Revisiting Dominance Pruning in Decoupled Search

Daniel Gnad
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
gnad@cs.uni-saarland.de

Abstract
In classical planning as search, duplicate state pruning is a standard method to avoid unnecessarily handling the same state multiple times. In decoupled search, similar to symbolic search approaches, search nodes, called decoupled states, do not correspond to individual states, but to sets of states. Therefore, duplicate state pruning is less effective in decoupled search, and dominance pruning is employed, taking into account the state sets. We observe that the time required for dominance checking dominates the overall runtime, and propose two ways to tackle this issue. Our main contribution is a stronger variant of dominance checking for optimal planning. The new variant greatly improves the latter, without incurring a computational overhead. Moreover, we develop three methods that make the dominance check more efficient: exact duplicate checking, which, albeit resulting in weaker pruning, can pay off due to the use of hashing; avoiding the dominance check in non-optimal planning if leaf state spaces are invertible; and exploiting the transitivity of the dominance relation to only check against the relevant subset of visited decoupled states. We show empirically that all our improvements are indeed beneficial in many standard benchmarks.

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 is exponential in the size of its description. Many 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 Rosenstock 2011; Domshlak, Katz, and Shleymfan 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 Engerlhardt 2003; Braffman 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 $L = \{L_1, \ldots, L_n\}$, while any interaction between leaves has to involve the center, too. A decoupled state $s^D$ corresponds to a single center state, an assignment to $C$, and a non-empty set of leaf states (assignments to an $L_i$) for each $L_i$. The member states of $s^D$, i.e., the set of explicit states it represents, 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 the reduction achieved by decoupled search can be exponentially larger than it is for other state-space-reduction methods like 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).

Since a decoupled state corresponds to a set of states, namely its member states, 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 only considered dominance pruning (Torralba et al. 2016; Gnad and Hoffmann 2018), where a decoupled state $s^D_1$ with member states $S_1$ is dominated by a decoupled state $s^D_2$ that represents the set of states $S_2$ if $S_1 \subseteq S_2$. In optimal planning the pricing function has to be checked, too. Only if each member state of $s^D_2$ is reachable in $s^D_1$ with at most the price it has in $s^D_2$, we can safely prune $s^D_1$, like duplicate states in explicit state search.

Initiating this work was the observation that in optimal planning on average around 60% of the overall runtime of decoupled search is spent on dominance checking (on instances from our evaluation solved in $\geq 0.1s$). Thus, we take a closer look at (1) algorithmic improvements that lead to an increased pruning power for optimal planning, and (2) ways to make the dominance check more efficient in general. Regarding (1), we introduce two new extensions to the domi-
nance check. First, we take into account not only the pricing function, but incorporate the $g$-value of $A^*$ in the check. Second, we propose a decoupled-state transformation that moves cost from the pricing function into the $g$-value. Both make the dominance check more informed without introducing a computational overhead. For (2), we experiment with an implementation of exact duplicate checking, which, albeit resulting in weaker pruning, can be beneficial runtime-wise due to the use of hashing; we identify cases for non-optimal planning where leaves can be skipped in the check, namely if their leaf state space is invertible; and, exploiting the transitivity of the dominance relation, we only check against the non-dominated subset of visited decoupled states.

In our experimental evaluation, we see that the improvements as of (2) indeed have a positive impact on the runtime. The stronger pruning variants from (1) lead to a substantial reduction in search effort and runtime.

**Background**

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

An action $a$ is applicable in state $s$ if $\text{pre}(a) \subseteq s$. Applying $a$ in a (partial) state $s$ changes the value of all $v \in \text{vars}(\text{eff}(a)) \cap \text{vars}(s)$ to $\text{eff}(a)[v]$, and leaves $s$ unchanged elsewhere. The outcome state is denoted $s[a]$. A plan for $Π$ is an action sequence $\pi$ applicable in $I$ that results in a state $s_G \supseteq G$. A plan $\pi$ is optimal if the sum of the cost of its actions, denoted $\text{cost}(\pi)$, 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 $F$, 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 $C \in F$ that interacts arbitrarily with the other factors $L \in L := F \setminus \{C\}$, 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 $A^C$, and those affecting a leaf are called leaf actions, denoted $A^L$. The actions that affect a particular leaf $L \in L$ are denoted $A^L$. A sequence of center actions applicable in $I$ in the projection onto $C$ is a center path, a sequence of leaf actions affecting $L$, applicable in $I$ in the projection onto $L$, is a leaf path. A complete assignment to $C$, respectively an $L \in L$, is called a center state, respectively leaf state. The set of all leaf states is denoted $S^L$, and that of a particular leaf $L$ is denoted $S^L_L$.

