Symbolic Leaf Representation in Decoupled Search

Daniel Gnad, Álvaro Torralba, Jörg Hoffmann
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{gnad,torralba,hoffmann}@cs.uni-saarland.de

Abstract

Star-Topology Decoupled Search has recently been introduced in classical planning. It splits the planning task into a set of components whose dependencies take a star structure, where one center component interacts with possibly many leaf components. Here we address a weakness of decoupled search, namely large leaf components, whose state space is enumerated explicitly. We propose a symbolic representation of the leaf state spaces via decision diagrams, which can be dramatically smaller, and also more runtime efficient. We further introduce a symbolic version of the LM-cut heuristic, that nicely connects to our new leaf representation. We show empirically that the symbolic representation indeed pays off when the leaf components are large.

Introduction

Classical planning is the task of checking whether there exists a sequence of (deterministic) actions that leads from a given initial state to a goal state, in a large state space described compactly through state variables. A prominent approach to tackle this is (heuristic) forward search.

Decoupled state space search is a recent paradigm to search reformulation (Gnad and Hoffmann 2015; Gnad, Hoffmann, and Domshlak 2015). It partitions the state variables into subsets (components), called *factors*. This is inspired by *factored planning* (e. g., Amir and Engelhardt (2003), Kelareva et al. (2007), Fabre et al. (2010), Brafman and Domshlak (2013)). Decoupled search proves to be particularly effective, through imposing a restriction on the interaction across factors: a star topology. In a star topology, a single center factor interacts with possibly many leaf factors, while no direct leaf-leaf interactions are allowed.

Such a topology entails a particular form of "conditional independence" between leaf factors, given a fixed move sequence by the center. Decoupled search branches over only the center-affecting actions. It enumerates, for each center-action sequence (each *center path*), the possible moves by the leaf factors – yet *for each leaf factor separately*.

To capture the possible movements of each leaf, the search maintains the so-called *pricing function*. Such a function maps each *leaf state* s^L of a given leaf factor (a value assignment to the leaf factor's variables) onto the cost of the cheapest sequence of leaf-affecting actions (the cheapest *leaf path*) that leads to s^L, and that is *compliant* with (can be embedded into) the given center path. In other words, the price of a leaf state is its cost given the current center path. The pricing function for a leaf factor can be maintained in time low-order polynomial in the size of the leaf factor's state space. This is done through a fixed-point operation after each application of a new center action, exploring all compliant leaf-path continuations. This works well for small leaf factors, but can be prohibitive for large leaves containing several state variables, thus severely limiting the kinds of factorings that can be usefully considered.

We address this through symbolic leaf-state-space representation. This results in a new hybrid of explicit and symbolic search, performing explicit search on the center component and symbolic search on the leaf state spaces.

Binary decision diagrams (BDDs) have been used in the past as an alternative to explicit search, as they allow to represent large state spaces compactly (Bryant 1986; McMillan 1993; Edelkamp and Helmert 1999). We build on prior work in planning that has derived efficient techniques to run symbolic search (Jensen, Veloso, and Bryant 2008; 2008; Kissmann and Edelkamp 2011; Torralba et al. 2017). But we use BDDs merely to maintain the leaf state spaces, not the entire search space.

Heuristic search is essential for the performance of forward search planning (e. g. (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010)). Gnad and Hoffmann (2015) introduced a compilation technique that allows to plug any planning heuristic function into decoupled search. The compilation does not suit our framework, however, as it requires to enumerate all leaf states. We overcome this here through a variant of the LM-cut heuristic (Helmert and Domshlak 2009), which is an admissible heuristic widely-used in planning. Our variant allows to avoid explicit enumeration, thus benefiting from our compact symbolic leaf representation.

We show that, on standard planning benchmark domains, the performance of our explicit-symbolic hybrid search is competitive with its explicit-search counterpart from prior work. On several domains, where leaves are large, our new techniques perform significantly better.

We provide further details in a technical report (Gnad, Torralba, and Hoffmann 2017).

