An Analysis of Dominance Pruning in Decoupled Search

Submission # 12

Abstract

Decoupled search is a state compression technique that tackles the state space explosion problem inherent in the process of solving classical planning problems by search. Similar to symbolic search approaches, decoupled search nodes, or *decoupled states*, do not correspond to individual states, but to entire *sets* of states. As a result, duplicate state pruning, a standard method in explicit state search, cannot be applied in a straightforward manner. Instead, prior work has introduced the notion of *dominance pruning*, where a decoupled state is pruned if all explicit states it represents are contained in an already visited decoupled state. In this work, we shed further light on dominance pruning in decoupled search, and suggest two improvements: in optimal decoupled search using A*, we incorporate the g-value of the current search node into the dominance check, increasing the potential for pruning at very little computational cost. Second, with the observation that the pruning can sometimes dominate the overall runtime, we introduce exact duplicate checking on decoupled states, which can benefit from standard notions like hashing to be more efficient. We implemented our improvements and show empirically that they are indeed beneficial across several IPC benchmark domains.

Introduction

In classical planning, the most popular approach to solve planning tasks is heuristic search in the explicit state space (Bonet and Geffner 1999). Heuristic search, however, suffers from the state explosion problem that arises from the fact that the size of the state space of a task can be exponential in the size of its description. Several methods have been introduced to tackle this explosion, such as partial-order reduction (Valmari 1989; Godefroid and Wolper 1991; Edelkamp, Leue, and Lluch-Lafuente 2004; Alkhazraji et al. 2012; Wehrle et al. 2013; Wehrle and Helmert 2014), symmetry breaking (Starke 1991; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012), dominance pruning (Hall et al. 2013; Torralba and Hoffmann 2015; Torralba 2017), or symbolic representations (Bryant 1986; Edelkamp and Helmert 1999; Torralba et al. 2017). In this work, we look into a recent addition to this set of techniques, namely star-topology decoupled state space search, or decoupled search for short (Gnad and Hoffmann 2018).

Decoupled search is a form of factored planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006; 2008; Fabre et al. 2010) that partitions the variables of a planning task into components such that the causal dependencies between the components form a star topology. The *center* C of this topology can interact arbitrarily with the other components, the *leaves* $\mathcal{L} = \{L_1, \dots, L_n\}$, while any interaction between leaves has to involve the center, too. A *decoupled state* $s^{\mathcal{F}}$ then corresponds to a unique *center state*, an assignment to C, and a non-empty set of *leaf states* (assignments to a leaf factor L_i) for each L_i. The *member states* of $s^{\mathcal{F}}$, i. e., the set of explicit states represented by it, result from all combinations of leaf states across leaf factors, sharing the same center state. Thereby, a decoupled state represents exponentially many explicit states, leading to a reduction in search effort. Prior work has shown that there exist task families where this reduction for decoupled search is exponentially larger than that of partial-order reduction (Gnad, Hoffmann, and Wehrle 2019), symmetry breaking (Gnad et al. 2017), symbolic representations (Gnad and Hoffmann 2018), and Petri-net unfolding (Gnad and Hoffmann 2019).

In decoupled search, search nodes, i. e., decoupled states $s^{\mathcal{F}}$, correspond to *sets of states*, namely the member states of $s^{\mathcal{F}}$. Therefore, the standard concept of duplicate elimination, ignoring a state that has already been visited (on a cheaper path) to avoid repeated work, cannot be applied so easily. More importantly, it is not as effective as in explicit state search, because two decoupled states are only equal if the entire sets of member states they represent are equal. Therefore, prior work on decoupled search only considered dominance pruning (Torralba et al. 2016; Gnad and Hoffmann 2018), where a decoupled state $s_1^{\mathcal{F}}$ that contains the member states S_1 is *dominated by* a decoupled state $s_2^{\mathcal{F}}$ that represents the set of states S_2 if $S_1 \subseteq S_2$, respecting the so-called *pricing function* of $s_1^{\mathcal{F}}$ and $s_2^{\mathcal{F}}$ in case of optimal planning. Then, if each member state of $s_2^{\mathcal{F}}$ is reached in $s_1^{\mathcal{F}}$ with at most the price it has in $s_2^{\mathcal{F}}$, we can safely prune $s_1^{\mathcal{F}}$, because all states reachable from it can be reached from $s_2^{\mathcal{F}}$ as well, with smaller or equal cost.

In this work, we analyze the criteria under which a decoupled state dominates another one in more detail. More specifically, in optimal planning, we also incorporate the g-value of the decoupled states into the dominance check when comparing the leaf-state prices. This increases the poten-

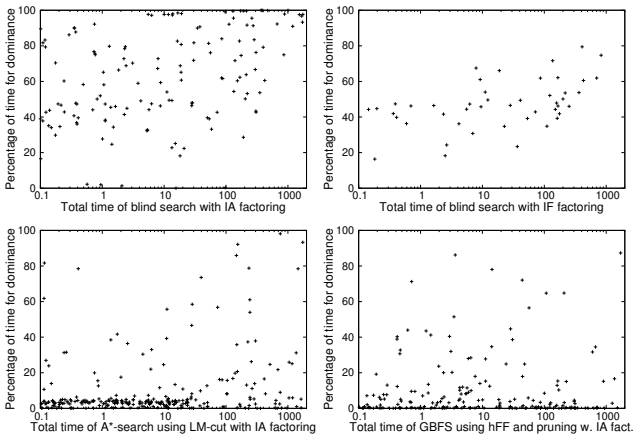

Figure 1: Ratio of the overall search time spent on dominance checking.

tial for pruning, because a decoupled state $s_1^{\mathcal{F}}$ with a much higher g-value than a state $s_2^{\mathcal{F}}$ can now be dominated by $s_2^{\mathcal{F}}$ even if some leaf-state prices of $s_1^{\mathcal{F}}$ are lower. Interestingly, this comes at almost no computational overhead.

