. Porteous, L. Sebastia, Ordered landmarks in planning, Journal of Artificial Intelligence Research 22 (2004) 215–278.

[96] H. Nakhost, J. Hoffmann, M. Müller, Resource-constrained planning: A Monte Carlo random walk approach, in: [121], 2012, pp. 181–189.

[97] M. Helmert, Concise finite-domain representations for PDDL planning tasks, Artificial Intelligence 173 (2009) 503–535.

[98] A. L. Blum, M. L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1997) 279–298.

[99] H. Kautz, B. Selman, Unifying SAT-based and graph-based planning, in: [124], 1999, pp. 318–325.

[100] D. Long, M. Fox, The 3rd international planning competition: Results and analysis, Journal of Artificial Intelligence Research 20 (2003) 1–59.

[101] Á. Torralba, C. Linares López, D. Borrajo, Abstraction heuristics for symbolic bidirectional search, in: [116], 2016, pp. 3272–3278.

[102] M. Wehrle, M. Helmert, A. Shleyfman, M. Katz, Integrating partial order reduction and symmetry elimination for cost-optimal classical planning, in: [118], 2015.

[103] D. Gnad, M. Wehrle, J. Hoffmann, Decoupled strong stubborn sets, in: [116], 2016, pp. 3110–3116.

[104] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: An automatic algorithm configuration framework, Journal of Artificial Intelligence Research 36 (2009) 267–306.

[105] C. Fawcett, M. Helmert, H. Hoos, E. Karpas, G. Röger, J. Seipp, FD-Autotune: Automated configuration of Fast Downward, in: IPC 2011 planner abstracts, 2011, pp. 31–37.

[106] J. Seipp, F. Pommerening, S. Sievers, M. Wehrle, Fast downward aidos, in: UIPC 2016 planner abstracts, 2016, pp. 28–38.

[107] Á. Torralba, Sympa: Symbolic perimeter abstractions for proving unsolvability, in: UIPC 2016 planner abstracts, 2016, pp. 8–11.

[108] B. Jonsson, State-space exploration for concurrent algorithms under weak memory orderings, SIGARCH Computer Architecture News 36 (2008) 65–71.

[109] A. Linden, P. Wolper, A verification-based approach to memory fence insertion in PSO memory systems, in: N. Piterman, S. A. Smolka (Eds.), Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13), Springer-Verlag, 2013, pp. 339–353.

[110] O. Travkin, A. Mütze, H. Wehrheim, SPIN as a linearizability checker under weak memory models, in: Proceedings of the 9th International Haifa Verification Conference (HVC'13), 2013, pp. 311–326.

[111] Y. A. Alrahman, M. Andric, A. Beggiato, A. Lluch-Lafuente, Can we efficiently check concurrent programs under relaxed memory models in maude?, in: Revised Selected Papers of the 10th International Workshop on Rewriting Logic and Its Applications (WRLA'14), 2014, pp. 21–41.

[112] D. Gnad, V. Poser, J. Hoffmann, Beyond forks: Finding and ranking star factorings for decoupled search, in: C. Sierra (Ed.), Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), AAAI Press/IJCAI, 2017.

[113] D. Gnad, Á. Torralba, A. Shleyfman, J. Hoffmann, Symmetry breaking in star-topology decoupled search, in: Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17), AAAI Press, 2017.

[114] D. Gnad, Á. Torralba, J. Hoffmann, Symbolic leaf representation in decoupled search, in: A. Fukunaga, A. Kishimoto (Eds.), Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17), AAAI Press, 2017.

[115] D. Borrajo, S. Fratini, S. Kambhampati, A. Oddi (Eds.), Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13), AAAI Press, Rome, Italy, 2013.

[116] S. Kambhampati (Ed.), Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16), AAAI Press/IJCAI, 2016.

[117] W. Burgard, D. Roth (Eds.), Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11), AAAI Press, San Francisco, CA, USA, 2011.

[118] Q. Yang (Ed.), Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15), AAAI Press/IJCAI, 2015.

[119] M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), Morgan Kaufmann, Hyderabad, India, 2007.

[120] B. Bonet, S. Koenig (Eds.), Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15), AAAI Press, 2015.

[121] B. Bonet, L. McCluskey, J. R. Silva, B. Williams (Eds.), Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12), AAAI Press, 2012.

[122] J. Rintanen, B. Nebel, J. C. Beck, E. Hansen (Eds.), Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08), AAAI Press, 2008.

[123] A. Cesta, D. Borrajo (Eds.), Proceedings of the 6th European Conference on Planning (ECP'01), Springer-Verlag, 2001.

[124] M. Pollack (Ed.), Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99), Morgan Kaufmann, Stockholm, Sweden, 1999.