Preliminaries

We use the finite-domain representation (FDR) of planning (e. g. Bäckström and Nebel (1995), Helmert (2006)), where a planning task is a tuple $\Pi = \langle V, A, I, G\rangle$. V is a set of finite domain *state variables*, where each $v \in V$ is associated with its domain $\mathcal{D}(v)$. A (partial) variable assignment is a set of variable/value pairs. A complete assignment to V is called a *state*. I is the *initial state*, and G is the *goal*, a partial assignment to V. A is a finite set of *actions*, where each action $a \in A$ is a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a)\rangle$. The *precondition* $\mathsf{pre}(a)$ and *effect* $\mathsf{eff}(a)$ are partial assignments to V, and $\mathsf{cost}(a) \in \mathbb{R}^{0+}$ is the action's non-negative *cost*.

For a partial assignment p, we denote the subset of state variables instantiated by p as $\mathcal{V}(p) \subseteq V$. For any $V' \subseteq \mathcal{V}(p)$, $p[V']$ denotes the value of all $v \in V'$ in p. An action a is *applicable* in a state s if $\mathsf{pre}(a) \subseteq s$, i. e., if for all $v \in \mathcal{V}(\mathsf{pre}(a)) : s[v] = \mathsf{pre}(a)[v]$. Applying a in s results in a state where the value of each $v \in \mathcal{V}(\mathsf{eff}(a))$ is changed to $\mathsf{eff}(a)[v]$, and the value of all other variables is unchanged. The outcome state is denoted $s[\![a]\!]$. The outcome of applying a sequence of (respectively applicable) actions to a state s is denoted $s[\![\langle a_1, \ldots, a_n\rangle]\!]$. An action sequence $\langle a_1, \ldots, a_n\rangle$ is called a *plan* for Π if it maps the initial state to a state satisfying G, i. e., $G \subseteq I[\![\langle a_1, \ldots, a_n\rangle]\!]$; it is *optimal* if its summed-up cost is minimal among all plans for Π.

Base Methods

Each of decoupled search and symbolic search can reduce search effort exponentially, though for entirely different reasons. As pointed out, our new technique can be understood as a hybrid of both. We briefly introduce both techniques.

Decoupled Search

In decoupled search, a planning task is split into a set of factors $\mathcal{F}$, partitioning its state variables. The dependencies across factors are required to take the form of a star, where a *center factor* F^C may arbitrarily interact with each of the set of *leaf factors* $\mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$, but no other interactions are allowed. The key observation is that, given this, for a fixed sequence of *center actions* (actions with an effect on F^C), the movements of the leaf factors are independent of each other, and can be maintained separately. Decoupled search then only branches over the center actions, enumerating the *compliant leaf paths* for each leaf. A leaf path π^L, i. e., a sequence of actions affecting a single $F^L \in \mathcal{F}^L$, is compliant with a given center path π^C, if it can be embedded into π^C such that the resulting action sequence is applicable to I when projecting onto the variables $F^C \cup F^L$.

Each center path π^C ends in a *decoupled state* $s^\mathcal{F}$, which is defined as a *center state* $cs(s^\mathcal{F})$, a complete assignment to F^C, and a *pricing function* $prices(s^\mathcal{F})$. The pricing function maps each leaf state s^L (an assignment to an $F^L \in \mathcal{F}^L$) to a non-negative *price* $prices(s^\mathcal{F})[s^L]$, namely the cost of a cheapest leaf path to s^L compliant with π^C. If no such path exists for s^L, its price is $prices(s^\mathcal{F})[s^L] = \infty$. A decoupled state thus consists of a single center state, augmented with a set of leaf states for each F^L, annotated by their price.