A decoupled state $s^F$ is a pair $(\text{center}(s^F), \text{prices}(s^F))$, where $\text{center}(s^F)$ is a center state, and $\text{prices}(s^F) : S^C \rightarrow \mathbb{R}^{\geq 0} \cup \{\infty\}$ 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 is compliant, i.e., that can be scheduled alongside the center path executed up to $s^F$. By $S^F$ we denote the set of all decoupled states. We say that a decoupled state $s^F$ satisfies a condition $p$, denoted $s^F \models p$, iff (i) $p[\text{center}(s^F)] \subseteq \text{center}(s^F)$ and (ii) for every $L \in L$ there exists an $s^L \in S^L$ s.t. $p[L] \subseteq s^L$ and $\text{prices}(s^F)[s^L] < \infty$. We define the set of leaf actions enabled by a center state $s^C$ as $A^L_{\text{center}(s^C)} := \{a^L \mid a^L \in A_C \land \text{pre}(a^C)[C] \subseteq s^C\}$.

The initial decoupled state $s^F_0$ is defined as $s^F_0 := (\text{center}(s^F_0), \text{prices}(s^F_0))$, where $\text{center}(s^F_0) = I[C]$. Its pricing function is given, for each $L \in L$, as $\text{prices}(s^F_0)[s^L] = 0$, where $s^L = I[L]$; and elsewhere as $\text{prices}(s^F_0)[s^L] = c_{s^C}(s^F_0, s^L)$, where $c_{s^C}(s^F_0, s^L)$ is the cost of a cheapest path of $A^L_{\text{center}(s^F)} \setminus A^C$ actions from $s^L_0$ to $s^L$. If no such path exists $c_{s^C}(s^F_0, s^L) = \infty$. The set of decoupled goal states is $S^F_G := \{s^F_G \mid \text{center}(s^F_G) \models G\}$.

Decoupled-state transitions are induced only by center actions, where a center action $a^C$ is applicable in a decoupled state $s^F$ if $s^F \models \text{pre}(a^C)$. By $S^F_{a^C}(s^F)$ we define the set of leaf states of $L$ in $s^F$ that comply with the preleaf condition of $a^C$, i.e., $S^F_{a^C}(s^F) := \{s^L \mid \text{pre}(a^C)[L] \subseteq s^L \land \text{prices}(s^F)[s^L] < \infty\}$. Applying $a^C$ to $s^F$ results in the decoupled state $s^F := s^F[a^C]$, as follows: $\text{center}(s^F) = \text{center}(s^F)[a^C]$, and $\text{prices}(s^F)[s^L] = \min_{s^L \in S^F_{a^C}(s^F)}(\text{prices}(s^F)[s^L] + c_{s^C}(a^L, s^L))$, where $s^L[\text{center}(a^C)] = s^L[a^C]$.

By $\pi^C(s^F)$ we denote the center path that starts in $s^F_0$ and ends in $s^F$. Accordingly, we define the $g$-value of $s^F$ as $g(s^F) := \text{cost}(\pi^C(s^F))$, the cost of its center path.

