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 contains 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 many IPC 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 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 to 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 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).

Decoupled search only branches over actions that affect the center factor, so-called *center actions*, i. e., that have an effect on a variable in C. A decoupled state $s^{\mathcal{F}}$ reached by a sequence of center actions π^C, its *center path*, contains exactly those member states that are reachable in the task by paths that extend π^C with arbitrary *leaf actions*, i. e., actions that do not affect the center.

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 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 on 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 dominantes another 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 potential for pruning, because a decoupled state $s_1^{\mathcal{F}}$ with a much high 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. This is because dominance cannot be checked by standard means such as hashing. Instead, the prices of all leaf states need to be compared. We experiment with an implementation of exact duplicate checking, which, although less powerful, can benefit 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 literally 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 s was reached from I by the search. 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 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 leaf $L \in \mathcal{L}$ 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 $\mathsf{prices}(s_0^{\mathcal{F}})[s^L] = c(s_0^L)$ where $c(s_0^L)$ 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, then $c(s_0^L) = \infty$. 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_{compl}^L we define the set of leaf states of L that satisfy the leaf precondition of a^C, i. e., $S_{compl}^L := \{s^L \mid s^L \models \mathsf{pre}(a^C)[L] \wedge \mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty\}$. Applying a^C to $s^{\mathcal{F}}$ results in the decoupled state $t^{\mathcal{F}} = s^{\mathcal{F}}[\![a^C]\!]$ as follows: $\mathsf{center}(t^{\mathcal{F}}) := \mathsf{center}(s^{\mathcal{F}})[\![a^C]\!]$, $\mathsf{prices}(t^{\mathcal{F}})[t^L] := \min_{s^L \in S_{compl}^L}(\mathsf{prices}(s^{\mathcal{F}})[s^L] + c(u^L))$ where $s^L[\![a^C]\!] = u^L$, and $c(u^L)$ is the cost of a cheapest path of $\mathcal{A}^L|_{\mathsf{center}(t^{\mathcal{F}})} \setminus \mathcal{A}^C$ actions from u^L to t^L if such a path exists, else $c(u^L) = \infty$.

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 $g(s^{\mathcal{F}}) = \mathsf{cost}(\pi^C(s^{\mathcal{F}}))$ as the cost of its center path.

A decoupled state $s^{\mathcal{F}}$ can be interpreted as 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 in 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 during search. 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*, its g-value is not reduced – the new state can be pruned safely.

Prior work on decoupled search has only considered dominance pruning instead of exact duplicate checking during search (Torralba et al. 2016; Gnad and Hoffmann 2018). In dominance pruning, as the name suggests, instead of duplicate states, we prune 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 $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$, followed by a component-wise comparison of the prices of reached leaf states.

In the next subsection, we will look into exact duplicate checking for decoupled search, showing how an efficient hashing can be implemented. In the subsequent subsection, we 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

Since in decoupled search a search node, i. e., a decoupled state $s^{\mathcal{F}}$, does not represent a single state of the planning task, but a set of states, namely its hypercube $[s^{\mathcal{F}}]$, 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 somewhat small, even if explicit state search generates many duplicate states in a planning task. Furthermore, care must be taken when hashing decoupled states, properly taking into account both reachability *and* prices. Hence, 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, overapproximating reachability by projecting the task on 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 compute a unique ID for every leaf state, starting with 0, up to $|S^L_R| - 1$, where S^L_R 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 $L \in \mathcal{L}$ 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 ∞.

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 enables 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.

We identified an optimization that in some cases does not even require hashing when performing satisficing planning. The next paragraph describes this optimization.

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 in linear time in the size of such a graph if it is strongly connected, when only considering transitions of leaf actions that do not affected, nor are preconditioned by the center factor. 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 $I[L]$ 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 be reached from $I[L]$ are reached in $s^{\mathcal{F}}$. We call such leaf factors *leaf-invertible*.

Thus, 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 when doing satisficing planning, we can simply skip these leaves. For opti-

mal planning, however, we still need to compare the prices, since these might differ, so hashing is still required.