Pricing functions are maintained as follows. To expand a decoupled state $s^\mathcal{F}$, **(1) the applicable center actions are obtained** by checking if their center precondition holds in $s^\mathcal{F}$; and by checking, whenever $\mathcal{V}(\mathsf{pre}(a^C)) \cap F^L \neq \emptyset$, if there exists a reached leaf state s^L that satisfies this precondition (*CheckPre*$(s^\mathcal{F}, a^C)$), i. e., where $\mathsf{pre}(a^C)[\mathcal{V}(\mathsf{pre}(a^C)) \cap F^L] \subseteq s^L$, and $prices(s^\mathcal{F})[s^L] < \infty$. **(2) Applying a center action** a^C to a decoupled state $s^\mathcal{F}$ (*Apply*$(s^\mathcal{F}, a^C)$) results in a new decoupled state $t^\mathcal{F}$ that is a copy of $s^\mathcal{F}$. In $t^\mathcal{F}$ the price of all inconsistent leaf states s^L, where $\mathsf{pre}(a^C)[\mathcal{V}(\mathsf{pre}(a)) \cap F^L] \not\subseteq s^L$, is set to ∞, and the leaf effects of a^C are applied to all remaining leaf states with finite price. Finally, **(3) the pricing function is updated** (*UpdatePrices*$(s^\mathcal{F})$), by computing the continuations of the cheapest compliant paths given the new leaf actions now enabled. To avoid duplicate states, decoupled search uses a **(4) dominance check** (*CheckDominance*$(s^\mathcal{F}, t^\mathcal{F})$). A newly generated decoupled state $s^\mathcal{F}$ is dominated by a previously seen $t^\mathcal{F}$ if $cs(s^\mathcal{F}) = cs(t^\mathcal{F})$ and, for all leaf states s^L, $prices(t^\mathcal{F})[s^L] \leq prices(s^\mathcal{F})[s^L]$.

Symbolic Search

Symbolic search is a state space exploration technique that uses efficient data structures to represent and manipulate sets of states (McMillan 1993). *Binary Decision Diagrams* (BDDs), in particular, often yield exponential gains compared to explicit enumeration (Bryant 1986). A number of operations allow to manipulate BDDs efficiently.

To perform search, planning actions are represented via *transition relations* (TRs). A TR T represents a set of actions $A' \subseteq A$ of the same cost, through a BDD that contains the set of all pairs (s, s') such that s' is reachable from s by applying an action $a \in A'$. Given a set of states S and a TR T, the *image* operation computes the set of successor states of S through T. The image operation has worst-case complexity exponential in the number of state variables, but is often more efficient than expanding the states in S one by one. The symbolic variant of standard search algorithms is implemented by starting from the BDD representation of the initial state, and iteratively computing the image.

Algebraic Decision Diagrams (Bahar et al. 1997) are an extension of BDDs that represent functions mapping each state to a value among a finite set of different values. In the context of planning, they have been used to represent heuristic functions and to perform A* search (Hansen, Zhou, and Feng 2002).

Symbolic Leaf Representation

When dealing with leaves that have a small state space, the pricing function can be kept explicitly. This requires a single entry per reachable leaf state that stores its price. Globally caching transitions between leaf states then allows to efficiently update the pricing function. This becomes prohibitive, however, both memory and runtime wise, for large leaf state spaces. We propose to use a symbolic representation of the pricing function, using decision diagrams (DDs).

We use different types of DDs to address the different requirements of the operations working on the pricing function. For each leaf F^L and price p, we keep a BDD B^L_p that

represents all leaf states of F^L with price p. Additionally, we compute a BDD B^L_R that represents all leaf states of F^L reached with finite price, as well as an ADD P^L that represents all B^L_p in a single data structure. As we will describe next, different data structures ease the computation of certain of the previously detailed operations required to perform decoupled search. For efficiency reasons, these operations must be performed directly on the symbolic representation, without enumerating all leaf states at any point, so that the complexity of each operation depends on the DD size, not on the size of the represented leaf state space. Additionally, due to the independence of the leaves, the operations are always performed for each individual leaf separately. We next describe how to do so using standard BDD and ADD operations.

To compute *CheckPre*$(s^\mathcal{F}, a^C)$ of a center action a^C, we encode the leaf preconditions of a^C on a leaf F^L as a BDD B^L_{pre} that describes the set of leaf states that satisfy such preconditions. Then, a^C is applicable if the intersection of B^L_{pre} and the set of reached leaf states B^L_R is not empty.

Apply$(s^\mathcal{F}, a^C)$ applies a^C to the pricing function of a decoupled state $s^\mathcal{F}$. If a^C has preconditions on a leaf, we compute the intersection of the BDDs B^L_p representing $prices(s^\mathcal{F})$ with B^L_{pre}. If a^C has an effect on the leaf, this can be applied to each B^L_p using the standard image operation with respect to the TR of a^C projected on F^L.