In general, however, we observe that the dominance pruning can make up for a big portion of the overall runtime of decoupled search. Figure 1 shows the ratio of the overall search time that is spend on dominance checking as a function of overall search time for several configurations. Here, *GBFS* is greedy best-first search, *hFF* is the h^{FF} heuristic used with preferred operator *pruning* (Hoffmann and Nebel 2001), and we use the factoring strategies based on incident arcs (IA) and inverted forks (IF) introduced by Gnad, Poser, and Hoffmann (2017). The plots clearly show that the time to check dominance of decoupled states can make up a substantial part of the overall runtime, up to almost 100% for blind search. Even when using the LM-cut heuristic (Helmert and Domshlak 2009) or h^{FF} in satisficing planning, there is still a large amount of time that is spent on dominance checking. The two plots in the top also indicate that the factoring method can make quite a difference. The reason why computing dominance takes so much time is that it cannot be checked by standard means such as hashing. Instead, the prices of all leaf states need to be compared individually.

We experiment with an implementation of exact duplicate checking, which, although less powerful in terms of pruning, can benefit runtime-wise a lot from hashing. In our experimental evaluation, we see that in domains where the pruning is similarly strong, the computational advantage of hashing indeed pays off. Regarding the incorporation of g-values in the dominance check, our results show that there is no drawback of using it, and the search space under pruning can be significantly smaller.

Background

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

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

During an A* search, we denote by $g(s)$ the minimum cost of a path on which a state s was reached from I. Note that the g-value of a state can get reduced during the search, in case a cheaper path from I to s is generated.

Decoupled Search

Decoupled search is a technique developed to avoid the combinatorial explosion of having to enumerate all possible variable assignments of causally independent parts of a planning task. It does so by partitioning the state variables into a *factoring* $\mathcal{F}$, whose elements are called *factors*. By imposing a structural requirement on the interaction between these factors, namely a *star topology*, decoupled search can efficiently handle cross-factor dependencies. A *star factoring* is one that has a *center* $C \in \mathcal{F}$ that interacts arbitrarily with all other factors $L \in \mathcal{L} := \mathcal{F} \setminus C$, called *leaves*, but where the only interaction between leaves is via the center.

Actions affecting C, i. e., with an effect on a variable in C, are called *center actions*, denoted $\mathcal{A}^C$, and those affecting a leaf are called *leaf actions*, denoted $\mathcal{A}^{\mathcal{L}}$. The actions that affect a particular leaf $L \in \mathcal{L}$ are denoted $\mathcal{A}^L$. A sequence of center actions applicable to I in the projection onto C is a *center path*, a sequence of leaf actions affecting L, applicable to I in the projection onto L, is a *leaf path*. A complete assignment to C is called a *center state*, an assignment to an $L \in \mathcal{L}$ is called a *leaf state*. The set of all leaf states is denoted $S^{\mathcal{L}}$, and that of a particular leaf L is denoted S^L. We define the set of leaf actions *enabled* in a center state s^C as $\mathcal{A}^{\mathcal{L}}|_{s^C} := \{a^L \mid a^L \in \mathcal{A}^{\mathcal{L}} \wedge s^C \models \mathsf{pre}(a^L)[C]\}$.

A *decoupled state* $s^{\mathcal{F}}$ is a pair $\langle \mathsf{center}(s^{\mathcal{F}}), \mathsf{prices}(s^{\mathcal{F}}) \rangle$, where $\mathsf{center}(s^{\mathcal{F}})$ is a center state, and $\mathsf{prices}(s^{\mathcal{F}}) : S^{\mathcal{L}} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ is the *pricing function*, that assigns every leaf state a non-negative price. The pricing function is maintained during decoupled search in a way so that the price of a leaf state s^L is the cost of a cheapest leaf path that ends in s^L and that is *compliant*, i. e., that can be scheduled alongside the center path executed up to $s^{\mathcal{F}}$. By $S^{\mathcal{F}}$ we denote the set of all decoupled states.

We say that a decoupled state $s^{\mathcal{F}}$ *satisfies* a condition p, denoted $s^{\mathcal{F}} \models p$, iff (i) $\mathsf{center}(s^{\mathcal{F}}) \models p[C]$ and

(ii) for every $L \in \mathcal{L}$ there exists $s^L \in S^L$ s.t. $s^L \models p[L]$ and $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$. The *initial decoupled state* $s_0^{\mathcal{F}}$ is defined as $s_0^{\mathcal{F}} := \langle \mathsf{center}(s_0^{\mathcal{F}}), \mathsf{prices}(s_0^{\mathcal{F}}) \rangle$, where $\mathsf{center}(s_0^{\mathcal{F}}) = I[C]$. The pricing function is given, for each $L \in \mathcal{L}$, as $\mathsf{prices}(s_0^{\mathcal{F}})[s_0^L] = 0$ where $s_0^L = I[L]$; and elsewhere as $\mathsf{prices}(s_0^{\mathcal{F}})[s^L] = c_{s_0^{\mathcal{F}}}(s_0^L, s^L)$ where $c_{s_0^{\mathcal{F}}}(s_0^L, s^L)$ is the cost of a cheapest path of $\mathcal{A}^L|_{\mathsf{center}(s_0^{\mathcal{F}})} \setminus \mathcal{A}^C$ actions from s_0^L to s^L. If no such path exists $c_{s_0^{\mathcal{F}}}(s_0^L, s^L) = \infty$. The set of *decoupled goal states* $S_G^{\mathcal{F}}$ is $S_G^{\mathcal{F}} := \{s_G^{\mathcal{F}} \mid s_G^{\mathcal{F}} \models G\}$.

In decoupled search, only center actions are applied. A center action a^C is *applicable* in a decoupled state $s^{\mathcal{F}}$ if $s^{\mathcal{F}} \models \mathsf{pre}(a^C)$. By $S_{a^C}^L(s^{\mathcal{F}})$ we define the set of leaf states of L in $s^{\mathcal{F}}$ that *comply* with the leaf precondition of a^C, i. e., $S_{a^C}^L(s^{\mathcal{F}}) := \{s^L \mid s^L \models \mathsf{pre}(a^C)[L] \wedge \mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty\}$. Applying a^C to $s^{\mathcal{F}}$ results in the decoupled state $t^{\mathcal{F}} = s^{\mathcal{F}}[\![a^C]\!]$ as follows: $\mathsf{center}(t^{\mathcal{F}}) := \mathsf{center}(s^{\mathcal{F}})[\![a^C]\!]$, $\mathsf{prices}(t^{\mathcal{F}})[t^L] := \min_{s^L \in S_{a^C}^L(s^{\mathcal{F}})}(\mathsf{prices}(s^{\mathcal{F}})[s^L] + c_{t^{\mathcal{F}}}(u^L, t^L))$ where $s^L[\![a^C]\!] = u^L$.