A decoupled state $s^F$ represents a set of explicit states, which takes the form of a hypercube whose dimensions are the leaf factors $L$. Hypercubes are defined as follows:

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

---

1 An action can be center and leaf action, then $A^C \cap A^F \neq \emptyset$. 11810
Dominance Pruning for Decoupled Search

Prior work on decoupled search has only considered dominance pruning instead of exact duplicate checking (Toralba et al. 2016; Gnäd and Hoffmann 2018). With dominance pruning, instead of duplicate states, the search prunes decoupled states that are dominated by an already visited decoupled state (with lower g-value). We formally define the dominance relation $\preceq_D \subseteq S^F \times S^F$ over decoupled states as $(t^F, s^F) \preceq_D (t^{F'}, s^{F'})$ if (1) $s^F = t^{F'}$ and (2) for all $s^L \in S^L : \text{prices}(s^{F'})[s^L] \leq \text{prices}(t^{F'})[s^L]$. Instead of $(t^F, s^F) \preceq_D (t^{F'}, s^{F'})$, we often write $t^F \preceq_D s^F$ to denote that $s^F$ dominates $t^F$. Note that (2) is only required for optimal planning. In satisficing planning, 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 hashing the states and then comparing them component-wise. If a state is re-visited during search – and, in case of hashing, followed by a component-wise comparison of the leaf states – the new state can only be pruned if there exists an already visited state $s' \preceq_D s$. Importantly, care must be taken when hashing decoupled states, to properly take into account both reachability and prices of leaf states. To do so 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 all action preconditions on $V \setminus L$ are reached. The resulting transition systems are called the leaf state spaces for every leaf $L \in L$. These are unique, able to identify each leaf state, where $S^L_R$ is the set of leaf states of $L$ that can be reached from $J[L]$ in the leaf state space of $L$. With the leaf state IDs, we can efficiently store the pricing function of a decoupled state $s^F$ for each leaf as an array $A$ of numbers. Then $A[i]$ is the price of the leaf state with ID $i$. To get a canonical representation of $s^F$, and to keep the memory footprint of its pricing function small, we decide to limit the size of the array to just fit the highest ID of a leaf state with finite price. Implicitly, all leaf states with a higher ID are not reached in $s^F$ and have cost $\infty$. This does incur a memory overhead, in the extreme case wasting $S^L_R - 1$ entries in the array, if only the leaf state with ID $S^L_R - 1$ is reached, so the entries for all other leaf states are $\infty$. However, leaf state spaces are mostly “well-behaved” in the sense that such pathologic behaviour does not usually occur. In non-optimal planning, where, as previously noted, we do not require the actual leaf state prices, but only reachability information, we keep a bitvector $A$ for each leaf. Here, $A[i] = \top$ indicates that the leaf state with ID $i$ is reached.

Storing the pricing function as standard arrays allows the use of hash functions, where two decoupled states can only be equal if the hashes of their center state, and for each leaf factor, the hashes of the representation of the pricing functions match. In this case, since our hashing is non-perfect, we still need to ensure that the states are indeed equal.

Improved Dominance for Optimal Planning

In this section, we introduce two improvements over the basic dominance relation $\preceq_F$ for optimal planning. The first one incorporates the $g$-value of decoupled states into the dominance check and compares the prices across leaves. This increases the potential for pruning, e.g. allowing to prune states that have a lower $g$-value. The second technique is a decoupled-state transformation that moves part of the leaf-state prices into the $g$-value of a decoupled state, enhancing search guidance by fully accounting for costs that have to be spent to reach the cheapest member state.

Incorporating All Costs in Dominance Checking

In optimal planning, a decoupled state $t^F$ can only be pruned with $\preceq_F$ if there exists an already visited state $s^F$ with lower $g$-value that dominates it. The dominance check considers $g$-values and pricing function separately. We next show that these can actually be combined, i.e., the $g$-value difference

\[
\begin{align*}
t^F : g(t^F) &= 10 \\
s^1_1 &\rightarrow x \\
s^2_1 &\rightarrow x \\
s^1_2 &\rightarrow x \\
s^2_2 &\rightarrow x \\
\end{align*}
\]

Figure 1: Two decoupled states $t^F, s^F$; their $g$-values, and pricing functions; $s^F$ can only be pruned with $\preceq_G$. The hypercube of $s^F$ captures both the reachability and the prices of all member states of $s^F$. For every member state $s$ of a decoupled state $s^F$, we can construct the global plan, i.e., the sequence of actions that starts in $s$ and ends in $s$ by augmenting $s^C(s^F)$ with cheapest-compliant leaf paths, i.e., leaf action sequences that lead to the pricing function of $s^F$. 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 $s^F$ from $I$ can be computed as follows: $\text{cost}(s^F, s) = g(s^F) + \text{price}(s^F, s)$. 