UpdatePrices$(s^\mathcal{F})$ is a fundamental operation in decoupled search. The price updates correspond to performing a symbolic uniform-cost search for every leaf factor $F^L \in \mathcal{F}^L$, using only those TRs corresponding to leaf actions with an effect on F^L whose center preconditions are satisfied by $cs(s^\mathcal{F})$. The open list of this search is initialized to the previous pricing function, i. e. the B^L_p, inserting a leaf state s^L with a g-value of $prices(s^\mathcal{F})[s^L]$. The search is run until the open list is empty, exhausting the leaf state space reachable with the center preconditions provided by $cs(s^\mathcal{F})$. After the search, the closed list represents the desired pricing function, with a new BDD B^L_p for each cost layer containing the set of leaf states with this price.

The computation of *CheckDominance*$(s^\mathcal{F}, t^\mathcal{F})$ makes use of the ADD representation of pricing functions. Dominance corresponds to the standard "lower or equal" operation on ADDs, which checks whether one ADD has lower or equal value than the other for every possible assignment.

The described implementation is suitable for optimal planning, where we need to keep the price of each leaf state. When cost is not of interest, e. g., for satisficing planning, or proving unsolvability, there is no need to keep the actual price of every leaf state. The only thing that matters is whether a state is reachable, or not. This corresponds to a task transformation where the cost of all leaf actions is set to 0, so the pricing function of any leaf state is either 0 or ∞. Therefore, it suffices to keep a single BDD B^L_R to represent the reached leaf states of each leaf factor. The operations are similar, except that updates can be done with the simpler symbolic breadth first search, and dominance is performed by checking whether one BDD is a subset of another.

Connecting Symbolic Leaves to Heuristics

In previous work, Gnad and Hoffmann (2015) introduced a compilation that allows the usage of standard heuristics in decoupled search. This compilation depends on the decoupled state $s^\mathcal{F}$ for which the heuristic is computed. The key is to add a new *auxiliary action* $a^L_{s^L}$ for each reached leaf state s^L with empty precondition, effect s^L, and whose cost is $prices(s^\mathcal{F})[s^L]$. Thereby, the heuristic has to "buy" a leaf state before being able to perform any other operation on the variables of the corresponding leaf. Gnad and Hoffmann implemented the compilation for many delete-relaxation-based heuristics, like $h^{\max}$ (Bonet and Geffner 2001), h^{FF} (Hoffmann and Nebel 2001), or $h^{\text{LM-cut}}$, to which this concept naturally applies. However, it does not easily extend to the symbolic leaf representation, since we must avoid the explicit enumeration of leaf states. We re-use the ADD representation of the pricing function to compute the heuristic.

A rather easy case is $h^{\max}$, where it suffices to add an auxiliary action for each leaf fact. We set the cost of these actions to the minimum price of any leaf state containing that fact. This can be computed easily for each leaf by a single traversal over the ADD that represents its pricing function. Intuitively, we pretend that the $h^{\max}$ algorithm does not start from a full variable assignment, but a "delete-relaxed" extension thereof. This extended state contains the center facts reached in the given decoupled state $s^\mathcal{F}$ (at a cost of 0), as well as possibly many facts per leaf variable, at the cost of the minimum price leaf state that contains the fact in $s^\mathcal{F}$.

The case of LM-cut is more complicated. The heuristic iteratively runs $h^{\max}$, and computes a disjunctive action landmark (Zhu and Givan 2003; Hoffmann, Porteous, and Sebastia 2004; Richter, Helmert, and Westphal 2008; Richter and Westphal 2010) after each iteration. This landmark is called the *cut*, a set of non-zero cost actions at least one of which must be applied in any plan. The cost of the actions in the cut is decreased by the minimum c_{min} of their costs. This process is repeated until the value of $h^{\max}$ is 0. The final $h^{\text{LM-cut}}$ value is the sum of all c_{min}.