By $\pi^C(s^{\mathcal{F}})$ we denote the center path that starts in $s_0^{\mathcal{F}}$ and ends in $s^{\mathcal{F}}$. Accordingly, we define the g-value of $s^{\mathcal{F}}$ as $g(s^{\mathcal{F}}) = \mathsf{cost}(\pi^C(s^{\mathcal{F}}))$, the cost of its center path.

A decoupled state $s^{\mathcal{F}}$ represents a set of explicit states. This set takes the form of a *hypercube* whose dimensions are the leaf factors $\mathcal{L}$. Formally, such a hypercube is defined as follows:

Definition 1 (Hypercube) *Let Π be a planning task, and $\mathcal{F}$ a star factoring. Then a state s of Π is a* member state *of a decoupled state $s^{\mathcal{F}}$, if $s[C] = \mathsf{center}(s^{\mathcal{F}})$ and, for all leaves $L \in \mathcal{L}$, $\mathsf{prices}(s^{\mathcal{F}})[s[L]] < \infty$. We say that s has* price $\mathsf{price}(s^{\mathcal{F}}, s)$ *in $s^{\mathcal{F}}$, where $\mathsf{price}(s^{\mathcal{F}}, s) := \sum_{L \in \mathcal{L}} \mathsf{prices}(s^{\mathcal{F}})[s[L]]$. The* hypercube *of $s^{\mathcal{F}}$, denoted $[s^{\mathcal{F}}]$, is the set of all member states of $s^{\mathcal{F}}$.*

The hypercube of $s^{\mathcal{F}}$ captures both, the reachability and the prices of all member states s of $s^{\mathcal{F}}$. For every member state s of a decoupled state $s^{\mathcal{F}}$, we can construct the *global plan*, i. e., the sequence of actions that starts in I and ends in s by augmenting $\pi^C(s^{\mathcal{F}})$ with cheapest-compliant leaf paths, i. e., leaf action sequences that lead to the pricing function of $s^{\mathcal{F}}$. The cost of member states in a hypercube only takes into account the cost of the leaf actions, since center action costs are not included in the pricing function. The cost of a plan reaching a member state s of $s^{\mathcal{F}}$ from I can be computed as follows: $\mathsf{cost}(s^{\mathcal{F}}, s) = g(s^{\mathcal{F}}) + \mathsf{price}(s^{\mathcal{F}}, s)$.

Dominance Pruning

In explicit state search, duplicate checking is performed to avoid unnecessary repeated handling of the same state. This can be implemented efficiently by means of hashing functions: if a state is re-visited during search – and, in case of optimal planning using A*, the path on which it is reached is not cheaper than its current g-value – the new state can be pruned safely.

Prior work on decoupled search has only considered dominance pruning instead of exact duplicate checking (Torralba et al. 2016; Gnad and Hoffmann 2018). In dominance pruning, instead of duplicate states, the search prunes decoupled states that are *dominated* by an already visited decoupled state (with lower g-value). A decoupled state $s^{\mathcal{F}}$ dominates another decoupled state $t^{\mathcal{F}}$ if (1) $[t^{\mathcal{F}}] \subseteq [s^{\mathcal{F}}]$ and (2) for all $s^L \in S^{\mathcal{L}} : \mathsf{prices}(s^{\mathcal{F}})[s^L] < \mathsf{prices}(t^{\mathcal{F}})[s^L]$. Note that (2) is only required for optimal planning. In satisficing search, e. g. using greedy best-first search (GBFS), we can simply set the price of all reached leaf states to 0, ignoring the leaf action costs completely. In practice these checks are performed by first comparing the center states $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$ via hashing, followed by a component-wise comparison of the prices of reached leaf states.

In the following, we will look into exact duplicate checking for decoupled search, showing how an efficient hashing can be implemented. We then show how the g-value of a decoupled state can be incorporated into the dominance check, increasing the potential for dominance pruning.

Exact Duplicate Checking in Decoupled Search

In decoupled search a search node, i. e., a decoupled state $s^{\mathcal{F}}$, does not represent a single state, but a set of states, namely its hypercube $[s^{\mathcal{F}}]$. As a consequence, duplicate checking is less effective. This is because the chances of finding a decoupled state with the exact same hypercube (including leaf state prices) are small, even if explicit state search generates many duplicates. Furthermore, care must be taken when hashing decoupled states, to properly take into account both reachability *and* prices. So the question is how much more efficient duplicate checking can become, in comparison to comparing leaf state prices one-by-one.

In order to hash decoupled states, we need a canonical form that provides a unique representation of a decoupled state. We achieve this by, prior to the search, constructing *all* reachable leaf states s^L for each leaf L, over-approximating reachability by projecting the task onto L. This ignores all interaction between the center and the leaf, assuming that action preconditions on $\mathcal{V} \setminus L$ are always achieved. This results in what is called the *leaf state space* for every leaf factor. Given the leaf state spaces, we assign a unique ID for every leaf state, starting with 0, up to $|S_R^L| - 1$, where S_R^L is the set of leaf states of L that can be reached from $I[L]$ in the leaf state space of L.

With the leaf state IDs, we can efficiently store the pricing function of each leaf $L \in \mathcal{L}$ of a decoupled state $s^{\mathcal{F}}$ as an array A of numbers, where $A[i]$ contains the price of the leaf state with ID i. To get a canonical representation, and to keep the memory footprint of the pricing function as small as possible, we decide to limit the size of the array to just fit the price of the highest ID of the leaf state that has finite cost. Implicitly, all leaf states with a higher ID are not reached in $s^{\mathcal{F}}$ and have cost ∞. This does incur a memory overhead, in the extreme case wasting $|S_R^L| - 1$ entries in the array, if only the leaf state with ID $|S_R^L| - 1$ is reached, so the entries for all other leaf states are ∞. However, leaf state spaces are mostly "well-behaved" in that such pathologic behaviour does not usually occur. In satisficing planning, where, as previously noted, we do not require the actual leaf state prices, but only reachability information, we only keep a bitvector

A for each leaf $L \in \mathcal{L}$, indicating whether the leaf state with ID i is reached if $A[i] = \top$.

The unique IDs and the maintenance of the pricing function as standard arrays allow the use of hashing functions, where two decoupled states are equal if the hashes of their center state, and for each leaf factor, the hashes of the representation of the pricing functions match.