Exact Duplicate Checking

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. In this section, we will look into exact duplicate checking for decoupled search, showing how an efficient hashing can be implemented.

Formally, we define the duplicate state relation over decoupled states $\preceq_D \subseteq S^F \times S^F$ as the identity relation where $(t^F, s^F) \preceq_D (t^{F'}, s^{F'})$ if $s^F = t^{F'}$. Like in explicit state search, a decoupled state $t^F$ can safely be pruned if there exists an already visited state $s^F$ where $g(s^F) \leq g(t^F)$ and $t^F \preceq_D s^F$. We remind that a search node, i.e., a decoupled state $s^F$, does not represent a single state, but a set of states, namely its hypercube $s^F$. Consequently, duplicate checking is less effective, because the chances of finding a decoupled state with the exact same hypercube (including leaf state prices) are smaller than finding a duplicate in explicit state search. Importantly, care must be taken when hashing decoupled states, to properly take into account both reachability and prices of leaf states. To do so 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...
of two decoupled states can be traded against differences in the pricing function. To see this, recall the definition of the cost of a member state \( s \) of a decoupled state \( s^F \):

\[
\text{cost}(s^F, s) = g(s^F) + \text{price}(s^F, s) = g(s^F) + \sum_{L \in \mathcal{L}} \text{prices}(s^F)[s[L]]
\]

For a new decoupled state \( t^F \), instead of only comparing its pricing function to the ones of visited decoupled states with lower \( g \)-value, we can directly compare the costs of its member states to those of \( s^F \). All member states of \( t^F \) have lower cost in \( s^F \): \( \forall s \in [t^F] : \text{cost}(s^F, s) \leq \text{cost}(t^F, s) \). In this case, analogously to pruning duplicate states with higher \( g \)-value in explicit state search, we can safely prune \( t^F \).

Consider the example in Figure 1. Each box represents a decoupled state, and an arrow \( s^L \rightarrow 6 \) indicates e.g. that in \( s^F \) we have \( \text{price}(s^F)[s^L] = 6 \). Say \( s^F \) is visited and \( t^F \) is a new state, where \( g(t^F) = 10 \) and \( g(s^F) = 5 \). Further, the prices in leaf factor \( L \) of both states are identical. In leaf \( L \), we have \( \text{price}(s^F)[s^L] = \text{prices}(t^F)[s^L] + 5 \), so all leaf states of \( L \) in \( t^F \) are cheaper by a cost of 5, but \( s^F \) has a \( g \)-value that is by 5 lower than that of \( t^F \). With the dominance relation \( \preceq_B \) from prior work, \( t^F \) cannot be pruned, because its prices are lower than the ones of \( s^F \). However, the cost of all its member states is equal to the cost of the states in \( s^F \), so it is actually safe to prune \( t^F \).

An important question is how to compute this check efficiently, i.e., without explicitly enumerating the costs of all member states. We next show that, similar to \( \preceq_B \), dominance can be checked component-wise by only considering the leaf state with the highest price difference per leaf.

Formally, we define the all-costs dominance relation \( \preceq_G \subseteq S^F \times S^F \) as follows:

\[
(t^F, s^F) \preceq_G s^F \iff g(t^F) - g(s^F) \geq \sum_{L \in \mathcal{L}} \max_{s^L \in S^L_L} (\text{prices}(s^F)[s^L] - \text{prices}(t^F)[s^L]),
\]

where \( S^L_L = \{s^L \in S^L \mid \text{prices}(t^F)[s^L] < \infty \} \).