The computation of $h^{\max}$ within LM-cut can be performed as described above, by adding an auxiliary action per leaf fact. The difficulty is to determine for which of the auxiliary actions we have to reduce the cost in each iteration. For single-variable leaves, this is just the set of actions in the cut, as usual. For multi-variable leaves, however, there is no auxiliary-fact-action to leaf-state mapping, and it can be necessary to reduce the cost of some auxiliary actions that are not in the cut. For example, consider a planning task with a single leaf of two variables, where $D(v_1) = \{q_1, q'_1\}$, $D(v_2) = \{q_2, q'_2\}$. Let $s^\mathcal{F}$ be a decoupled state with two finite-price leaf states, $prices(s^\mathcal{F})[\{q_1, q_2\}] = 0$, $prices(s^\mathcal{F})[\{q'_1, q'_2\}] = 1$. The goal is $\{q'_1, q'_2\}$, so $h^*(s^\mathcal{F}) = 1$. Say the first iteration of LM-cut finds the cut $\{a_{v_1,q_1}\}$ with a single auxiliary action, so $c_{min} = 1$. In this case, the cost of the auxiliary action a_{v_2,q_2} must be reduced as well, because otherwise the cost of acquiring the same leaf state would be counted more than once, resulting in an inadmissible heuristic. This is because the cost of both a_{v_1,q_1}, and a_{v_2,q_2} is due to the same leaf state. Thus, de-

creasing the cost of one auxiliary-fact-action may cause a reduced cost of other auxiliary actions.

Instead of directly reducing the cost of the auxiliary actions involved in the cut, we reduce the prices of the corresponding states in the symbolic pricing function, and recompute the cost of the auxiliary actions with respect to the new pricing function. Given a cut, we construct an ADD that assigns a value of c_{min} to all leaf states containing a fact whose corresponding auxiliary action is in the cut and 0 elsewhere. We subtract this ADD from the pricing function and recompute the cost of each auxiliary action by a new traversal over the resulting ADD. So whenever the cost of an auxiliary action a_q for a leaf fact q is decreased by c_{min}, we subtract c_{min} from every leaf state that satisfies q. This results in a non-negative price, i. e., $P^L[s^L] - c_{min} \geq 0$ for all s^L, because c_{min} is the minimum action cost of all actions in the cut so $c_{min} \leq \mathsf{cost}(a_q) \leq P^L[s^L]$.

Furthermore, this procedure will reduce the price of exactly those leaf states containing a fact whose corresponding auxiliary leaf actions would be in the cut when using the explicit LM-cut implementation (modulo different tie-breaking). This is due to the fact that each leaf fact q in the initial layer of LM-cut's landmark graph can only result from the application of an auxiliary leaf action. So, if the auxiliary achiever of q is in the cut in the symbolic version, so are all auxiliary leaf actions whose effect contains q in the explicit variant, and vice versa.

Experiments

Our techniques are implemented in Fast Downward (Helmert 2006), extending the decoupled search implementation of Gnad et al. (2015). We use the *CUDD* library by Fabio Somenzi to store and manage the symbolic leaf representations. Planners are run on a cluster of Intel E5-2660 machines with 2.20 GHz, using time (memory) cut-offs of 30 minutes (4 GB). We conduct experiments in optimal planning, on all IPC STRIPS benchmarks (1998 – 2014).

We use Gnad et al.'s (2017) *incident arcs* (**IA**) factoring strategy and a modification thereof (**mIA**), which both greedily compute a factoring by putting variables with many causal graph dependencies to the center, and setting the leaves to be the connected components in the remaining part of the causal graph. Both return the factoring with the highest number of *mobile* leaves, i. e., those that have at least one leaf-only action. Gnad et al.'s method breaks ties by choosing the factoring with the largest center factor, i. e., with small leaves, our new variant chooses the one with the smallest center, to illustrate the benefit of our symbolic representation with large leaf factors. Like Gnad et al., we *abstain* from solving the task if the obtained factoring has less than two (mobile) leaf factors, because the potential gain of decoupled search is exponential in the number of leaves.

Table 1 shows coverage results for our new symbolic leaf representation of both factoring strategies (**IAS/mIAS**) compared to the explicit variants, standard search (**A***), symbolic forward search (**S**), and the SymBA* planner (**SB**) (Torralba, Linares López, and Borrajo 2016). To get a fair comparison, we disabled the h^2-preprocessor of SymBA* (Alcázar and Torralba 2015). The table includes