Next, we show an optimization that, for satisficing planning in some cases does not even require hashing.

Invertibility of Leaf State Spaces Given the precomputed leaf state spaces, it is straightforward to compute the connectivity of these graphs. In particular, we can check if such a graph is strongly connected, when only considering transitions of leaf actions that do not affect, nor are preconditioned by the center factor. This check is linear in the size of the graph using Tarjan's algorithm (Tarjan 1972). Formally, we define the set of *no-center* actions of a leaf L as $\mathcal{A}^L_{\neg C} := \{a^L \in \mathcal{A}^L \mid \mathsf{pre}(a) \cap C = \emptyset \wedge \mathsf{eff}(a) \cap C = \emptyset\}$. If the part of the leaf state space of a leaf L that is reachable from $I[L]$ using all actions $\mathcal{A}$ is strongly connected when only considering transitions that are labeled with $\mathcal{A}^L_{\neg C}$-actions, then in every decoupled state $s^{\mathcal{F}}$ that is reachable from $s^{\mathcal{F}}_0$, *all leaf states of* L that can possibly be reached from $I[L]$ are reached in $s^{\mathcal{F}}$. We call such leaf factors *leaf-invertible*.

All decoupled states reached during search can only differ in the leaf-state prices for leaf-invertible factors, but will always have the same set of leaf states reached. Therefore, no hashing is required for such leaves for satisficing planning, we can simply skip these leaves in the dominance check. For optimal planning, however, we still need to compare the prices, since these might differ, so hashing is still required.

Another minor optimization that can be performed with the leaf-invertibility information is successor generation during search. More specifically, when checking which center actions are applicable in a decoupled state, we usually need to check preconditions on leaf factors by looking for a reached leaf state that enables an action. For leaf-invertible leaf factors, however, this check is no longer needed (even for optimal planning), because the set of reached leaf states remains constant. We precompute the set of applicable center actions given the set of reachable leaf states prior to the search, and then skip the check for leaf preconditions on leaf-invertible factors.

Both optimizations based on invertibility of leaf state spaces only serve to make the computation more efficient, but do not affect the search behavior.

Incorporating the g-value in Dominance Checking

Our second contribution is for optimal planning only, where a decoupled state $s^{\mathcal{F}}$ can only be pruned if there exists an already visited state $t^{\mathcal{F}}$ with lower g-value $g(t^{\mathcal{F}}) \leq g(s^{\mathcal{F}})$, larger hypercube $[s^{\mathcal{F}}] \subseteq [t^{\mathcal{F}}]$, and lower prices $\mathsf{prices}(t^{\mathcal{F}})[s^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$ for all leaf states $s^L \in S^{\mathcal{L}}$. We define this to be our *base dominance relation* $\preceq_B$.

The key observation is that the g-value difference of two decoupled states can be traded against differences in the pricing function. To see this, recall the definition of the cost

$s^{\mathcal{F}} : g(s^{\mathcal{F}}) = 10$	$\not\preceq_B$	$t^{\mathcal{F}} : g(t^{\mathcal{F}}) = 5$
$s^L_1 \to 1 \quad s^{L'}_1 \to x$		$s^L_1 \to 6 \quad s^{L'}_1 \to x$
$s^L_2 \to 1 \quad s^{L'}_2 \to x$	$\preceq_G$	$s^L_2 \to 6 \quad s^{L'}_2 \to x$

Figure 2: Illustrating example where $s^{\mathcal{F}}$ can only be pruned when using the new dominance criterion $\preceq_G$.

of a member state s of a decoupled state $s^{\mathcal{F}}$:

$$\begin{aligned}\mathsf{cost}(s^{\mathcal{F}}, s) &= g(s^{\mathcal{F}}) + \mathsf{price}(s^{\mathcal{F}}, s)\\ &= g(s^{\mathcal{F}}) + \sum_{L \in \mathcal{L}} \mathsf{prices}(s^{\mathcal{F}})[s[L]]\end{aligned}$$

A new decoupled state $s^{\mathcal{F}}$ can be pruned if there exists an already visited state $t^{\mathcal{F}}$ such that $[s^{\mathcal{F}}] \subseteq [t^{\mathcal{F}}]$ and for all $s \in [s^{\mathcal{F}}] : \mathsf{cost}(s^{\mathcal{F}}, s) \geq \mathsf{cost}(t^{\mathcal{F}}, s)$. The question is how to compute such a dominance relation without enumerating the costs of all member states.

Consider the example in Figure 2. Each box represents a decoupled state, e. g. $s^{\mathcal{F}}$, and an arrow $s^L_i \to 3$ indicates that in $s^{\mathcal{F}}$ we have $\mathsf{prices}(s^{\mathcal{F}})[s^L_i] = 3$. We have $g(s^{\mathcal{F}}) = 10$ and $g(t^{\mathcal{F}}) = 5$, and the prices in all but one leaf factor L of both states are identical. In leaf L, we have $\mathsf{prices}(s^{\mathcal{F}})[s^L] = \mathsf{prices}(t^{\mathcal{F}})[s^L] - 5$, so all leaf states of L in $s^{\mathcal{F}}$ can be reached cheaper by a cost of 5 in $s^{\mathcal{F}}$, but $s^{\mathcal{F}}$ has a g-value that is by 5 higher than that of $t^{\mathcal{F}}$. Then, with the existing dominance $\preceq_B$ from prior work, $s^{\mathcal{F}}$ cannot be pruned, because its prices are lower than the ones of $t^{\mathcal{F}}$. However, the cost of all its member states is equal to the cost of the states in $t^{\mathcal{F}}$, so $s^{\mathcal{F}}$ could be pruned.