If \( t^F \) has a higher \( g \)-value than \( s^F \), but has leaf states with lower prices, 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. Thereby, we essentially compare only the member state \( s \in [t^F] \) for which the price-advantage is maximal. This can be done component-wise, so is efficient to compute. Indeed, \( \preceq_G \) detects that \( t^F \) in the above example is dominated and can be pruned.
into the g-value we achieve that the heuristic of a decoupled state (which takes into account the pricing function, cf. Gnad and Hoffmann 2018) can only get lower, aiding A* to focus on 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 represented in the pricing function.

Note that the g-value adaptation is independent of the new dominance relation \( \preceq_G \). 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 \( \preceq_D \).

**Efficient Implementation**

In this section, we propose two optimizations that make the dominance check more efficient. First, we show that with invertible leaf state spaces the comparison of leaf reachability can be entirely avoided in non-optimal planning. Second, we show how to exploit the transitivity of dominance relations to focus the check on the relevant subset of decoupled states. Both optimizations do not affect the pruning behavior.

### Invertible Leaf State Spaces

Given the precomputed leaf state spaces described before, it is straightforward to compute the connectivity of these graphs. In particular, we can efficiently check if a leaf state space is strongly connected when only considering transitions of leaf actions that do not affect, nor are preconditions by, the center factor. Formally, we define the set of no-center actions of a leaf \( \mathcal{A}^L_{CC} := \{ a^L \in \mathcal{A}^L | \text{vars}(\\text{pre}(a)) \cap C = \emptyset \wedge \text{vars}(\\text{eff}(a)) \cap C = \emptyset \} \).

Let \( S^L_R \) be the set of L-states that is reachable from \( I[L] \) in the projection onto \( L \) using all actions \( \mathcal{A} \). Let further \( S^L_R \setminus \mathcal{A}^L_{CC} \) be the corresponding set using only the no-center actions \( \mathcal{A}^L_{CC} \) of \( L \). We say that \( L \) is leaf-invertible, if \( S^L_R = S^L_R \setminus \mathcal{A}^L_{CC} \), i.e., any L-state reachable from \( I[L] \) can be reached using no-center actions, and the part of the leaf state space induced by \( S^L_R \) and \( \mathcal{A}^L_{CC} \) is strongly connected.

**Proposition 1** Let \( L \) be leaf-invertible and \( S^L_R \) the set of L-states reachable from \( I[L] \), then in every decoupled state \( s^F \) reachable from \( s^0_F \), the set of reached L-states in \( s^F \) is \( S^L_R \).

**Proof:** In \( s^F \), the claim trivially holds. Let \( s^F \) be a (not necessarily direct) successor of \( s^0_F \). The center action that generates \( s^F \) can possibly restrict the set of compliant leaf states \( S^L_L \), and affect the remaining ones, resulting in a set of leaf states that is a subset of \( S^L_R \). Since \( S^R_L \) is strongly connected by \( \mathcal{A}^L_{CC} \), all L-states of \( S^L_R \) have a finite price in \( s^F \).  

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. Thus, at least for satisficing planning, these leaves do not need to be compared in the dominance check at all. For optimal planning, we still need to compare the prices, since these might differ.

Another minor optimization that can be performed with the leaf-invertibility information is successor generation during search. When computing the center actions that are applicable in a decoupled state, we usually need to check leaf preconditions 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, and skip the check for leaf preconditions on leaf-invertible factors.

### Transitivity of the Dominance Relation

In explicit state search, a state can be pruned if it has already been visited (with a lower g-value). This can be efficiently implemented using a hash table. In decoupled search with dominance pruning, the corresponding check needs to iterate over all previously visited states (with a lower g-value) that have the same center state, and compare the pricing function. Instead of iterating over all visited decoupled states, though, we can exploit the transitivity of our dominance relations to focus on the relevant visited states, namely those that are not themselves dominated by another visited state.

**Proposition 2** Let \( V \) be the set of decoupled states already visited during search and let \( t^F \) be a newly generated decoupled state. If there exist \( s^F_1, s^F_2 \in V \) such that \( s^F_1 \preceq s^F_2 \), where \( \preceq \) is a transitive relation over decoupled states, then \( t^F \not\preceq s^F_2 \) implies \( t^F \not\preceq s^F_1 \).