		Blind Search						$h^{\text{LM-cut}}$					
Domain	#	A*	IA	IAS	mIA	mIAS	S	A*	IA	IAS	mIA	mIAS	SB
Chsnack	20	0	0	0	0	0	**4**	0	0	0	0	0	**3**
Depots	22	4	4	4	4	4	4	**7**	**7**	**7**	**7**	**7**	5
Driverl	20	7	**11**	**11**	10	**11**	**11**	13	13	13	12	12	**14**
Elevator	50	26	16	19	16	19	**35**	40	40	40	40	40	**44**
Floortile	40	2	2	2	2	2	2	13	8	8	8	8	**34**
GED	20	15	15	15	13	15	15	15	15	15	15	15	**19**
Logistic	63	12	**24**	**24**	**24**	**24**	20	26	**35**	**35**	**35**	**35**	25
Miconic	145	50	45	46	45	46	**91**	**136**	135	135	135	135	102
Mystery	1	1	1	1	1	1	1	1	1	1	1	1	1
NoMyst	20	8	**19**	**19**	**19**	**19**	11	14	**20**	**20**	**20**	**20**	15
Openst	70	42	35	36	34	35	**70**	40	35	35	34	34	**70**
Pathw	30	4	4	4	4	4	5	5	5	5	5	5	5
PSR	48	47	46	46	46	46	**48**	47	47	47	47	47	**48**
Satellite	36	6	5	5	5	5	**7**	7	**9**	**9**	**9**	**9**	8
Scanaly	14	**10**	4	4	4	4	**10**	8	8	8	8	8	**10**
Tetris	13	**7**	5	6	5	5	**7**	4	**5**	**5**	**5**	**5**	**5**
Transp	36	**19**	17	17	17	17	**19**	18	18	18	18	18	**19**
Trucks	27	5	4	4	4	4	**8**	9	**10**	**10**	**10**	**10**	9
Zenotr	20	**8**	6	**8**	7	**8**	**8**	**13**	11	12	11	12	10
$\sum$	695	273	263	271	260	269	**376**	416	422	423	420	421	**446**
Freecell	42	**3**	0	2	0	2	2	**2**	1	**2**	1	**2**	**2**
Grid	5	**1**	**1**	**1**	0	**1**	**1**	**2**	**2**	**2**	0	**2**	**2**
Mprime	6	**6**	4	**6**	3	**6**	**6**	**6**	4	**6**	3	**6**	**6**
ParcP	23	1	5	7	7	**11**	3	7	**13**	**13**	11	9	3
Rovers	40	6	7	8	7	9	**13**	7	9	10	8	10	**13**
Tidybot	40	8	**19**	**19**	**19**	**19**	15	22	**24**	**24**	23	**24**	6
TPP	29	5	5	5	**23**	**23**	7	5	5	5	**18**	**18**	7
Woodw	43	9	9	12	13	14	**23**	23	26	27	29	**31**	30
$\sum$	228	39	50	60	72	**85**	70	74	84	89	93	**102**	69
$\sum$	923	312	313	331	332	354	**446**	490	506	512	513	**523**	515

Table 1: Coverage data (number of solved instances). Best performance in **bold face**. The top part shows domains with small leaves, domains with large leaves are in the bottom.

data for blind search, and search with $h^{\text{LM-cut}}$. We split it into two parts, where the upper one contains domains with small leaves – estimating the size of a leaf F^L by the domain size product of its variables. The lower part contains domains with large leaves, i. e., at least one solved instance has an average leaf size ≥ 1000 when using mIA. In the top part, we see that the coverage is only little influenced by the leaf representation. In most domains, the coverage does not change – even less when using $h^{\text{LM-cut}}$, where only in Zenotravel the coverage differs by $+1$ instance. With blind search, the highest difference occurs in Elevators, where the symbolic leaf representation inherits the apparent strength of pure symbolic search. In the bottom part, observe that using the symbolic representation clearly pays off if the leaves are large. The only negative outlier is ParcPrinter, where we loose coverage with mIAS and $h^{\text{LM-cut}}$, probably due to the computational overhead of our symbolic LM-cut implementation. In all other domains except TPP the symbolic representation gains coverage, often by 2 to 4 instances.