Based on this observation, we define our new dominance relation $\preceq_G$ that includes the g-value in the comparison as follows:

$$\begin{aligned}&s^{\mathcal{F}} \preceq_G t^{\mathcal{F}} \Leftrightarrow [s^{\mathcal{F}}] \subseteq [t^{\mathcal{F}}] \text{ and}\\ &\quad g(s^{\mathcal{F}}) - g(t^{\mathcal{F}})\\ &\geq \sum_{L \in \mathcal{L}} max_{s^L \in S^L}(\mathsf{prices}(t^{\mathcal{F}})[s^L] - \mathsf{prices}(s^{\mathcal{F}})[s^L])\\ &\quad \forall s^L \in S^L \text{ where } \mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty\end{aligned}$$

If $s^{\mathcal{F}}$ has a g-value that is higher than the one of $t^{\mathcal{F}}$, but has leaf states that have a lower price, then the disadvantage in g-value can be traded against the advantage in leaf state prices. More concretely, it suffices to sum-up only the maximal price-difference of any leaf state over the leaves. By doing so, we basically compare only the member state $s \in [s^{\mathcal{F}}]$ for which the price-advantage of $s^{\mathcal{F}}$ is maximal. This can be done component-wise, so is efficient to compute. If $\mathsf{prices}(t^{\mathcal{F}}, s) - \mathsf{prices}(s^{\mathcal{F}}, s) \leq g(s^{\mathcal{F}}) - g(t^{\mathcal{F}})$, then this also holds for all other $s' \in [s^{\mathcal{F}}]$. Indeed, $\preceq_G$ detects that $s^{\mathcal{F}}$ in the above example is dominated and can be pruned.

Things become more subtle if the prices differ in more leaf factors, see for example Figure 3. We then need to *distribute* the difference in g-values *across* the leaf factors, i. e., we cannot use the full difference for each factor. However, we can even trade lower prices in one leaf by higher

$s^{\mathcal{F}} : g(s^{\mathcal{F}}) = 10$	$\not\preceq_B$	$t^{\mathcal{F}} : g(t^{\mathcal{F}}) = 5$
$s_1^L \to 1 \quad s_1^{L'} \to 3$		$s_1^L \to 6 \quad s_1^{L'} \to 1$
$s_2^L \to 1 \quad s_2^{L'} \to 3$	$\preceq_G$	$s_2^L \to 6 \quad s_2^{L'} \to 1$

Figure 3: Illustrating example where $s^{\mathcal{F}}$ has lower prices in one leaf factor (L), higher prices in another leaf (L') and we can still prove that $s^{\mathcal{F}}$ is dominated by $t^{\mathcal{F}}$.

$s^{\mathcal{F}} : g(s^{\mathcal{F}}) = 5$	$\to$	$t^{\mathcal{F}} : g(t^{\mathcal{F}}) = 8$
$s_1^L \to 3 \quad s_1^{L'} \to 2$		$s_1^L \to 2 \quad s_1^{L'} \to 0$
$s_2^L \to 1 \quad s_2^{L'} \to 3$		$s_2^L \to 0 \quad s_2^{L'} \to 1$

Figure 4: Illustrating example showing the a decoupled state $s^{\mathcal{F}}$ on the left and its price-moved-canonical representative on the right.

prices in another, incorporating these different prices in the g-difference. Say, extending the above example, there is a leaf factor L' where $\mathsf{prices}(s^{\mathcal{F}})[s^{L'}] = \mathsf{prices}(t^{\mathcal{F}})[s^{L'}] + 2$ for all $s^{L'} \in S^{L'}$. Then, we can add-up the price disadvantage of $+2$ in L' for $s^{\mathcal{F}}$ with its g-disadvantage $+5$ to make up for a price advantage of 7 in total for $s^{\mathcal{F}}$ in other leaves, where $t^{\mathcal{F}}$ might have higher prices:

$$\begin{aligned}
\mathsf{cost}(s^{\mathcal{F}}, s) &= g(s^{\mathcal{F}}) + \mathsf{price}(s^{\mathcal{F}}, s) \\
&= g(s^{\mathcal{F}}) + \sum_{L \in \mathcal{L}} \mathsf{prices}(s^{\mathcal{F}})[s[L]] \\
&= g(t^{\mathcal{F}}) + 5 + \mathsf{prices}(s^{\mathcal{F}})[s[L]] \\
&+ \mathsf{prices}(s^{\mathcal{F}})[s[L']] \\
&+ \sum_{L_i \in \mathcal{L} \setminus \{L, L'\}} \mathsf{prices}(s^{\mathcal{F}})[s[L_i]] \\
&= g(t^{\mathcal{F}}) + 5 + \mathsf{prices}(t^{\mathcal{F}})[s[L]] - 5 \\
&+ \mathsf{prices}(t^{\mathcal{F}})[s[L']] + 2 \\
&+ \sum_{L_i \in \mathcal{L} \setminus \{L, L'\}} \mathsf{prices}(t^{\mathcal{F}})[s[L_i]] \\
&= g(t^{\mathcal{F}}) + \sum_{L \in \mathcal{L}} \mathsf{prices}(t^{\mathcal{F}})[s[L]] + 2 \\
&= g(t^{\mathcal{F}}) + \mathsf{price}(t^{\mathcal{F}}, s) + 2 \\
&= \mathsf{cost}(t^{\mathcal{F}}, s) + 2
\end{aligned}$$

In the actual formulation of the definition of $\preceq_G$, we have:

$$\begin{aligned}
&[s^{\mathcal{F}}] = [t^{\mathcal{F}}] \text{ and} \\
&\quad g(s^{\mathcal{F}}) - g(t^{\mathcal{F}}) = 5 \\
&\geq \sum_{L \in \mathcal{L}} max_{s^L \in S^L} (\mathsf{prices}(t^{\mathcal{F}})[s^L] - \mathsf{prices}(s^{\mathcal{F}})[s^L]) \\
&= (6 - 1) + (1 - 3) = 2
\end{aligned}$$

In this case, given that $t^{\mathcal{F}}$ is visited before $s^{\mathcal{F}}$ during search, we can prune $s^{\mathcal{F}}$, although the prices of its leaf states are neither lower-equal, nor higher-equal than the prices of $t^{\mathcal{F}}$. There is even a difference of cost 2 left that could be used to trade higher prices of $t^{\mathcal{F}}$ in another leaf factor.

Finally, we introduce another sort of canonical form, which moves as much of the leaf-state prices into the g-value of a decoupled state as possible. It is easy to see that, if in a decoupled state $s^{\mathcal{F}}$ there exists a leaf L such that all leaf states s^L have a minimum non-zero price p, so $\forall s^L \in S^L : \mathsf{prices}(s^{\mathcal{F}})[s^L] \geq p$, then we can reduce the prices of all these leaf states by p and increase $g(s^{\mathcal{F}})$ by p without affecting the cost $\mathsf{cost}(s^{\mathcal{F}}, s)$ of the member states $s \in [s^{\mathcal{F}}]$. Intuitively, the transformation moves the price that has *at least* to be spent to reach any member state of $s^{\mathcal{F}}$ into its g-value, reducing the price of all leaf states accordingly, so that in every leaf L there exists at least one leaf state with price 0. See Figure 4 for an example of a decoupled state $s^{\mathcal{F}}$ and its canonical representative $t^{\mathcal{F}}$.