Clearly, we do not need to check dominance of \( t^F \) against \( s^F_1 \), but only need to compare \( s^F_2 \) and \( t^F \) to see if \( t^F \) can be pruned. During search, we incrementally compute the set of “dominated visited states” as a side product of the dominance check. If a new state \( t^F \) dominates an existing state \( s^F_1 \), then either there exists another visited state \( s^F_2 \) that dominates \( t^F \), so it will be pruned, or there is no state yet that dominates \( t^F \). In both cases, \( s^F_1 \) can be skipped in every future dominance check because there exists another state, either \( s^F_2 \) or \( t^F \), that is visited and that dominates it.

### Experimental Evaluation

We implemented all proposed methods in the decoupled search planner by Gnad & Hoffmann (2018), which itself builds on the Fast Downward planning system (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. We also run decoupled search to prove planning tasks unsolvable, using the benchmarks of UIPC’16 and Hoffmann, Kissmann, and Torralba (2014). In all benchmark sets, we eliminated duplicate instances that appeared in several iterations.

For optimal planning, we run blind search and A* with \( h^{\text{M-cal}} \) (Helmert and Domshlak 2009); in satisficing planning, we use greedy best-first search (GBFS) with the \( h^{\text{FF}} \) heuristic without preferred operator pruning (Hoffmann and Nebel 2001); to prove unsolvability, we run A* with the \( h^{\text{max}} \) heuristic (Bonet and Geffner 2001). The experiments were performed on a cluster of Intel E5-2660 machines running at 2.20 GHz with the common runtime/memory limits of 30min/4GiB. The code and experimental data of our evaluation are publicly available (Gnad 2021).
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, mostly the incident arcs factoring method (IA), and inverted-fork factorings (IF) – only for satisficing planning in Figure 7 (Gnad, Poser, and Hoffmann 2017). We expect IF factorings to 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 use the following naming convention for search configurations: we distinguish the three dominance relations $\preceq_B$, $\preceq_T$, and $\preceq_G$. We indicate the $g$-value adaptation, and the invertibility and transitivity optimizations by adding a superscript $g$, respectively $T$ and $G$ to the relation symbol, e.g. $\preceq_{T}^g$ for a configuration that uses $\preceq_G$ and has the invertibility and the transitivity optimization enabled. To save space, we focus on configurations that are most interesting in the comparison, omitting some that perform similarly.

Tables 1 and 2 show coverage data (number of instances solved) for the benchmarks where the factoring methods are able to detect a star factoring. For optimal planning, Table 1, we see that all methods individually can lead to an increase in coverage. There also seems to be a positive correlation between $\preceq_G$ and the $g$-value adaptation, shown by the fact that the combination outperforms both its components. We do not separately evaluate the invertibility optimization, since it only changes the successor generation, which does not influence coverage. The advantage of the $\preceq_{T}^g$ configurations stems from the transitivity optimization.

Duplicate checking without $g$-adaptation shows a significant drop in coverage compared to $\preceq_B$, so although the checking is computationally more efficient, this does not even out the weaker pruning power. When enabling the $g$-adaptation, total coverage is a lot higher than for $\preceq_B$, and even beats $\preceq_{T}^g$ in some domains (most remarkably in Miconic with blind search). $\preceq_{T}^g$ also solves two Freecell instances that no other configuration can solve with blind search. This shows that duplicate checking can indeed pay off if the pruning does not become a lot weaker.

Table 2 has coverage results for proving unsolvability (number of instances proved unsolvable) and satisficing planning. 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, mostly the incident arcs factoring method (IA), and inverted-fork factorings (IF) – only for satisficing planning in Figure 7 (Gnad, Poser, and Hoffmann 2017). We expect IF factorings to 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 use the following naming convention for search configurations: we distinguish the three dominance relations $\preceq_B$, $\preceq_T$, and $\preceq_G$. We indicate the $g$-value adaptation, and the invertibility and transitivity optimizations by adding a superscript $g$, respectively $T$ and $G$ to the relation symbol, e.g. $\preceq_{T}^g$ for a configuration that uses $\preceq_G$ and has the invertibility and the transitivity optimization enabled. To save space, we focus on configurations that are most interesting in the comparison, omitting some that perform similarly.