Checking if the leaf state spaces are indeed strongly connected can be done efficiently using Tarjan's algorithm (Tarjan 1972).

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 described in this paragraph 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 $\mathsf{prices}(t^{\mathcal{F}})[s^L]\mathsf{prices}(s^{\mathcal{F}})[s^L]$ for all leaf states $s^L \in S^{\mathcal{L}}$.

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 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 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 $s \in [s^{\mathcal{F}}] : \mathsf{cost}(s^{\mathcal{F}}, s) \geq \mathsf{cost}(t^{\mathcal{F}}, s)$. Say 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 from prior work, $s^{\mathcal{F}}$ cannot be pruned, because its prices are lower that the ones of $t^{\mathcal{F}}$. However, given the criterion from the last paragraph, $s^{\mathcal{F}}$ could actually be pruned, because the cost of all its member states is equal to the cost of the states in $t^{\mathcal{F}}$:

Things become more subtle if the prices differ in more leaf factors. We need to *split* 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 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.

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

In 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 not dominated by the prices of $t^{\mathcal{F}}$. There is even a difference of cost 2 left that could be used to trade higher prices if $t^{\mathcal{F}}$ in another leaf state.

In real search, the prices obviously are not the same for all leaf states of a factor. Hence, we use the minimum price of a reached leaf state for the calculations. This greatly increases the pruning potential for decoupled search, as we will see in the next section.

Experimental Evaluation

We implemented our optimizations, as well as the incorporation of g-values in the decoupled search planner by Gnad & Hoffmann (2018), which itself is build in the Fast Downward planning system (FD) (Helmert 2006). We conducted our experiments 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 standard runtime and memory limits of 30 minutes and 4GiB.

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 three basic factoring methods, namely inverted-fork factorings (IF) as well as the incident arcs factoring (IA) as described in Gnad et al. (2017).

We show experimental data for the following configurations: as a baseline (Base), we use the standard dominance pruning of prior work. We extend this with the incorporation

Domain	#	Blind Search Base	Blind Search gIn	Blind Search DUP	A* with $h^{\text{LM-cut}}$ Base	A* with $h^{\text{LM-cut}}$ gIn	A* with $h^{\text{LM-cut}}$ DUP
DataNetwork	20	5	9	5	12	14	12
Driverlog	20	11	11	10	13	13	13
Elevators08	30	9	9	10	22	22	23
Elevators11	20	7	7	8	18	18	18
Floortile11	20	1	2	0	6	6	4
Floortile14	20	0	0	0	2	4	2
Freecell	42	0	0	2	2	2	2
Logistics00	28	22	22	22	28	28	27
Miconic	145	45	46	42	135	135	135
NoMystery	20	17	19	16	20	20	19
Openstacks14	20	2	2	2	2	1	2
PSR	48	46	48	46	47	48	47
Woodworking08	28	7	7	7	16	17	16
Woodworking11	18	2	2	2	10	11	10
Zenotravel	20	8	8	7	12	12	11
Other	527	143	143	143	188	188	188
$\sum$	1026	325	335	322	533	539	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".

of g-values for optimal planning (gIn). For satisficing planning, we show a configuration that enables the invertibility-leaves optimizations (INV). For all search types, we show data where we only do exact duplicate checking as described above (DUP) with all optimizations. The gIn and DUP configurations for optimal planning also include the successor generation optimization, and the DUP variant includes the g-value improvement.

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, Tables 1, we see that the incorporation of g-values leads to an increase in coverage across several domains for both blind search and A* with $h^{\text{LM-cut}}$. There is only a single instance that cannot be solved any more with the new pruning enabled, which is solved just within the timeout for the Base variant. In total, we see an increase of $+10$ instances solved, distributed over 5 domains for blind search. When using $h^{\text{LM-cut}}$, the total coverage increases by 6 instances, again distributed over 5 domains.

The duplicate checking configuration does not loose a lot compared to gIn, which is quite surprising. As we will see in the plots in more detail later, although there is an increase in search space size and runtime, this often does not affect coverage. Interestingly, the reduction in coverage is highly domain-dependent. It is most pronounced in NoMystery and Miconic. Surprisingly, the duplicate checking variant can even solve some instances that the dominance pruning cannot solve. These are instances where the size of the search space does not increase (by a lot), where the more efficient hashing and the invertibility optimization pay off. This advantage disappears except for one instance in Elevators when using $h^{\text{LM-cut}}$.