The main advantage of moving the leaf state prices into the g-value of the decoupled state occurs when executing decoupled search using the A* algorithm, which on a cost layer (f) gives priority to states with lower heuristic value. By moving the cost into the g-value we achieve that the heuristic of a decoupled state (which works on the leaf-state prices (Gnad and Hoffmann 2018)) can only get lower, thus helping A* to focus the search to more promising states. A second important effect is that the part of the prices moved into the g-value will always be considered entirely by the search, whereas heuristics (in the extreme case blind search) might not be able to capture all the cost incurred in the pricing function. We remark that moving the minimum leaf prices into the g-value is independent of the new dominance criterion $\preceq_G$. It can have a positive impact on the number of state expansions of $\preceq_G$, the base dominance check $\preceq_B$, and exact duplicate checking.

Experimental Evaluation

We implemented our invertibility optimizations, exact duplicate checking, as well as the g-value-aware dominance check in the decoupled search planner by Gnad & Hoffmann (2018), which itself builds on the Fast Downward planning system (FD) (Helmert 2006). We conducted our experiments using the lab python package (Seipp et al. 2017) on all benchmark domains of the International Planning Competition (IPC) from 1998-2018 in both the optimal and satisficing tracks. For optimal planning, we run blind search and A* with $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009); in satisficing planning, we use greedy best-first search (GBFS) with the h^{FF} heuristic, both with and without preferred operator pruning (Hoffmann and Nebel 2001). We use the common runtime and memory limits of 30 minutes and 4GiB.[1]

Decoupled search needs a method that provides a factoring, i. e., that detects a star topology in the causal structure of the input planning task. We use two basic factoring methods, inverted-fork factorings (IF) – only for satisficing planning – as well as the incident arcs factoring (IA) as described in Gnad et al. (2017). We expect inverted-fork factorings to

[1] If the paper gets accepted, we will make all experimental data publicly available.

		Blind Search						A* with $h^{\text{LM-cut}}$					
Domain	#	base	$\preceq_G$	pInG	$\preceq_G$,pInG	Dup	Dup,pInG	base	$\preceq_G$	pInG	$\preceq_G$,pInG	Dup	Dup,pInG
DataNetwork	20	**9**	**9**	5	**9**	5	5	**14**	**14**	12	**14**	12	12
Depots	22	3	**4**	**4**	**4**	2	**4**	**7**	**7**	**7**	**7**	5	**7**
Driverlog	20	**11**	**11**	**11**	**11**	9	10	13	13	13	13	13	13
Elevators08	30	6	6	9	**16**	0	10	11	11	22	**23**	0	22
Elevators11	20	4	4	7	**13**	0	8	9	9	**18**	**18**	0	**18**
Floortile11	20	**2**	**2**	**2**	**2**	0	0	**6**	**6**	**6**	**6**	3	4
Floortile14	20	0	0	0	0	0	0	**4**	**4**	**4**	**4**	2	2
Freecell	42	0	0	0	0	0	**2**	1	**2**	**2**	**2**	1	**2**
GED	20	13	**15**	**15**	**15**	7	**15**	**15**	**15**	**15**	**15**	13	**15**
Grid	5	1	1	1	1	1	1	**2**	**2**	**2**	**2**	1	**2**
Logistics00	28	22	22	22	22	22	22	**28**	**28**	**28**	**28**	27	27
Logistics98	35	2	**3**	2	**3**	0	2	6	6	7	**8**	3	7
Miconic	145	**46**	**46**	**46**	**46**	42	42	135	135	135	135	135	135
NoMystery	20	**19**	**19**	**19**	**19**	16	16	**20**	**20**	**20**	**20**	19	19
Openstacks14	20	**2**	1	**2**	**2**	**2**	**2**	2	2	2	2	2	2
PSR	48	**48**	**48**	46	**48**	42	46	**48**	**48**	47	**48**	45	47
Rovers	40	**7**	**7**	**7**	**7**	6	**7**	8	8	8	8	8	8
Satellite	36	5	5	5	5	5	5	7	7	**9**	**9**	7	**9**
Transport08	30	8	8	**9**	**9**	0	**9**	**9**	**9**	**9**	**9**	5	**9**
Transport11	20	3	3	**4**	**4**	0	**4**	**4**	**4**	**4**	**4**	0	**4**
Transport14	20	2	3	4	**6**	0	4	3	3	**5**	**5**	1	**5**
Woodworking08	28	7	7	7	7	7	7	16	**17**	16	**17**	16	16
Woodworking11	18	2	2	2	2	2	2	10	**11**	10	**11**	10	10
Zenotravel	20	8	8	8	**9**	6	7	12	12	12	**13**	8	12
Other	299	92	92	92	92	92	92	122	122	122	122	122	122
$\sum$	1026	322	326	329	**352**	266	322	512	515	535	**543**	458	529

Table 1: Coverage data for optimal search with blind search (left) and with A* using $h^{\text{LM-cut}}$. All configurations use the incident arcs factoring. Domains without difference in coverage are summarized in "Other". Best coverage is highlighted in **bold face**.

nicely show the advantage of the more efficient handling of invertible leaf state spaces, since there are several domains that have such state spaces in this case, but not using IA. IA is the canonical choice since it is fast to compute and finds good decompositions in many domains.

We show experimental data for the following configurations: as a baseline (**Base**), we use the standard dominance pruning $\preceq_B$ of prior work. In optimal planning, we compare the baseline to (1) our new dominance checking $\preceq_G$ that incorporates the g-values in the check, (2) a variant of decoupled search that moves the minimum leaf prices into the g-value of states (**pInG**), and (3) a combination of $\preceq_G$ an pInG. For satisficing planning, we compare Base to a configuration that enables the invertible-leaves optimizations (**INV**). For both search types, we also show data with exact duplicate checking as described above (**DUP**). All configurations for optimal planning also include the successor generation optimization, and there are two DUP variants, with and without pInG. For satisficing planning, DUP includes the INV optimizations.