Tables 1 and 2 show coverage data (number of instances solved) for the benchmarks where the factoring methods are able to detect a star factoring. For optimal planning, Table 1, we see that all methods individually can lead to an increase in coverage. There also seems to be a positive correlation between $\preceq_G$ and the $g$-value adaptation, shown by the fact that the combination outperforms both its components. We do not separately evaluate the invertibility optimization, since it only changes the successor generation, which does not influence coverage. The advantage of the $\preceq_{T}^g$ configurations stems from the transitivity optimization.

Duplicate checking without $g$-adaptation shows a significant drop in coverage compared to $\preceq_B$, so although the checking is computationally more efficient, this does not even out the weaker pruning power. When enabling the $g$-adaptation, total coverage is a lot higher than for $\preceq_B$, and even beats $\preceq_{T}^g$ in some domains (most remarkably in Miconic with blind search). $\preceq_{T}^g$ also solves two Freecell instances that no other configuration can solve with blind search. This shows that duplicate checking can indeed pay off if the pruning does not become a lot weaker.

Table 2 has coverage results for proving unsolvability (number of instances proved unsolvable) and satisficing planning. 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, mostly the incident arcs factoring method (IA), and inverted-fork factorings (IF) – only for satisficing planning in Figure 7 (Gnad, Poser, and Hoffmann 2017). We expect IF factorings to 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 use the following naming convention for search configurations: we distinguish the three dominance relations $\preceq_B$, $\preceq_T$, and $\preceq_G$. We indicate the $g$-value adaptation, and the invertibility and transitivity optimizations by adding a superscript $g$, respectively $T$ and $G$ to the relation symbol, e.g. $\preceq_{T}^g$ for a configuration that uses $\preceq_G$ and has the invertibility and the transitivity optimization enabled. To save space, we focus on configurations that are most interesting in the comparison, omitting some that perform similarly.

Tables 1 and 2 show coverage data (number of instances solved) for the benchmarks where the factoring methods are able to detect a star factoring. For optimal planning, Table 1, we see that all methods individually can lead to an increase in coverage. There also seems to be a positive correlation between $\preceq_G$ and the $g$-value adaptation, shown by the fact that the combination outperforms both its components. We do not separately evaluate the invertibility optimization, since it only changes the successor generation, which does not influence coverage. The advantage of the $\preceq_{T}^g$ configurations stems from the transitivity optimization.

Duplicate checking without $g$-adaptation shows a significant drop in coverage compared to $\preceq_B$, so although the checking is computationally more efficient, this does not even out the weaker pruning power. When enabling the $g$-adaptation, total coverage is a lot higher than for $\preceq_B$, and even beats $\preceq_{T}^g$ in some domains (most remarkably in Miconic with blind search). $\preceq_{T}^g$ also solves two Freecell instances that no other configuration can solve with blind search. This shows that duplicate checking can indeed pay off if the pruning does not become a lot weaker.

Table 2 has coverage results for proving unsolvability (number of instances proved unsolvable) and satisficing planning. 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, mostly the incident arcs factoring method (IA), and inverted-fork factorings (IF) – only for satisficing planning in Figure 7 (Gnad, Poser, and Hoffmann 2017). We expect IF factorings to 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 use the following naming convention for search configurations: we distinguish the three dominance relations $\preceq_B$, $\preceq_T$, and $\preceq_G$. We indicate the $g$-value adaptation, and the invertibility and transitivity optimizations by adding a superscript $g$, respectively $T$ and $G$ to the relation symbol, e.g. $\preceq_{T}^g$ for a configuration that uses $\preceq_G$ and has the invertibility and the transitivity optimization enabled. To save space, we focus on configurations that are most interesting in the comparison, omitting some that perform similarly.