Figure 1 sheds more light on the runtime performance of the two representations. Observe that there is an initial overhead of the symbolic leaf variant, which is mostly due to the initialization of the symbolic data-structures. With blind search (left), we see the benefit of the symbolic leaves for non-trivially solved tasks (above around 20s). While for small leaves (the black points), there is just a minor improvement, the advantage is clearly visible for instances with large

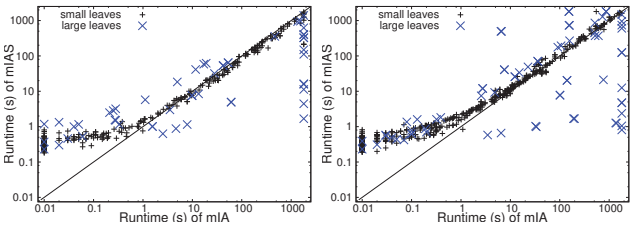

Figure 1: The plots show per-instance runtime comparisons using blind search (left) and $h^{\text{LM-cut}}$ (right). Instances with large leaves are highlighted in blue.

leaves. This is also true when using $h^{\text{LM-cut}}$, though coming at an increased risk, where the overhead of symbolic LM-cut can outweigh the gain from the leaf representation.

We also conducted experiments in proving unsolvability. Here, the picture is similar to optimal planning, but most domains of the unsolvability IPC have only small leaves.

Conclusion

We have introduced a new hybrid of explicit and symbolic state space search, within the framework of star-topology decoupled search. Empirically, our approach works well in planning domains with large leaf factors. Having derived a well-working adaptation for LM-cut, a topic for future work is to connect our symbolic representation to other heuristic functions, and to other improvements of decoupled search, like partial-order reduction (Gnad, Wehrle, and Hoffmann 2016) and structural symmetries (Gnad et al. 2017).

Acknowledgments. D. Gnad was supported by the German Research Foundation (DFG), grant HO 2169/6-1. Á. Torralba was supported by the German Federal Ministry of Education and Research (BMBF), CISPA grant no. 16KIS0656.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *"Proc. of ICAPS"*, 2–6.

Amir, E., and Engelhardt, B. 2003. Factored planning. In *"Proc. of IJCAI"*, 929–935.

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

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebraic decision diagrams and their applications. *Formal Methods in System Design* 10(2/3):171–206.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Brafman, R., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

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

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *"Proc. of ECP"*, 135–147.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-optimal factored planning: Promises and pitfalls. In *"Proc. of ICAPS"*, 65–72.

Gnad, D., and Hoffmann, J. 2015. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In *"Proc. of ICAPS"*, 88–96.

Gnad, D.; Torralba, Á.; Shleyfman, A.; and Hoffmann, J. 2017. Symmetry breaking in star-topology decoupled search. In *"Proc. of ICAPS"*.

Gnad, D.; Hoffmann, J.; and Domshlak, C. 2015. From fork decoupling to star-topology decoupling. In *"Proc. of SOCS"*, 53–61.

Gnad, D.; Poser, V.; and Hoffmann, J. 2017. Beyond forks: Finding and ranking star factorings for decoupled search. In *"Proc. of IJCAI"*.

Gnad, D.; Torralba, Á.; and Hoffmann, J. 2017. Symbolic leaf representation in decoupled search – technical report. Technical report, Saarland University. Available at *http://fai.cs.uni-saarland.de/hoffmann/papers/socs17a-tr.pdf*.

Gnad, D.; Wehrle, M.; and Hoffmann, J. 2016. Decoupled strong stubborn sets. In *"Proc. of IJCAI"*, 3110–3116.

Hansen, E. A.; Zhou, R.; and Feng, Z. 2002. Symbolic heuristic search using decision diagrams. In *"Proc. of SARA"*, 83–98.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *"Proc. of ICAPS"*, 162–169.

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.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Jensen, R. M.; Veloso, M. M.; and Bryant, R. E. 2008. State-set branching: Leveraging BDDs for heuristic search. *Artificial Intelligence* 172(2-3):103–139.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored planning using decomposition trees. In *"Proc. of IJCAI"*, 1942–1947.

Kissmann, P., and Edelkamp, S. 2011. Improving cost-optimal domain-independent symbolic planning. In *"Proc. of AAAI"*, 992–997.

McMillan, K. L. 1993. *Symbolic Model Checking*. Kluwer Academic Publishers.

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

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *"Proc. of AAAI"*, 975–982.

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

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *"Proc. of IJCAI"*, 3272–3278.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.