Tables 1 and 2 show coverage data (number of instances solved) across the IPC benchmarks where the factoring methods are able to detect a star factoring. For optimal planning, Table 1, we see that both $\preceq_G$ and pInG lead to an increase in coverage across several domains for both blind search and A* with $h^{\text{LM-cut}}$. There are only a few domains, 1 for $\preceq_G$ and 2 for pInG when using blind search, and 0 respectively 2 when using $h^{\text{LM-cut}}$, where the new methods lose in coverage. The combination of $\preceq_G$ and pInG leads to an increase in several domains, most noticeably in Elevators and Transport, and never has a coverage lower than Base. In total, the combination of $\preceq_G$ and pInG gains 30 instances compared to Base with blind search and 31 instances for A* with $h^{\text{LM-cut}}$.

The plain duplicate checking configurations without pInG show a significant drop in coverage compared to Base, so although the checking is computationally more efficient, this does not even out the weaker pruning power. When enabling pInG, which is orthogonal to the use of duplicate or dominance pruning, gets the total coverage back to the level of Base for blind search, although the per-domain numbers differ, and increases it by 17 instances for $h^{\text{LM-cut}}$. Compared to the pInG baseline, though, in both search variants the exact duplicate checking drops in coverage. Surprisingly, with blind search the duplicate checking variant can solve two Freecell instances that the dominance pruning cannot solve. These are instances where the size of the search space does not increase (by a lot), and where the more efficient hashing pays off.

The invertibility optimization in the successor generation only leads to a mild improvement in runtime, which does not affect coverage. Therefore, we enable it in all reported opti-

Domain	#	no preferred Base	INV	DUP	preferred Base	INV	DUP
DataNetwork	20	6	6	6	**7**	**7**	5
Floortile11	20	**3**	**3**	1	**3**	2	1
Floortile14	20	**2**	**2**	1	**2**	**2**	1
NoMystery	20	**19**	**19**	16	**19**	**19**	16
Rovers	40	**22**	**22**	20	40	40	40
Tetris	17	4	4	4	**12**	11	11
TPP	29	24	24	24	**26**	**26**	25
Trucks	27	13	13	13	13	13	**14**
Other	844	637	637	637	686	686	686
$\sum$	1037	**730**	**730**	722	**808**	806	799

Table 2: Coverage for satisficing planning with GBFS with h^{FF} and IA factoring, without (left) and with the use preferred operator pruning. Domains without difference in coverage are summarized in "Other".

mal planning configurations and don't report detailed data.

In satisficing planning (Table 2), the invertibilty optimization does not affect coverage except in two instances that the baseline solves just within the timeout. Duplicate checking loses a few instances in coverage in Floortile, NoMystery, Rovers, and TPP. In Trucks, though, it is able to solve one more instance and average runtime drops by a factor of 1.9.

Comparing the data in Table 2 that shows the IA factoring to configurations with inverted-fork factorings (not shown here), reveals that for the other factoring there is no change in coverage for any of the new methods. We conjecture that this is the case because with inverted-fork factorings, the leaf state spaces are usually smaller and often invertible. Thus, there is no (significant) loss in pruning power of exact duplicate checking. On the other hand, this indicates that neither the invertibility optimization, nor the duplicate checking, have a great impact on the performance even in this good scenario. This might be because, as can be seen in Figure 1, less time is spent on dominance pruning for satisficing search in general, so the improvement would need to be significant to translate into a gain in coverage.

The scatter plots in Figure 5 and 6 shed further light on the runtime and search space size comparison between some optimal planning configurations. Figure 5 shows the number of expanded states in the top row and the runtime in the bottom row. All configurations use the incident arcs factoring, with blind search in the left column and $h^{\text{LM-cut}}$ in the right column. All plots compare the baseline to the combination of $\preceq_G$ and pInG. Quite obviously, the effect of the more clever dominance check and moving the minimum price into the g-value leads to a significant advantage of up to several orders of magnitude for both search types.

Figure 6 illustrates the effect of exact duplicate checking. The left plot shows the increase in the number of expanded states, which is expected from the reduced pruning power. The left plot indicates that where the increase in search space size is small the more efficient computation via hashing indeed pays off runtime-wise. The two dashed lines are highlighting a difference of a factor of 2, which can often be gained with duplicate checking.

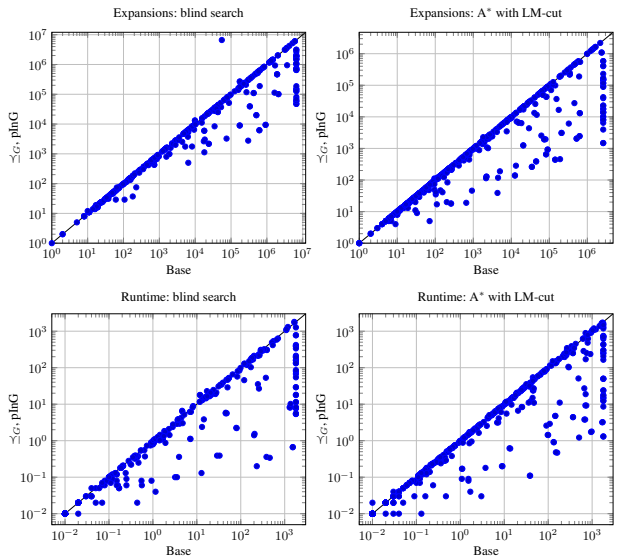

Figure 5: The plots on the left show blind search, on the right A* with $h^{\text{LM-cut}}$. The top row shows the comparison of the number of expanded states, the bottom row total time as reported by FD.

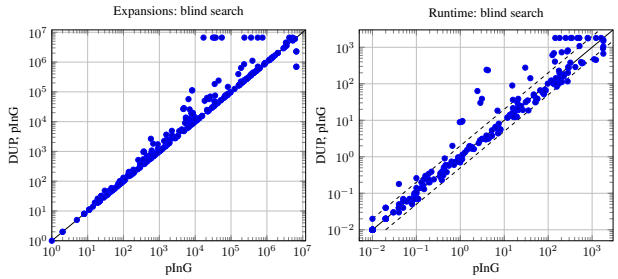

Figure 6: Comparison of pruning power and runtime of dominance and exact duplicate pruning with blind search, both using pInG with IA factoring.