Tables 1 and 2 show coverage data (number of instances solved) for the benchmarks where the factoring methods are able to detect a star factoring. For optimal planning, Table 1, we see that all methods individually can lead to an increase in coverage. There also seems to be a positive correlation between $\preceq_G$ and the $g$-value adaptation, shown by the fact that the combination outperforms both its components. We do not separately evaluate the invertibility optimization, since it only changes the successor generation, which does not influence coverage. The advantage of the $\preceq_{T}^g$ configurations stems from the transitivity optimization.

Duplicate checking without $g$-adaptation shows a significant drop in coverage compared to $\preceq_B$, so although the checking is computationally more efficient, this does not even out the weaker pruning power. When enabling the $g$-adaptation, total coverage is a lot higher than for $\preceq_B$, and even beats $\preceq_{T}^g$ in some domains (most remarkably in Miconic with blind search). $\preceq_{T}^g$ also solves two Freecell instances that no other configuration can solve with blind search. This shows that duplicate checking can indeed pay off if the pruning does not become a lot weaker.
not affected in most domains, it increases by checking can pay off. While results are mixed in some, and state expansions for optimal planning.

Figure 4: Scatter plots comparing runtime and number of state expansions for optimal planning.

Figure 5: Like Figure 4, comparing dominance pruning to duplicate checking.

across two domains. For unsolvability, it looks like duplicate checking can pay off. While results are mixed in some, and not affected in most domains, it increases by 5 in OverFPP.

The scatter plots in Figure 4 and 5 shed further light on the per-instance runtime and search space size comparison between some optimal planning configurations. Figure 4 shows the number of expanded states in the top row and the runtime in the bottom row. All configurations use the IA factoring and compare $\leq B$ to $\leq G^{IT}$, with blind search in the left column and $\lambda^{LM}$ in the right column. The advantage of the more clever dominance check, the $g$-adaptation, and our runtime optimizations is obvious, saving up to several orders of magnitude for state expansions and runtime. Some domains with a particularly pronounced improvement across both settings are Elevators, Logistics, and Transport.

Figure 5 illustrates the effect of exact duplicate checking. The left plot shows the expected increase in search space size, due to the reduced pruning power. The right plot indicates that where the increase in search space size is small the more efficient computation indeed pays off runtime-wise. This is most visible in Miconic and Openstacks.

Figure 6: Improvement factors of $\leq y$ over $\leq y$ showing the impact of the transitivity optimization in optimal planning and proving unsolvability.

Figure 7: Like Figure 6, showing the impact of the invertibility optimization in satisficing planning with IA vs. IF.

Figures 6 and 7 show the impact of the transitivity and invertibility optimizations. The plots show per-instance runtime improvement factors of configuration $Y$ on the y-axis over configuration $X$ on the x-axis, where a $y$-value of $a$ indicates that the runtime of $Y$ is $a$ times the runtime of $X$ (values below 1 are a speed-up). The transitivity optimization clearly has a positive impact on runtime, reducing it up to 40% in optimal planning and up to 60% for proving unsolvability. The invertibility optimization (Figure 7) does not show such a clear picture when using the IA factoring (left plot). With IF, though, it indeed nicely accelerates the dominance check, as the optimization is applicable more often.

**Conclusion**

We have taken a closer look at the behavior and implementation details of dominance pruning in decoupled search. We introduced exact duplicate checking, which, in spite of its weaker pruning, can improve search performance in practice due to higher computational efficiency under certain conditions. Furthermore, we developed two optimizations that make the dominance check more efficient to compute.

Our main contribution are two extensions of dominance pruning for optimal planning, that incorporate the $g$-value of decoupled states. Both methods are highly beneficial and their combination significantly improves the performance of decoupled search in many benchmark domains.

For future work, we want to further investigate dominance pruning for decoupled search, e.g. by a combination with the quantitative dominance pruning of Torralba (2017).
Acknowledgements

Daniel Gnad was supported by the German Research Foundation (DFG), under grant HO 2169/6-2, “Star-Topology Decoupled State Space Search”. Jörg Hoffmann’s research group has received support by DFG grant 389792660 as part of TRR 248 (see perspicuous-computing.science).

References