Domain	#	no preferred Base	no preferred INV	no preferred DUP	preferred Base	preferred INV	preferred 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 data for satisficing planning using GBFS with h^{FF} and incident arcs factoring. The configurations on the left do not use preferred operator pruning, the ones on the right do. Domains without difference in coverage are summarized in "Other".

In satisficing planning (Table 2), the invertibilty optimization does not affect coverage. The duplicate checking configuration again looses a few instances in coverage, as before in NoMystery, and additionally in Floortile. The two instances not solved by INV with preferred operator pruning are again only solved close to the timeout by the base variant.

The scatter plots in Figure 1 and 2 shed further light on the runtime and search space size comparison between some configurations. Figure 1 shows the number of expanded states until the last f-layer in the top row and the runtime in the bottom row. All configurations use the incident arcs factoring, with blind search in the two leftmost columns and $h^{\text{LM-cut}}$ in the two rightmost columns.

Observe that the advanced dominance pruning reduces the search space size significantly, leading to a good runtime improvement for both search variants. The duplicate checking often increases search space size, and then also runtime, but often does not lead to a big increase.

In satisficing planning (Figure 2), we show the number of evaluated nodes in the top row and the runtime in the bottom. For both incident arcs, as well as inverted-fork factorings we see that, as expected, the optimizations do not influence the search space size. For the duplicate checking, the effect is less pronounced than in optimal planning. Interestingly, with inverted fork, we see a minor speed-up with the optimizations, which is not visible with incident arcs factoring. 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.

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 theory, can improve search performance in practice due to higher computational efficiency. We have shown two implementational optimizations that lead to this improvement. In satisficing planning, where duplicate/dominated states are seen not as regulary