For satisficing planning, Figure 7 and 8 have scatter plots similar to the setting in Figure 6, showing the difference in search space size (number of evaluated nodes) and runtime for GBFS with h^{FF} and IA factoring. Again, the number of evaluated nodes increases in many cases, where it leads to an increase in runtime. In cases where the search space is (mostly) unaffected, the advantage in runtime is less pronounced, compared to the optimal planning setting.

Finally, Figure 8 shows the advantage in runtime of INV compared to Base for GBFS with h^{FF} with IA (left) and IF factoring. Since there is no difference in terms of search space size, we only see the effect of the invertible-leaf-state-space optimization for the successor generation and dominance check. With inverted fork factoring, as indicated before, we see a much higher speed-up with the optimizations enabled. This is due to the fact that, with inverted fork factorings, a lot more leaf state spaces are leaf-invertible, so the optimization triggers more often.

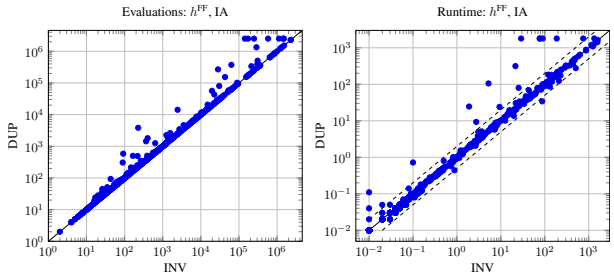

Figure 7: Satisficing search with GBFS and h^{FF}, comparing the number of state evaluations (left) and FD's "total time" of decoupled search with dominance (Base) and exact duplicate checking (DUP) with IA factoring.

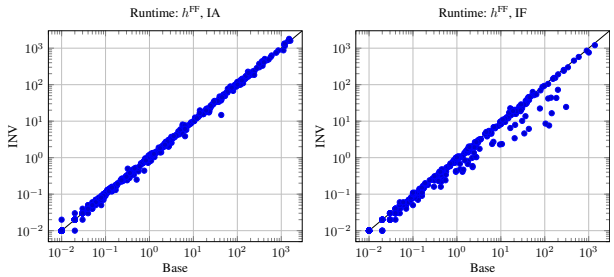

Figure 8: Same setting as Figure 7, comparing only the runtimes of Base and INV using incident-arcs (left) and inverted-fork factoring.

Conclusion

In this work, we have taken a closer look at the behavior and implementation details of dominance pruning in decoupled search. We introduced exact duplicate checking, which, although inferior to dominance pruning in terms of pruning power, can improve search performance in practice due to higher computational efficiency. Under certain circumstances, exact duplicate checking can indeed pay off. Overall, however, the reduced pruning leads to worse empirical results in both optimal and satisficing planning. We leave it to future work to identify required conditions where duplicate checking can be enabled without risk.

We have shown two implementational optimizations based on invertible leaf state spaces that make the computation of the successor generator and the dominance check more efficient. In satisficing planning, where both optimizations apply, this had only had a minor effect on the runtime, though. We conjecture that this is because the ratio of the overall runtime spent on dominance checking is lower than in optimal planning. There, since only the first optimization is applicable, there is no significant gain, either.

More interestingly, we developed two extensions of dominance pruning in optimal planning, one that takes into account the g-value of decoupled states, another that moves minimum leaf state prices into the g-value. This is an interesting new insight in the behavior of dominance pruning in decoupled search and leads to a better understanding. On the practical side, both methods are beneficial and their combination significantly outperforms the baseline, improving the performance of decoupled search across many benchmark domains. The two extensions come at no negative effect on the runtime, so it is always advisable to use them.

For future work, we think it is worthwhile to further investigate dominance pruning methods, given that in some benchmarks, checking dominance takes a large part in the overall search time of the planner. We think that a combination with the qualitative dominance pruning in explicit state search (Torralba 2017), where two states can be compared according to a minimal guaranteed difference in their goal distance, can be interesting and beneficial. Computing this difference in a projection onto the center factor could further increase the pruning potential between decoupled states, similar to the incorporation of g-values.

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In Raedt, L. D., ed., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 891–892. Montpellier, France: IOS Press.

Amir, E., and Engelhardt, B. 2003. Factored planning. In Gottlob, G., ed., *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 929–935. Acapulco, Mexico: Morgan Kaufmann.

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

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In Biundo, S., and Fox, M., eds., *Proceedings of the 5th European Conference on Planning (ECP'99)*, 60–72. Springer-Verlag.

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

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 28–35. AAAI Press.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S., and Fox, M., eds., *Proceedings of the 5th European Conference on Planning (ECP'99)*, 135–147. Springer-Verlag.

Edelkamp, S.; Leue, S.; and Lluch-Lafuente, A. 2004. Partial-order reduction and trail improvement in directed model checking. *International Journal on Software Tools for Technology Transfer* 6(4):277–301.

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

Gnad, D., and Hoffmann, J. 2018. Star-topology decoupled state space search. *Artificial Intelligence* 257:24 – 60.

Gnad, D., and Hoffmann, J. 2019. On the relation between star-topology decoupling and petri net unfolding. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*. AAAI Press.

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

Gnad, D.; Hoffmann, J.; and Wehrle, M. 2019. Strong stubborn set pruning for star-topology decoupled state space search. *Journal of Artificial Intelligence Research* 65:343–392.

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

Godefroid, P., and Wolper, P. 1991. 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)*, 332–342.

Hall, D.; Cohen, A.; Burkett, D.; and Klein, D. 2013. Faster optimal planning with partial-order pruning. In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Rome, Italy: AAAI Press.

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

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

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

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*. San Francisco, CA, USA: AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Starke, P. 1991. Reachability analysis of petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in Systems Analysis* 8(4/5):293–304.

Tarjan, R. E. 1972. Depth first search and linear graph algorithms. *SIAM Journal on Computing* 1(2):146–160.

Torralba, Á., and Hoffmann, J. 2015. Simulation-based admissible dominance pruning. In Yang, Q., ed., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 1689–1695. AAAI Press/IJCAI.

Torralba, Á.; Gnad, D.; Dubbert, P.; and Hoffmann, J. 2016. On state-dominance criteria in fork-decoupled search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press/IJCAI.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence* 242:52–79.

Torralba, Á. 2017. From qualitative to quantitative dominance pruning for optimal planning. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, 4426–4432. AAAI Press/IJCAI.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, 491–515.

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

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