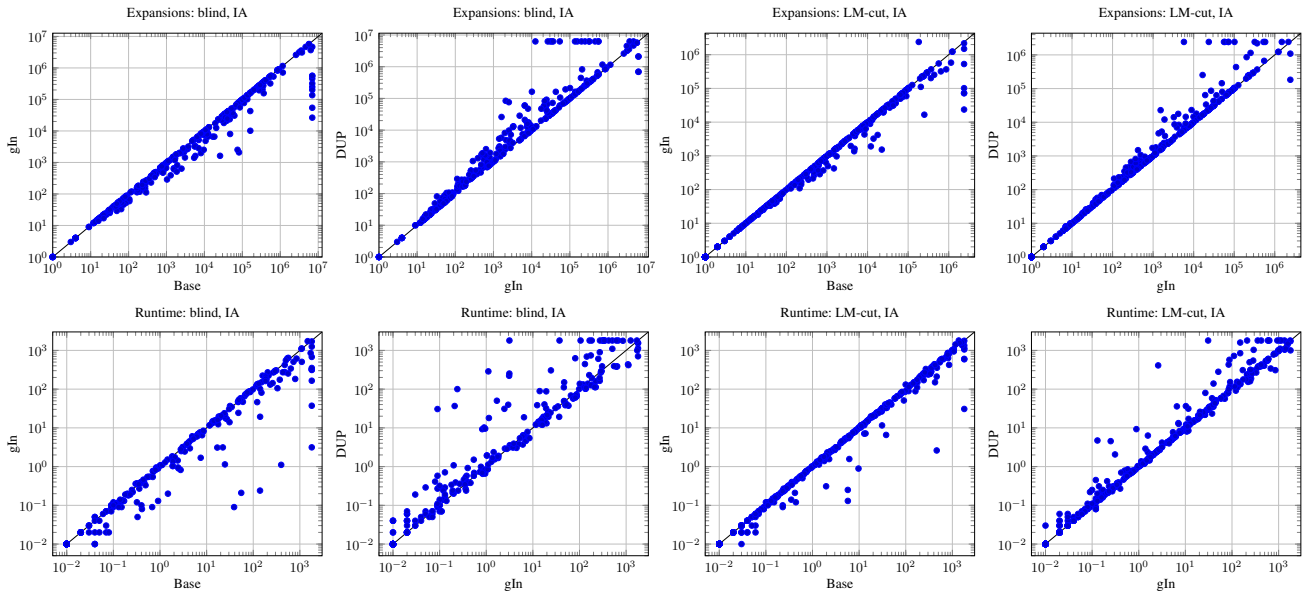

Figure 1: Optimal search with incident arcs factoring. The plots on the left show blind search, on the right A* with $h^{\text{LM-cut}}$. The top row shows the comparison of the number of expanded states until the last f-layer, the bottom row total time as reported by FD.

during search, it turns out that exact duplicate checking is competitive with dominance pruning. In optimal planning, however, the reduced pruning potential outweighs the increased efficiency, and in many IPC benchmarks the results get worse.

Further, we developed an extension of dominance pruning in optimal planning that takes into account the g-value of decoupled states. This has shown to be more powerful than standard dominance pruning by increasing the pruning potential at literally no negative effect on the runtime. This improves the performance of decoupled search across many benchmark domains and is an interesting new insight in the behavior of dominance pruning.

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 on the center factor could further increase the dominance 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

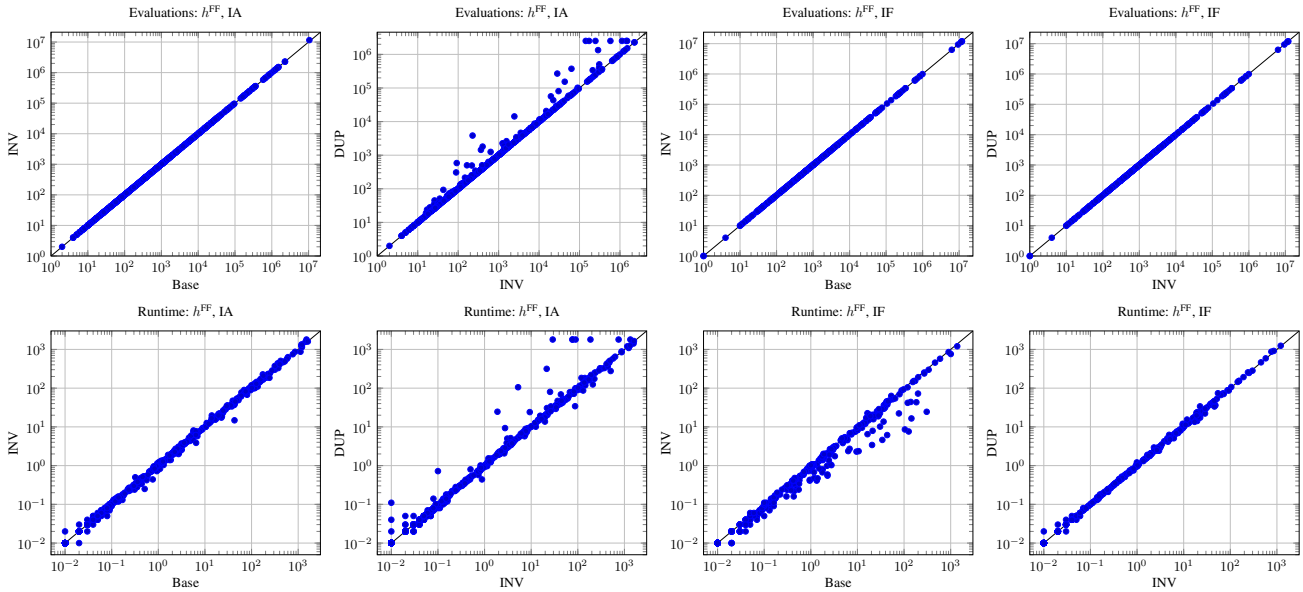

Figure 2: Satisficing search with GBFS and h^{FF}. Plots on the left show incident arcs factoring, on the right inverted fork factoring. The top row shows the comparison of the number of evaluated nodes, the bottom row total time as reported by FD.

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.

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.